МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ
ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ
ФАКУЛЬТЕТ №2
Кафедра інформаційних технологій

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ЗВЕРНЕНЬ ГРОМАДЯН У ВІДДІЛІ ПОЛІЦІЇ.

кваліфікаційна робота
здобувача вищої освіти
4 курсу денної форми навчання
Романа МАРЧУКА

Науковий керівник:
доцент, кандидат технічних наук
Тарас РУДИЙ

Рецензент:
доцент, кандидат технічних наук
Ігор КРОШНИЙ

Кваліфікаційна робота допущений до захисту
«__» _____ 2026 р., протокол № ____

Завідувач кафедри інформаційних технологій
_____ **Олег ЗАЧЕК**
(підпис)

Львів

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система

БД – база даних

СКБД – система керування базами даних

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

Р

r

o

g

r

a

m

m

i

n

g

S

t

r

u

c

t

u

r

e

SPA (Single Page Application) – односторінковий Web-додаток

ІПНП – інформаційний портал Національної поліції України

UX (User Experience) – досвід користувача

JSON Web Token) – токен для автентифікації користувачів

RBAC (Role-Based Access Control) – модель розмежування доступу за ролями

Interface) – інтерфейс програмування застосунків

CORS.(Cross-Origin Resource Sharing) – механізм контролю доступу між сервісів доменами

АНОТАЦІЯ

МАРЧУК Р. Розроблення інформаційної системи обліку звернень громадян у підрозділах Національної поліції України. – Рукопис.

Дослідження на здобуття освітнього ступеня «бакалавр» за спеціальністю 126 «Інформаційні системи та технології». – Львівський державний університет внутрішніх справ, МВС України, Львів, 2026.

У даній роботі здійснено розроблення Web-орієнтованої інформаційної системи обліку звернень громадян у підрозділах Національної поліції України. У ході виконання роботи проведено аналіз нормативно-правової бази, що регулює процес обробки звернень громадян, а також досліджено існуючі інформаційні системи та традиційні підходи до ведення обліку. Визначено їх основні недоліки, зокрема відсутність прозорості, обмежену доступність для громадян та низький рівень автоматизації процесів.

У роботі обґрунтовано доцільність використання Web-орієнтованої клієнт-серверної архітектури, яка забезпечує доступ до системи у режимі реального часу та дозволяє підвищити ефективність взаємодії між громадянами та правоохоронними органами. Запропоновано функціональну модель системи, що передбачає можливість створення, обробки, фільтрації та відстеження звернень, а також розмежування прав доступу між користувачами.

Практична частина роботи полягає у розробці програмного забезпечення із використанням сучасних технологій, зокрема React для клієнтської частини, Node.js для серверної логіки та SQLite для зберігання даних. Реалізовано механізми автентифікації користувачів на основі JWT, систему фільтрації та пошуку звернень, зміну статусів та взаємодію через REST API.

Ключові слова: інформаційна система, звернення громадян, Web-додаток, REST API, JWT, клієнт-серверна архітектура.

Marchuk R. Development of an Information System for Citizens' Appeals Management in the National Police of Ukraine. – Manuscript.

Research for obtaining the bachelor's degree in specialty 126 "Information Systems and Technologies". – Lviv State University of Internal Affairs, Ministry of Internal Affairs of Ukraine, Lviv, 2026.

This paper presents the development of a web-based information system for managing citizens' appeals within the National Police of Ukraine. The study includes an analysis of the legal framework governing the processing of citizens' appeals, as well as a review of existing information systems and traditional methods used in this domain. The main shortcomings of existing approaches are identified, including lack of transparency, limited accessibility for citizens, and insufficient level of process automation.

The research substantiates the feasibility of using a web-based client-server architecture, which provides real-time access to the system and improves interaction between citizens and law enforcement agencies. A functional model of the system is proposed, including features for creating, processing, filtering, and tracking appeals, as well as role-based access control.

The practical part involves the development of software using modern technologies, including React for the frontend, Node.js for backend logic, and SQLite as the database. JWT-based authentication, appeal filtering and search mechanisms, status management, and REST API interaction are implemented. A key feature of the system is the combination of usability, functionality, and scalability for further development.

Keywords: information system, citizens' appeals, web application, REST API, JWT, client-server architecture.

Зміст

ВСТУП

Нинішній період становлення інформаційного суспільства в Україні відзначається інтенсивною реалізацією концепції «держава у смартфоні» та масовим впровадженням діджитал-інструментів у діяльність держструктур. Ключовим вектором цифрової трансформації визначено оновлення системи

взаємодії правоохоронних інституцій із громадськістю, що передбачає вдосконалення механізмів подання та обробки звернень громадян в електронному форматі.

Функціонування територіальних підрозділів поліції часто супроводжується надмірною бюрократизацією та орієнтацією на паперові носії інформації. Необхідність особистої присутності заявника у відділку та ручне ведення журналів реєстрації підвищують вірогідність технічних помилок, втрати відомостей та ускладнюють моніторинг дотримання встановлених строків. Брак ефективних каналів зворотного зв'язку формує негативне ставлення громадян до якості роботи поліції. Саме тому впровадження вебтехнологій для діджиталізації цієї сфери є нагальною потребою сьогодення.

Існуючі інформаційні системи, такі як ІПНП (Інформаційний портал Національної поліції) або ЄРДР, орієнтовані насамперед на внутрішні процесуальні потреби та аналітику. Вони є закритими контурами, що забезпечують збереження та обробку даних, проте не надають громадянам зручних інструментів для віддаленого моніторингу стану розгляду їхніх звернень. Це обмежує можливості оперативного інформування заявників та створює додаткове навантаження на чергові частини.

Саме тому виникає потреба у розробці спеціалізованої інформаційної системи, яка поєднує функції внутрішнього обліку для співробітників поліції та публічного доступу для громадян. Реалізування такої системи у вигляді Web-додатка (Web Application) забезпечує кросплатформеність, дозволяючи подавати звернення як з комп'ютера, так і з мобільного пристрою, без необхідності встановлення спеціалізованого програмного забезпечення.

Мета кваліфікаційної роботи полягає у розробці інформаційної системи обліку звернень громадян у відділі поліції, що забезпечує автоматизацію процесів реєстрації, розподілу, контролю виконання звернень та налагодження ефективного зворотного зв'язку із заявниками.

Для досягнення мети роботи сформульовано наступні завдання: Провести аналіз предметної області та процедур обробки звернень громадян у підрозділах Національної поліції.

- Дослідити існуючі електронні сервіси та виявити недоліки поточних методів взаємодії з населенням.
- Сформувати вимоги до функціональності системи, враховуючи розподіл ролей (громадянин, офіцер, адміністратор).
- Спроєктувати архітектуру бази даних та клієнт-серверну взаємодію (API).
- Реалізувати Web-додаток, який забезпечує повний цикл обробки звернення: від подачі до звіту про виконання.
- Забезпечити належний рівень захисту інформації користувачів та реалізувати механізми авторизації.
- Оцінити ефективність впровадження системи в роботу відділу поліції.

Об'єктом дослідження є сукупність процесів інформаційного обміну між представниками громадськості та структурами Національної поліції України у сфері подання та опрацювання офіційних звернень.

Предмет дослідження - методи та програмні засоби автоматизації обліку звернень та контролю їх виконання на основі Web-технологій.

У процесі виконання роботи застосовано системний підхід до формування функціональних вимог, а також використано методи моделювання під час проєктування архітектури бази даних і алгоритмів взаємодії. Практична частина роботи реалізована з використанням сучасних підходів до розробки програмного забезпечення, зокрема принципів об’єктно-орієнтованого та функціонального програмування на основі стеку JavaScript (Node.js, React). Перевірка коректності роботи системи здійснювалася шляхом тестування основних функціональних сценаріїв із використанням сучасних підходів до забезпечення якості програмного забезпечення.

Розроблена система має практичну цінність, оскільки реалізує працездатний прототип, який сприяє скороченню часу обробки звернень, зменшенню обсягу паперової документації, зниженню навантаження на підрозділи поліції та покращенню відкритості їх діяльності для громадян.

Структура дипломної роботи охоплює вступ, чотири розділи, підсумкові висновки та перелік використаних джерел. У першому розділі здійснено дослідження предметної області та аналіз існуючих рішень. Другий розділ присвячено проєктуванню архітектури системи та структури бази даних. У третьому розділі розглянуто реалізацію програмної частини, включаючи серверну та клієнтську складові. У четвертому розділі наведено оцінку ефективності запропонованого рішення та визначено напрями подальшого розвитку.

Загальний обсяг роботи становить 53 сторінки. У роботі представлено 19 ілюстрацій та 4 таблиці. Для підготовки дослідження використано 11 джерел.

У роботі розглянуто теоретичні та практичні аспекти розроблення інформаційної системи, що дозволяє всебічно підійти до реалізації поставленого завдання. Основний акцент зроблено на питаннях архітектури системи, забезпечення безпеки даних та зручності взаємодії користувача з інтерфейсом. Отримані результати демонструють можливість застосування розробленого рішення для оптимізації процесів обліку звернень громадян у підрозділах Національної поліції України.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ СИСТЕМ ОБЛІКУ ЗВЕРНЕНЬ ГРОМАДЯН У ПІДРОЗДІЛАХ НАЦІОНАЛЬНОЇ ПОЛІЦІЇ

Аналіз нормативно-правової бази та порядку розгляду звернень громадян.

Ефективна взаємодія між суспільством та правоохоронними органами є фундаментом довіри до Національної поліції. Правову основу цієї взаємодії складає низка нормативних актів, які регламентують права громадян на звернення та обов'язки поліції щодо їх розгляду.

Фундаментальну роль у регулюванні цього питання відіграє Конституція України. Зокрема, у ст. 40 закріплено право кожного на подання індивідуальних або колективних письмових запитів до владних інстанцій [1]. Подальший розвиток та уточнення цих положень міститься у Законі України «Про звернення громадян» [2]. У вказаному нормативному акті передбачено можливість подання звернень як в усній чи письмовій формі, так і дистанційно — через Internet та засоби електронної комунікації. Саме впровадження поняття «електронне звернення» формує необхідну правову базу для створення автоматизованих вебсистем облік.

Діяльність поліції у цій сфері регулюється Законом України «Про Національну поліцію», зокрема статтею 89, яка передбачає використання поліцією інформаційних ресурсів для виконання службових завдань [3]. Важливим аспектом є дотримання принципів прозорості та відкритості.

Порядок обробки інформації регламентується наказом МВС «Про затвердження Порядку ведення єдиного обліку в органах (підрозділах) поліції заяв і повідомлень про кримінальні правопорушення та інші події» [5]. Згідно з цим порядком, кожне звернення має бути зареєстроване, йому присвоюється унікальний номер, і заявник має бути повідомлений про результати розгляду. Також, оскільки система передбачає роботу з персональними даними (ПІБ, адреса, телефон), критично важливим є дотримання вимог Закону України «Про захист персональних даних» [4]. Це висуває суворі вимоги до технічної

реалізації системи, зокрема до необхідності шифрування паролів та захисту каналів передачі даних, що враховано у даному проєкті за допомогою стандартів безпеки та протоколів авторизації [10].

Аналітичний висновок: На сьогоднішній день інформаційна діяльність поліції трансформується від закритої бюрократичної моделі до концепції «поліцейського сервісу». Проте існуючі правові норми, декларуючи можливість подання електронних звернень, недостатньо регулюють технічні регламенти їх автоматизованої обробки на рівні районних відділів. Часто «електронне звернення» де-факто залишається звичайним електронним листом, який обробляється вручну, що нівелює переваги цифровізації.

Застосування автоматизованих Web-систем дозволяє мінімізувати корупційні ризики шляхом фіксації всіх дій у системі (audit logs) та виключити суб'єктивізм при реєстрації заяв. Однак, незважаючи на наявність потужних централізованих реєстрів (на кшталт ІПНП), існує гостра потреба у гнучких клієнт-орієнтованих рішеннях, які забезпечують «міст» між громадянином та поліцією. Глобальні системи є закритими для зовнішнього користувача, що унеможливлює контроль за станом розгляду заяви з боку заявника. Це обґрунтовує необхідність розробки Web-орієнтованих систем обліку та контролю звернень, які забезпечують прозорість процесів та розглядаються у даній роботі.

Аналіз існуючих інформаційних систем та методів обліку звернень

На сучасному етапі розвитку МВС України використовує розгалужену мережу інформаційних систем. Більшість із них орієнтовані на внутрішні процесуальні потреби поліції та закриті від зовнішнього світу. Розглянемо ключові інструменти крізь призму їхньої доступності для громадян та ефективності зворотного зв'язку.

1. Інформаційний портал Національної поліції (ІПНП). ІПНП — це інтегрована цифрова екосистема, що забезпечує автоматизацію службових процесів та консолідацію інформаційних ресурсів Нацполіції. Портал є

ключовим інструментом для збору, обробки та оперативного доступу до даних про правопорушення, підозрілих осіб, зброю та інші об'єкти обліку в режимі реального часу [3, 5].

- Переваги: Об’єднання різнорідних баз даних (у тому числі спадщини системи «ІПНП») у єдиному зручному інтерфейсі.
- Недоліки з точки зору громадянина: Обмежений доступ для громадян:

Як

2. Традиційний паперовий облік та локальні журнали. У багатьох районних відділах реєстрація звернень, що не містять ознак кримінального правопорушення, досі ведеться у паперових журналах або локальних Excel-файлах відповідно до загальних правил діловодства [5].

- Функціонал: Ручний запис даних черговим офіцером у журнал єдиного обліку.
- Критика: Зазначений підхід є морально застарілим і характеризується низькою ефективністю. Він вимагає особистого звернення до відділення поліції, а робота з паперовими архівами суттєво уповільнює доступ до інформації. У таких умовах підвищується ймовірність помилок при обробці даних або їх втрати. Також відсутні засоби автоматизованої аналітики та оперативного інформування заявника.

3. Система управління нарядами поліції «Цунамі» (IPNP Tsunami). Це сучасна телекомунікаційна система, інтегрована зі службою «102», яка забезпечує автоматизацію процесу прийняття екстрених викликів та управління нарядами патрульної поліції.

- Функціонал: Оператор лінії 102 реєструє виклик, і система автоматично визначає геолокацію заявника, відображає на карті найближчі вільні екіпажі та передає завдання безпосередньо на службовий планшет патрульного.
- Обмеження: Система «Цунамі» спроєктована виключно для оперативного реагування на події, що відбуваються в реальному часі. Вона не

призначена для довготривалого процесуального документообігу, розгляду скарг на дії працівників поліції, інформаційних запитів або заяв, що не потребують негайного виїзду наряду.

Таблиця 1.1.

Порівняльний аналіз доступності та функціональних можливостей систем обліку звернень.

Критерії порівняння	ІС «ІПНП»	Паперовий облік / Excel	ІС «Цунамі»	Розроблювана Web-система
Основне призначення	Внутрішня база даних (розшук, зброя, адмін. порушення)	Реєстрація некримінальних звернень у відділку	Оперативне реагування на екстрені виклики	Онлайн-сервіс для подачі та контролю звернень
Доступність для громадян	Повністю закрита (тільки для поліції)	Обмежена (вимагає фізичної присутності)	Телефонний канал (тільки голосовий зв'язок)	Повна (Web-інтерфейс, доступ 24/7)
Спосіб подачі звернення	Рапорт співробітника	Рукописна заява у відділку	Дзвінок оператору	Електронна форма через особистий кабінет
Відстеження статусу	Неможливо для заявника	Тільки через особистий запит/дзвінок	Не передбачено (система для нарядів)	Автоматичне
Тип комунікації	Одностороння (всередині поліції)	Повільна (офіційне листування)	Екстрена (в момент події)	Двостороння (коментарі, сповіщення)
Недоліки (згідно аналізу)	Непрозорість для громадськості	Ризик втрати даних, бюрократія	Не підходить для довготривалих скарг	Потребує доступу до

Джерело: табл. 1.1 складена автором на основі [2–5].

Результати проведеного аналізу показують, що в цифровій екосистемі МВС існує суттєвий розрив у функціональних можливостях. З одного боку,

ефективно працюють внутрішні інформаційні системи, зокрема відомчі бази даних («ІПНП», ЄРДР) та системи екстреного реагування («Цунамі»), які забезпечують службові потреби поліції. Водночас сфера роботи із зверненнями громадян, що належать до адміністративних послуг, залишається недостатньо автоматизованою та значною мірою базується на паперовому документообігу.

У зв'язку з цим виникає необхідність створення відкритого Web-ресурсу, який забезпечить доступ громадян до сервісу у режимі 24/7 без потреби особистого відвідування підрозділів поліції. Така система повинна надавати можливість подання звернень та відстеження їхнього статусу в режимі реального часу. Саме на вирішення цієї задачі спрямовано розроблення Web-орієнтованої інформаційної системи «Громадянин – Поліція», що реалізується в межах даної кваліфікаційної роботи.

Визначення проблем, вимог і цілей розроблення Web-орієнтованої системи обліку.

Проведене дослідження нормативно-правової бази та існуючих підходів до роботи з громадянами показало наявність суттєвих недоліків, які ускладнюють ефективну комунікацію між населенням та Національною поліцією, а також сповільнюють процес надання адміністративних послуг.

До основних проблем належать:

1. Низька доступність та фізичні бар'єри: Існуюча процедура подання звернень здебільшого вимагає фізичної присутності заявника у відділку поліції або відправки паперового листа. Це створює незручності для громадян, особливо для осіб з обмеженою мобільністю або тих, хто перебуває за кордоном, а також призводить до утворення черг у чергових частинах.

2. Відсутність прозорості та зворотного зв'язку. Після подання паперової заяви громадянин фактично втрачає контроль над ситуацією. Він не має інструментів для відстеження статусу розгляду свого звернення в режимі

реального часу (хто виконавець, чи прийнято рішення) і змушений чекати на офіційну письмову відповідь, яка може йти поштою кілька тижнів.

3. Низька швидкість обробки інформації: Співробітники поліції витрачають значний час на ручну реєстрацію вхідної кореспонденції, перенесення даних з паперових носіїв у локальні журнали обліку та формування звітів. Це призводить до дублювання роботи та збільшує ризик втрати документів або помилок при їх обробці.

Метою розроблення даної системи є створення захищеного Web-орієнтованого додатка, який автоматизує повний цикл обробки звернень громадян: від дистанційної подачі та реєстрації до інформування заявника про результати розгляду.

Для досягнення мети було сформовано наступні функціональні вимоги до системи:

- Розмежування прав доступу: Реалізування двох окремих інтерфейсів - публічного (для громадян) та адміністративного (для співробітників поліції) з різними рівнями доступу до даних.
- Дистанційне подання та валідація: Створення інтуїтивно зрозумілих Web-форм для реєстрації звернень із автоматичною перевіркою коректності введених даних (формат телефону, e-mail, обов’язкові поля) на стороні клієнта.
- Моніторинг статусів (Real-time): Забезпечення можливості для заявника переглядати поточний етап розгляду звернення («Нове», «В роботі», «Вирішено») через особистий кабінет.
- Автоматизація документообігу: Можливість для співробітника поліції змінювати статуси, додавати офіційні коментарі та формувати відповіді в електронному вигляді без необхідності використання паперових журналів.

Технічні вимоги до системи включають використання клієнт-серверної архітектури (Client-Server), адаптивність інтерфейсу для коректного відображення на мобільних пристроях (Responsive Design) та забезпечення

надійного захисту персональних даних користувачів (шифрування паролів, використання захищених токенів авторизації).

Використання Web-інтерфейсу обґрунтовано потребою у забезпеченні широкої доступності сервісу для населення 24/7 без необхідності встановлення додаткового програмного забезпечення на пристрої користувачів. Такий підхід відповідає сучасній концепції «держава у смартфоні» та стратегії цифрової трансформації діяльності органів МВС.

Отже, впровадження запропонованої Web-системи сприятиме цифровому оновленню процесів роботи поліції. Це дозволить автоматизувати обробку звернень громадян, а також підвищити прозорість і відкритість взаємодії з населенням.

Висновок до розділу 1

У першому розділі розглянуто сучасний стан систем обліку звернень громадян у підрозділах Національної поліції. Аналіз нормативно-правової бази свідчить, що законодавство України передбачає можливість подання звернень в електронному вигляді, проте на практиці рівень автоматизації їх обробки залишається недостатнім.

Аналіз існуючих інформаційних систем (ІПНП, «Цунамі», паперовий облік) дозволив виявити ключовий функціональний розрив між внутрішніми системами поліції та доступністю сервісів для громадян. Встановлено, що наявні рішення не забезпечують належного рівня прозорості, оперативності та зворотного зв’язку.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ЗВЕРНЕНЬ

2.1. Визначення функціональних вимог до клієнтської та серверної частин

Процес проєктування інформаційної системи починається з деталізації вимог до її функціональності. Враховуючи специфіку роботи підрозділів поліції, система розробляється як Web-додаток з розділенням прав доступу на основі ролей (Role-Based Access Control — RBAC). Виділено дві основні ролі користувачів: «Громадянин» (зовнішній користувач) та «Офіцер поліції» (внутрішній користувач/адміністратор).

Для клієнтської частини (Frontend), з якою взаємодіє громадянин, сформовано наступні вимоги:

1. Самореєстрація та авторизація: Забезпечення можливості створення облікового запису з прив’язкою до номера телефону або електронної пошти для подальшої ідентифікації.

2. Конструктор звернень: Інтерактивна форма подачі заяви, що включає вибір категорії правопорушення, опис події, вказання дати та часу.

3. Візуалізація статусів: Особистий кабінет повинен відображати життєвий цикл звернення («Нове» → «В роботі» → «Вирішено») у режимі реального часу.

4. Адаптивність інтерфейсу: Система повинна коректно відображатися на мобільних пристроях, оскільки більшість звернень подається зі смартфонів.

Вимоги до адміністративної панелі, з якою працює співробітник поліції:

1. Модерація контенту: Можливість перегляду повного списку звернень з фільтрацією за пріоритетністю (Low, Medium, High) та статусом виконання.

2. Управління життєвим циклом: Функціонал для зміни статусу звернення та призначення відповідального виконавця.

3. Зворотний зв'язок: Можливість додавання офіційних коментарів до картки звернення, які стають видимими для заявника.

Нефункціональні вимоги описують якісні характеристики системи:

1. Безпека (Security): Оскільки система оперує персональними даними, обов’язковим є використання криптографічного хешування паролів (алгоритм bcrypt) та захищених токенів авторизації (JWT) для запобігання несанкціонованому доступу до сесій користувачів [10].

2. Адаптивність (Responsiveness): Інтерфейс клієнтської частини має бути реалізований за принципом Mobile-First, забезпечуючи коректне відображення та зручність використання як на стаціонарних моніторах у відділку, так і на смартфонах громадян [8].

3. Продуктивність (Performance): Архітектура системи повинна базуватися на принципі Single Page Application (SPA), що дозволяє завантажувати сторінки миттєво без повного перезавантаження браузера, знижуючи навантаження на мережу та сервер [6].

Окремим етапом проєктування став вибір технологічного стеку для реалізації клієнтської частини. Було проведено порівняльний аналіз трьох найбільш популярних JavaScript-фреймворків: React, Angular та Vue.js.

1. Angular: Потужний інструмент, проте має надто високий поріг входження та надлишкову складність для розробки прототипу (MVVM архітектура).

2. Vue.js: Зручний для простих інтерфейсів, але має меншу екосистему готових компонентів порівняно з конкурентами.

3. React.js: Було обрано як основний інструмент через його компонентний підхід, високу швидкість рендерингу завдяки Virtual DOM та широку підтримку спільноти [7]. Використання бібліотеки React дозволяє створювати масштабований інтерфейс, де елементи (кнопки, форми, картки звернень) є незалежними модулями, що значно спрощує подальшу підтримку коду.

Для серверного середовища було обрано платформу Node.js. На відміну від традиційних мов (PHP, C#), Node.js використовує неблокуючу модель вводу-виводу (Non-blocking I/O), що є критично важливим для систем з великою кількістю одночасних запитів [9].

Серверна частина (Backend) повинна забезпечувати обробку HTTP-запитів, валідацію вхідних даних на стороні сервера та взаємодію з базою даних. Ключовою вимогою є забезпечення безстанової (stateless) архітектури за допомогою REST API [9].

Для візуалізації функціональних можливостей системи та взаємодії користувачів із програмним забезпеченням було розроблено діаграму варіантів використання (UML Use Case Diagram). Вона відображає основні прецеденти, доступні для кожної з ролей у системі, та межі самої системи. Графічне представлення моделі наведено на рисунку 2.1.

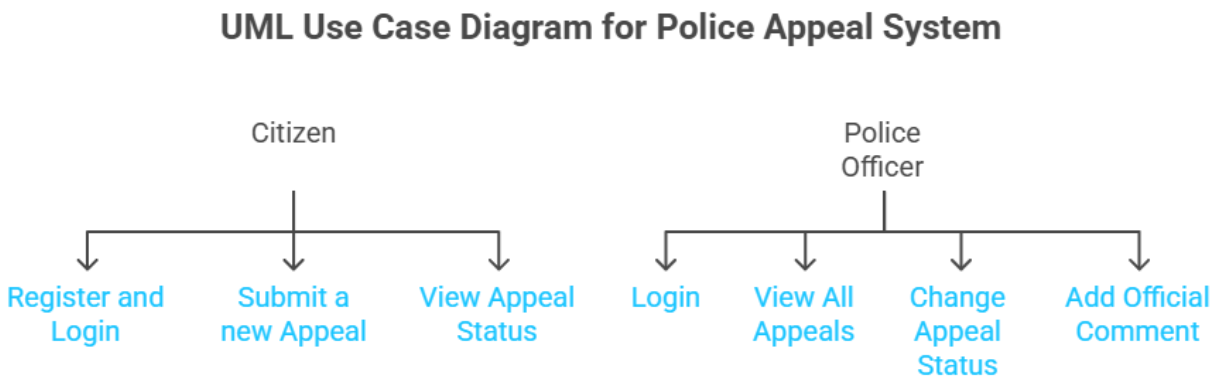


Рис. 2.1. Діаграма варіантів використання (Use Case) системи обліку звернень.

Як видно з наведеної діаграми, актор «Громадянин» ініціює процеси створення даних та їх перегляду. У той же час актор «Офіцер поліції» виконує адміністративні функції, маючи доступ до інструментів модерації та зміни станів бізнес-об'єктів. Спільним варіантом використання для обох ролей є процес автентифікації («Вхід у систему»), який є обов'язковою передумовою для доступу до захищених ресурсів. Таке розділення функціоналу на рівні проєктування дозволяє чітко імплементувати механізм розмежування прав доступу (RBAC) на етапі програмної реалізації.

Проєктування бази даних та логічних зв’язків

На відміну від файлових систем зберігання даних (наприклад, JSON-файлів), для забезпечення цілісності та структурованості інформації обрано реляційну модель бази даних. Як СКБД для прототипу використано SQLite, що дозволяє зберігати всю базу в одному файлі, забезпечуючи мобільність розгортання [11].

Інфологічна модель системи складається з наступних сутностей:

1. Сутність Users (Громадяни): Зберігає персональні дані заявників.

- Ключові атрибути: id (PK), full_name, phone, email (Unique), password_hash.

2. Сутність Officers (Співробітники): Зберігає дані про персонал відділу.

- Ключові атрибути: id (PK), badge_number (Unique), rank, department_id.

3. Сутність Appeals (Звернення): Центральна таблиця системи.

- Ключові атрибути: id, title, description, status, priority.
- Зв’язки: Має зовнішні ключі citizen_id (зв'язок Many-to-One з Users) та officer_id (зв'язок Many-to-One з Officers).

4. Сутність Categories: Довідник типів звернень (наприклад, «Крадіжка», «ДТП», «Хуліганство»).

Для забезпечення цілісності даних та уникнення дублювання інформації було розроблено структуру даних, яка включає чотири основні сутності. Центральним елементом системи є сутність «Звернення», яка акумулює інформацію про подію та містить посилання на довідники та облікові записи користувачів.

Інфологічна модель системи була нормалізована до Третьої нормальної форми (3NF) для уникнення надлишковості даних [11]. Детальна структура основної сутності системи Appeals (Звернення) наведена в таблиці 2.1.

Таблиця 2.1.

Детальна структура основної сутності системи Appeals (Звернення)

Назва атрибута	Тип даних (SQLite)	Обмеження (Constraints)	Опис призначення
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор звернення в системі
citizen_id	INTEGER	FOREIGN KEY, NOT NULL	Зовнішній ключ для зв'язку із таблицею Users
title	VARCHAR(150)	NOT NULL	Короткий заголовок суті звернення
description	TEXT	NOT NULL	Повний опис обставин події
status	VARCHAR(20)	DEFAULT 'New'	Поточний стан ('New', 'In Progress', 'Done')
priority	VARCHAR(20)	DEFAULT 'Medium'	Рівень важливості для сортування черги
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Автоматична мітка часу створення заяви

Джерело: табл. 2.1 складено автором за результатами роботи

Також спроєктовано супутню таблицю Users, яка містить хешовані паролі (поле password_hash типу VARCHAR(255)), що відповідає вимогам безпеки. Використання зовнішніх ключів (FOREIGN KEY) забезпечує посилальну цілісність: неможливо створити звернення від неіснуючого користувача або видалити користувача, поки в базі існують його активні звернення [11].

Для забезпечення цілісності даних та уникнення дублювання інформації було розроблено структуру даних, яка включає чотири основні сутності. Центральним елементом системи є сутність «Звернення», яка акумулює інформацію про подію та містить посилання на довідники та облікові записи користувачів.

Інфологічна модель структури бази даних, що відображає ієрархію сутностей та їх атрибутний склад, наведена на рисунку 2.2.

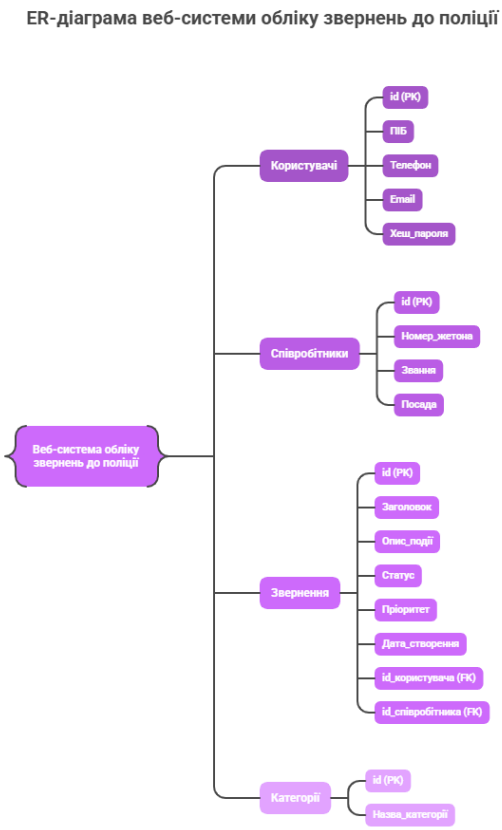


Рис. 2.2. Інфологічна модель структури бази даних системи.

Як видно зі схеми, спроєктована база даних складається з наступних нормалізованих таблиць:

1. Користувачі (Users): Зберігає облікові дані громадян, включаючи хеш пароля для безпечної автентифікації.

2. Співробітники (Officers): Містить інформацію про персонал поліції, включаючи номер жетона та звання.

3. Звернення (Appeals): Головна таблиця, що містить змістовну частину заяви (заголовок, опис) та службові поля (статус, пріоритет, дата).

4. Категорії (Categories): Довідник для класифікації правопорушень.

Зв'язок між таблицями реалізується через механізм зовнішніх ключів (FK), де поля id_користувача та id_співробітника в таблиці звернень посилаються на відповідні первинні ключі (PK) батьківських таблиць.

Архітектура клієнт-серверної взаємодії

Архітектура розроблюваної системи базується на патерні MVC (Model-View-Controller) у поєднанні з принципами RESTful API [9]. Взаємодія компонентів відбувається наступним чином:

- Клієнтський рівень (Frontend / View): Реалізований на бібліотеці React. Відповідає за відображення інтерфейсу та відправку асинхронних запитів (через бібліотеку Axios) до сервера. Сторінка не перезавантажується повністю при переході між розділами (Single Page Application — SPA), що забезпечує високу швидкодію [6].
- Маршрутизація та Контролери (Controller): Сервер на Node.js/Express приймає запит на певний «ендпоінт» (наприклад, POST /api/appeals). Маршрутизатор перенаправляє запит до відповідного контролера, який містить бізнес-логіку [9].
- Модель даних (Model): Контролер звертається до моделі бази даних для виконання операцій Create, Read, Update, Delete (CRUD). Отримані дані конвертуються у формат JSON і повертаються клієнту [11].

Взаємодія між клієнтом та сервером відбувається через чітко регламентований інтерфейс прикладного програмування (API). Використання стандартних методів HTTP (GET, POST, PATCH) забезпечує семантичну правильність архітектури та спрощує інтеграцію з іншими системами МВС у майбутньому [9].

Модель життєвого циклу звернення

Функціонування системи обліку звернень базується на чітко визначеній моделі життєвого циклу заявки. Кожне звернення проходить послідовні етапи обробки, що дозволяє контролювати його стан.

Таблиця 2.2.

Структура таблиці «Appeals» у базі даних

Назва атрибута	Тип даних (SQLite)	Обмеження (Constraints)	Опис призначення
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор звернення в системі
citizen_id	INTEGER	FOREIGN KEY, NOT NULL	Зовнішній ключ для зв'язку із таблицею Users
title	VARCHAR(150)	NOT NULL	Короткий заголовок суті звернення
description	TEXT	NOT NULL	Повний опис обставин події
status	VARCHAR(20)	DEFAULT 'New'	Поточний стан ('New', 'In Progress', 'Done')
priority	VARCHAR(20)	DEFAULT 'Medium'	Рівень важливості для сортування черги
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Автоматична мітка часу створення заяви

Джерело: табл. 2.2 складено автором за результатами роботи

Послідовність взаємодії клієнта та сервера:

1. Клієнт формує HTTP-запит.
2. Запит передається на сервер через API.
3. Сервер виконує перевірку токена.
4. Контролер обробляє бізнес-логіку.
5. Дані записуються або зчитуються з бази даних.
6. Сервер повертає відповідь у форматі JSON.
7. Клієнт оновлює інтерфейс без перезавантаження сторінки.

Переваги використання SPA-архітектури

Використання Single Page Application дозволяє мінімізувати кількість запитів до сервера та забезпечує швидке оновлення інтерфейсу. Дані підвантажуються асинхронно, що значно покращує продуктивність та користувацький досвід.

Використання стандартних методів HTTP (GET для отримання, POST для створення, PATCH для часткового оновлення) забезпечує семантичну правильність архітектури та спрощує інтеграцію з іншими системами МВС у майбутньому.

Заходи з безпеки та захисту персональних даних

Оскільки система призначена для використання у правоохоронних органах, питання безпеки є пріоритетним. На етапі проєктування було закладено наступні механізми захисту:

1. Автентифікація на основі токенів (JWT): Замість класичних сесій використовується стандарт JSON Web Token. Після успішного входу користувач отримує зашифрований токен, який передається у заголовку кожного наступного запиту (Authorization: Bearer) [10].

2. Хешування паролів: Паролі користувачів ніколи не зберігаються у відкритому вигляді. Використовується алгоритм хешування bcrypt, який додає до пароля випадковий рядок («сіль») перед шифруванням. Це робить

неможливим відновлення оригінального пароля навіть у випадку витоку бази даних [10].

3. Валідація вхідних даних: Для захисту від SQL-ін’єкцій та XSS-атак усі дані, що надходять від користувача, проходять сувору типізацію та очистку на стороні сервера перед виконанням запиту до БД [11].

Додатково впроваджено механізм CORS (Cross-Origin Resource Sharing). Оскільки клієнтська частина (React) та серверна частина (Node.js) можуть працювати на різних портах (наприклад, 3000 та 5000), сервер налаштований таким чином, щоб приймати запити виключно від довіреного домену. Це у н е м о ж л и в л ю є в и к о н а н н я ш

Висновок до розділу 2

У другому розділі було виконано проєктування інформаційної системи обліку звернень громадян. На основі визначених вимог розроблено функціональну модель системи із розмежуванням прав доступу для різних ролей користувачів (громадянин, офіцер).

У межах роботи спроєктовано структуру реляційної бази даних, нормалізовану до третьої нормальної форми, що сприяє впорядкованому та надійному зберіганню інформації. Визначено логічні зв’язки між сутностями та впроваджено використання зовнішніх ключів для підтримки коректних взаємозв’язків між даними.

Також визначено архітектуру клієнт-серверної взаємодії на основі REST API та SPA-підходу, що забезпечує швидкодію та масштабованість системи. Окрему увагу приділено питанням безпеки, зокрема автентифікації користувачів, захисту персональних даних та запобіганню основним типам Web-атак.

РОЗДІЛ 3. РЕАЛІЗУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ОБЛІКУ ЗВЕРНЕНЬ

Вибір та обґрунтування технологічного стеку

Для програмної реалізації спроєктованої у другому розділі архітектури було обрано сучасний стек технологій PERN (Postgres/SQLite, Express, React, Node.js), адаптований під вимоги швидкого прототипування.

Основою серверної частини обрано платформу Node.js. Вибір обумовлений наявністю величезного репозиторію пакетів npm, що дозволяє не писати стандартний функціонал з нуля, а використовувати перевірені рішення. Ключові бібліотеки, використані у проєкті:

- Express.js: Мікрофреймворк для обробки HTTP-запитів та маршрутизації. Він забезпечує швидке налаштування middleware для обробки JSON-даних.
- Sequelize (або sqlite3): ORM-бібліотека для безпечної взаємодії з базою даних, що захищає від SQL-ін’єкцій.
- Bcrypt.js: Бібліотека для хешування паролів.
- Jsonwebtoken (JWT): Для генерації токенів доступу.

Для клієнтської частини використано бібліотеку React.js. Збірка проєкту здійснюється за допомогою інструменту Vite, який забезпечує значно вищу швидкість компіляції порівняно з класичним Create-React-App. Для організації навігації без перезавантаження сторінки використано бібліотеку react-router-dom

Реалізування серверної частини

Розроблення серверної частини розпочалася з налаштування точки входу — файлу server.js, який ініціалізує підключення до бази даних SQLite та запускає прослуховування порту 5000.

Структура серверного проєкту реалізована за модульним принципом:

- /controllers - містить логіку обробки запитів;
- /routes - визначає маршрути API;
- /models - описує схеми таблиць бази даних.

Ключовим елементом безпеки системи є контролер авторизації (authController.js). Нижче наведено фрагмент коду, що відповідає за реєстрацію користувача з попереднім хешуванням пароля.

```
const register = async (req, res) => {
  try {
    const { full_name, position, badge_number, email, password, phone, role = 'officer' } = req.body;

    if (!full_name || !position || !badge_number || !email || !password) {
      return res.status(400).json({
        success: false,
        error: 'Всі обов\'язкові поля повинні бути заповнені'
      });
    }

    const passwordHash = await bcrypt.hash(password, 10);

    const result = await db.run(
      `INSERT INTO officers (full_name, position, badge_number, email, password, phone, role)
       VALUES (?, ?, ?, ?, ?, ?, ?)`,
      [full_name, position, badge_number, email, passwordHash, phone || null, role]
    );

    const newOfficer = await db.get(
      'SELECT id, full_name, position, badge_number, email, phone, role, created_at FROM officers WHERE id = ?',
      [result.id]
    );

    res.status(201).json({
      success: true,
      data: newOfficer
    });
  } catch (error) {
    if (error.message.includes('UNIQUE constraint failed')) {
      return res.status(409).json({
        success: false,
        error: 'Працівник з таким email або номером жетона вже існує'
      });
    }
    console.error('❌ Помилка реєстрації:', error);
    res.status(500).json({
      success: false,
      error: 'Помилка сервера при реєстрації'
    });
  }
};
```

Рис. 3.1. Реалізування методу реєстрації користувача з хешуванням пароля.

Для обробки звернень було реалізовано REST-контролер appealController.js. Він містить метод create, який приймає дані з фронтенду, валідує їх на наявність обов'язкових полів (заголовок, опис) та зберігає новий запис у таблицю Appeals. Важливою деталлю реалізації є автоматичне присвоєння статусу 'New' для кожного нового звернення.

Реалізацію серверної частини для роботи з категоріями, що включає обробку запитів отримання, створення, оновлення та видалення даних, наведено у додатку В.

Окрім логіки авторизації, критично важливим етапом була реалізування модуля підключення до бази даних. У файлі db.js створено екземпляр об'єкта Database бібліотеки sqlite3. Для забезпечення надійності роботи реалізовано обробку подій успішного з’єднання та виникнення помилок при відкритті файлу бази даних.

```
class Database {
  constructor() {
    const dbPath = process.env.DB_PATH || './database/appeals.db';
    const dbDir = path.dirname(dbPath);

    if (!fs.existsSync(dbDir)) {
      fs.mkdirSync(dbDir, { recursive: true });
    }

    this.db = new sqlite3.Database(dbPath, (err) => {
      if (err) {
        console.error('❌ Помилка підключення до БД:', err.message);
      } else {
        console.log('✓ Підключено до SQLite бази даних');
        this.initializeTables();
      }
    });
  }
```

Рис. 3.2. Ініціалізація підключення до бази даних SQLite.

Основним функціональним елементом системи є контролер appealController.js. Метод createAppeal відповідає за збереження даних, отриманих від громадянина. Особливістю реалізації є використання параметризованих SQL-запитів, що повністю унеможливлює атаки типу SQL Injection. Нижче наведено програмний код цього методу.

Алгоритм роботи методу поділяється на два етапи. Перший етап — це деструктуризація вхідних даних та сувора перевірка посилальної цілісності. Перед тим як створити запис, сервер перевіряє, чи існують у базі даних вказаний громадянин (citizen_id), категорія події (category_id) та, за наявності, відповідальний офіцер.

```
const { citizen_id, category_id, assigned_officer_id, title, description, status = 'new', priority = 'medium' } = req.body;

if (!citizen_id || !category_id || !title) {
  return res.status(400).json({
    success: false,
    error: 'Громадянин, категорія та заголовок обов\'язкові'
  });
}

const citizen = await db.get('SELECT id FROM citizens WHERE id = ?', [citizen_id]);
if (!citizen) {
  return res.status(404).json({
    success: false,
    error: 'Громадянин не знайдено'
  });
}

const category = await db.get('SELECT id FROM categories WHERE id = ?', [category_id]);
if (!category) {
  return res.status(404).json({
    success: false,
    error: 'Категорія не знайдена'
  });
}
```

Рис. 3.3. Блок валідації даних та перевірки існування сутностей.

Другий етап — це безпосереднє виконання транзакції збереження. Використовується параметризований SQL-запит INSERT, що захищає систему від SQL-ін'єкцій. Якщо запис пройшов успішно, система виконує повторний запит SELECT, щоб повернути на клієнтську частину повний об'єкт створеного звернення разом із його згенерованим id.

```
const result = await db.run(
  `INSERT INTO appeals (citizen_id, category_id, assigned_officer_id, title, description, status, priority)
   VALUES (?, ?, ?, ?, ?, ?, ?)`,
  [citizen_id, category_id, assigned_officer_id || null, title, description || null, status, priority]
);

const newAppeal = await db.get('SELECT * FROM appeals WHERE id = ?', [result.id]);

res.status(201).json({
  success: true,
  data: newAppeal
});
catch (error) {
console.error('❌ Помилка створення звернення:', error);
res.status(500).json({
  success: false,
  error: 'Помилка сервера при створенні звернення'
});
```

Рис. 3.4. Виконання запиту на збереження звернення та обробка помилок.

Такий підхід до розділення логіки дозволяє уникнути збереження "битих" посилань у базі даних та забезпечує інформативні повідомлення про помилки для користувача.

Для наочної демонстрації логіки роботи контролера та послідовності перевірок, які проходить запит від клієнта перед збереженням у базу даних, було розроблено блок-схему алгоритму обробки нового звернення. Графічне представлення алгоритму наведено на рисунку 3.5.

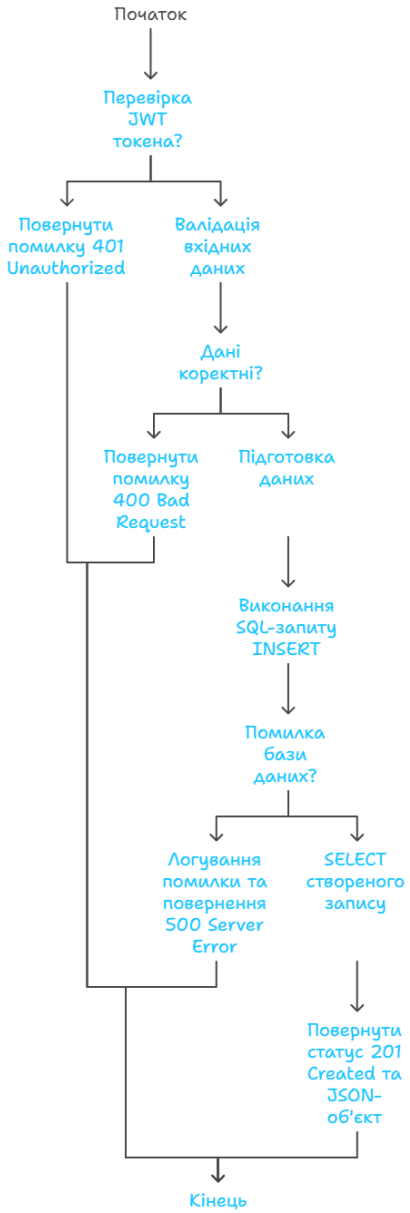


Рис. 3.5. Блок-схема алгоритму обробки запиту на створення звернення на сервері.

Як видно з блок-схеми, алгоритм реалізує патерн «fail-fast» (швидка відмова). Система послідовно перевіряє валідність токена авторизації та коректність вхідних даних. У разі виявлення будь-якої невідповідності обробка запиту негайно припиняється з поверненням відповідного HTTP-коду помилки клієнту. Ресурсномістка операція запису в базу даних виконується лише після успішного проходження всіх попередніх перевірок, що забезпечує стабільність та продуктивність серверної частини системи.

Окремим архітектурним компонентом сервера є проміжне ПЗ (Middleware). Його задача — перехоплювати HTTP-запит до моменту його передачі контролеру та перевіряти наявність прав доступу.

Мною було розроблено модуль authMiddleware.js, який виконує розпаршування заголовка Authorization. Якщо у заголовку відсутній валідний Bearer-токен, сервер автоматично перериває ланцюжок обробки та повертає статус 403 Forbidden. Це гарантує, що анонімні користувачі не зможуть звернутися до захищених маршрутів API.

Для розмежування прав доступу та захисту приватних маршрутів (таких як створення звернення) було розроблено спеціалізоване проміжне ПЗ (Middleware).

Модуль citizenAuth перехоплює HTTP-запити до сервера перед їх обробкою контролерами. Алгоритм його роботи наступний:

1. Вилучення токена: Система зчитує заголовок Authorization та виокремлює з нього Bearer-токен.

2. Верифікація: За допомогою методу jwt.verify перевіряється цифровій підпис токена. Якщо токен підроблений або його термін дії вичерпався, сервер повертає помилку 401.

3. Перевірка ролі: Додатково здійснюється перевірка типу користувача (decoded.type). Це гарантує, що офіцер поліції не зможе виконувати дії, призначені виключно для громадян, і навпаки.

Реалізацію центрального компонента клієнтської частини системи, який відповідає за маршрутизацію, управління станом авторизації та структуру додатку, наведено у додатку А

Реалізування механізму захисту наведена на рисунку 3.6.

```
const jwt = require('jsonwebtoken');

const citizenAuth = (req, res, next) => {
  try {
    const token = req.headers.authorization?.split(' ')[1];

    if (!token) {
      return res.status(401).json({
        success: false,
        error: 'Токен відсутній'
      });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET || 'your_super_secret_key_change_in_production');

    if (decoded.type !== 'citizen') {
      return res.status(403).json({
        success: false,
        error: 'Доступ заборонено'
      });
    }

    req.citizenId = decoded.id;
    next();
  } catch (error) {
    res.status(401).json({
      success: false,
      error: 'Недійсний або прострочений токен'
    });
  }
};

module.exports = citizenAuth;
```

Рис. 3.6. Реалізування Middleware для авторизації громадян.

Після успішної ідентифікації користувача сервер генерує цифровий ключ доступу. Для перевірки коректності формування вмісту токена було проведено його декодування за допомогою інструменту налагодження. Результат декодування наведено на рисунку 3.7.

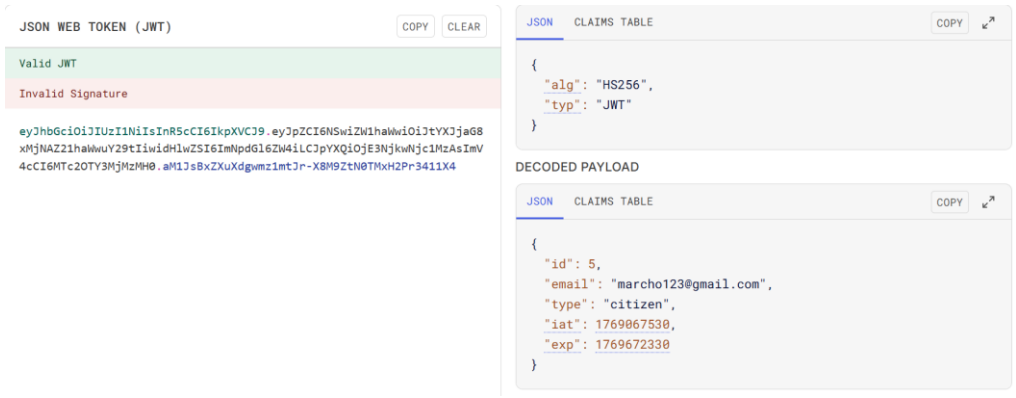


Рис. 3.7. Декодована структура JWT-токена авторизованого громадянина.

Як видно з наведеного рисунка, корисне навантаження токена (Payload) містить критично важливі дані для роботи системи:

- type: "citizen" - це поле (claim) використовується на сервері для розмежування прав доступу. Middleware перевіряє це значення і, якщо воно відповідає потрібній ролі, пропускає запит далі.
- exp (Expiration) - мітка часу закінчення дії токена. Це забезпечує безпеку: навіть якщо токен буде викрадено, зловмисник зможе використовувати його лише обмежений час.
- id та email - ідентифікатори, що дозволяють системі розуміти, хто саме виконує запит, без необхідності повторного звернення до таблиці користувачів у базі даних.

Переваги використання JWT у системі:

- Stateless архітектура (сервер не зберігає сесії)
- Висока масштабованість
- Швидка перевірка доступу
- Зменшення навантаження на базу даних
- Універсальність (можна використовувати в різних клієнтах)

Недоліки та обмеження JWT

- Неможливість відкликання токена до завершення терміну дії
- Ризик компрометації при зберіганні в localStorage
- Збільшений розмір запитів
- Потреба у додаткових механізмах безпеки (refresh tokens)

Використання JWT-токенів у системі дозволяє реалізувати ефективний та безпечний механізм автентифікації користувачів. Завдяки цьому забезпечується розмежування доступу, зниження навантаження на сервер та можливість масштабування системи без втрати продуктивності.

Процес обробки HTTP-запиту в системі

Кожен запит, що надходить від клієнта, проходить кілька етапів обробки. Спочатку middleware виконує перевірку автентифікації та прав доступу. Далі запит передається до відповідного контролера, який реалізує бізнес-логіку. Контролер взаємодіє з базою даних через модель та формує відповідь у форматі JSON.Такий підхід дозволяє розділити відповідальність між компонентами системи та забезпечує гнучкість при масштабуванні.

Обробка помилок у серверній частині

У системі реалізовано централізований механізм обробки помилок. Усі виняткові ситуації (некоректні дані, відсутність запису, помилки бази даних) перехоплюються та повертаються клієнту у стандартизованому форматі JSON. Це дозволяє забезпечити стабільність роботи системи та покращує взаємодію між frontend і backend.

Використання алгоритму шифрування HS256 (зазначеного у заголовку токена) гарантує цілісність переданих даних.»

Реалізування інтерфейсу користувача

Клієнтська частина побудована на основі бібліотеки React.js. Архітектура фронтенду базується на компонентному підході, де інтерфейс розбито на незалежні модулі, що значно спрощує підтримку та масштабування.

Структура проєкту включає такі ключові директорії:

- /components - перевикористовувані «дурні» (dumb) компоненти, що відповідають лише за відображення (кнопки, інпути, картки);

- /pages - компоненти-сторінки, які підключаються до Redux/Context та містять бізнес-логіку;
- /services - модулі для виконання асинхронних HTTP-запитів;
- /context - провайдери глобального стану (наприклад, даних авторизованого користувача).

1. Реалізування форми подачі звернення Центральним елементом системи для громадянина є форма створення звернення. Вона реалізована з використанням паттерну «Контрольовані компоненти» (Controlled Components). Це означає, що значення кожного поля форми зберігається у стані компонента (useState), що дозволяє реалізувати миттєву валідацію (наприклад, активувати кнопку відправки тільки коли всі поля заповнені).

Функція onSubmit виконує ключову роль у процесі обробки даних форми подачі звернення та є центральним елементом взаємодії між клієнтською та серверною частинами системи. Вона викликається після успішного проходження валідації полів форми та відповідає за підготовку, трансформацію та передачу даних на сервер.

На першому етапі функція отримує об’єкт із введеними користувачем даними, який формується бібліотекою react-hook-form. Оскільки деякі значення (зокрема ідентифікатор категорії) надходять із HTML-елементів у вигляді рядків, виконується їх приведення до відповідного типу даних. Зокрема, поле category_id перетворюється у числовий формат за допомогою функції Number(), що є необхідним для коректної обробки даних на сервері та відповідності структурі бази даних.

Після підготовки даних функція викликає сервісний метод createAppeal, який інкапсулює логіку взаємодії з REST API. Цей метод формує HTTP-запит типу POST та передає його на сервер разом із заголовком авторизації (Bearer-токен), що забезпечує ідентифікацію користувача. Таким чином, клієнтська частина не містить прямої логіки роботи з API, а делегує її спеціалізованому сервісному шару, що відповідає принципу розділення відповідальності (Separation of Concerns).

Функція onSubmit реалізована як асинхронна (з використанням async/await), що дозволяє обробляти результат виконання запиту без блокування інтерфейсу користувача. У процесі виконання запиту система може відображати стан завантаження (loading), що покращує взаємодію з користувачем та запобігає повторному натисканню кнопки відправки.

Для підвищення зручності використання (User Experience) інтегровано бібліотеку react-hot-toast, яка забезпечує миттєвий зворотний зв’язок із користувачем. У разі успішного створення звернення відображається повідомлення про успіх, після чого інтерфейс може бути оновлений (наприклад, перенаправлення на список звернень або очищення форми). У випадку виникнення помилки (наприклад, проблеми з мережею або сервером) користувач отримує відповідне повідомлення про помилку, що дозволяє оперативно реагувати та повторити дію.

Такий підхід до організації обробки форми забезпечує:

- коректність передачі даних між клієнтом і сервером;
- розділення логіки інтерфейсу та API-запитів;
- підвищення надійності системи;
- покращення користувацького досвіду завдяки інтерактивному зворотному зв’язку.

Нижче на рисунку 3.8 наведено фрагмент коду, що демонструє обробку стану форми та відправку даних на сервер.

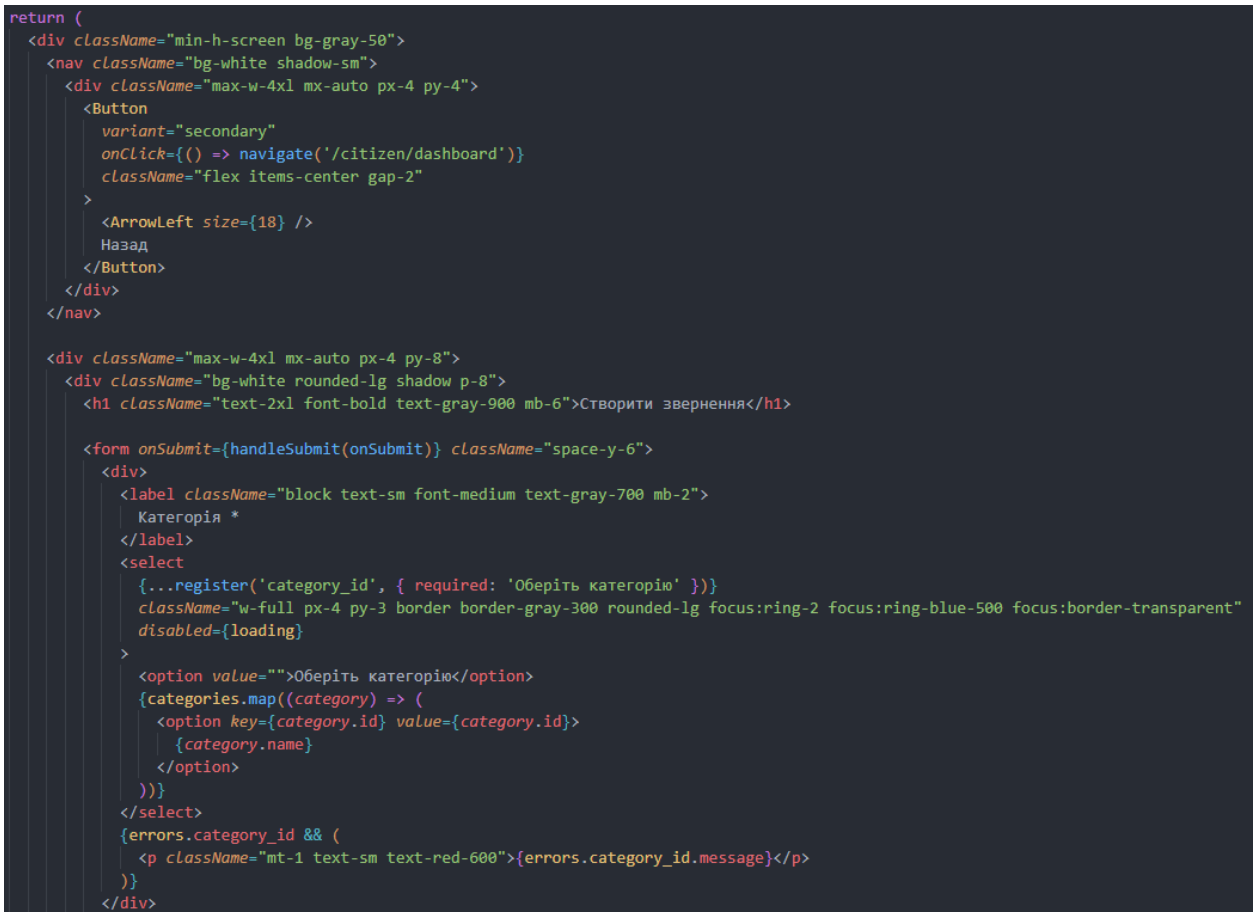
```
export default function CitizenAppealCreatePage() {
  const navigate = useNavigate();
  const [categories, setCategories] = useState([]);
  const [loading, setLoading] = useState(true);
  const { register, handleSubmit, formState: { errors } } = useForm();

  useEffect(() => {
    loadCategories();
  }, []);

  const loadCategories = async () => {
    try {
      const response = await categoryService.getAll();
      if (response.data.success) {
        setCategories(response.data.data);
      }
    } catch (error) {
      toast.error('Помилка завантаження категорій');
    } finally {
      setLoading(false);
    }
  };

  const onSubmit = async (data) => {
    try {
      const response = await citizenAppealService.createAppeal({
        category_id: parseInt(data.category_id),
        title: data.title,
        description: data.description
      });
      if (response.data.success) {
        toast.success('Звернення створено успішно');
        navigate('/citizen/dashboard');
      }
    } catch (error) {
      toast.error(error.response?.data?.error || 'Помилка створення звернення');
    }
  };
```

Рис. 3.8. Використання хуків та сервісних служб для обробки даних звернення.

2. Реалізування інтерфейсу та навігації: Візуальна структура сторінки складається з навігаційної панелі та основної області контенту. Для зручності користувача реалізовано кнопку «Назад», яка повертає до списку звернень за допомогою хука useNavigate.

Перша частина форми відповідає за класифікацію звернення. Для вибору категорії використано стандартний HTML-елемент select, який динамічно наповнюється даними, отриманими з сервера (метод .map). Інтеграція з react-hook-form здійснюється через функцію register, яка зв'язує поле з внутрішнім станом форми.

Фрагмент коду, що відповідає за структуру та вибір категорії, наведено на рисунку 3.6.



```
return (
  <div className="min-h-screen bg-gray-50">
    <nav className="bg-white shadow-sm">
      <div className="max-w-4xl mx-auto px-4 py-4">
        <Button
          variant="secondary"
          onClick={() => navigate('/citizen/dashboard')}
          className="flex items-center gap-2"
        >
          <ArrowLeft size={18} />
          Назад
        </Button>
      </div>
    </nav>

    <div className="max-w-4xl mx-auto px-4 py-8">
      <div className="bg-white rounded-lg shadow p-8">
        <h1 className="text-2xl font-bold text-gray-900 mb-6">Створити звернення</h1>

        <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-2">
              Категорія *
            </label>
            <select
              {...register('category_id', { required: 'Оберіть категорію' })}
              className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-transparent"
              disabled={loading}
            >
              <option value="">Оберіть категорію</option>
              {categories.map((category) => (
                <option key={category.id} value={category.id}>
                  {category.name}
                </option>
              ))}
            </select>
            {errors.category_id && (
              <p className="mt-1 text-sm text-red-600">{errors.category_id.message}</p>
            )}
          </div>
```

Рис. 3.9. Структура сторінки та реалізування динамічного випадаючого списку.

3. Реалізування полів вводу та елементів управління Для текстових даних (заголовок звернення) використано перевикористовуваний компонент <Input />. Це дозволяє інкапсулювати стилі та логіку відображення помилок в одному місці, роблячи код форми чистішим. Поле детального опису реалізовано через тег textarea для можливості введення великого обсягу тексту.

Внизу форми розміщено блок кнопок. Кнопка «Відправити» автоматично блокується, якщо валідація не пройшла успішно, або якщо триває процес відправки даних.

Завершальна частина форми наведена на рисунку 3.10.

```
<Input
  label="Заголовок *"
  name="title"
  register={register('title', { required: 'Заголовок обов\'язковий' })}
  error={errors.title}
  placeholder="Короткий опис проблеми"
/>

<div>
  <label className="block text-sm font-medium text-gray-700 mb-2">
    Детальний опис *
  </label>
  <textarea
    {...register('description', { required: 'Опис обов\'язковий' })}
    rows={6}
    className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-transparent"
    placeholder="Опишіть вашу проблему детально..."
  />
  {errors.description && (
    <p className="mt-1 text-sm text-red-600">{errors.description.message}</p>
  )}
</div>
```

Рис. 3.10. Використання кастомних компонентів вводу та кнопок дії.

3.4. Демонстрація роботи системи та сценарії використання

Перевірка працездатності розробленого програмного забезпечення здійснювалася шляхом тестування основних сценаріїв використання (Use Cases), визначених на етапі проєктування. Тестування проводилося у браузері Google Chrome (версія 120+) з використанням інструментів розробника (DevTools) для моніторингу мережевих запитів.

Сценарій 1: Реєстрація та авторизація користувача Першою точкою входу в систему є сторінка авторизації. Оскільки доступ до функціоналу подачі звернень мають лише ідентифіковані громадяни, неавторизований користувач автоматично перенаправляється на маршрут /login.

Інтерфейс сторінки входу містить форму з полями «Email» та «Пароль». Реалізовано клієнтську валідацію: кнопка входу залишається неактивною, доки поля не будуть заповнені коректно. Після успішної автентифікації сервер повертає JWT-токен, який зберігається у локальному сховищі, а користувач перенаправляється до головної панелі (Dashboard).

Реалізацію сторінки авторизації, зображеної на рисунку 3.11, наведено у додатку Б. У додатку представлено відповідні фрагменти програмного коду,

що демонструють структуру інтерфейсу та логіку взаємодії користувача із системою під час виконання процедури автентифікації.

Зовнішній вигляд сторінки авторизації наведено на рисунку 3.11.

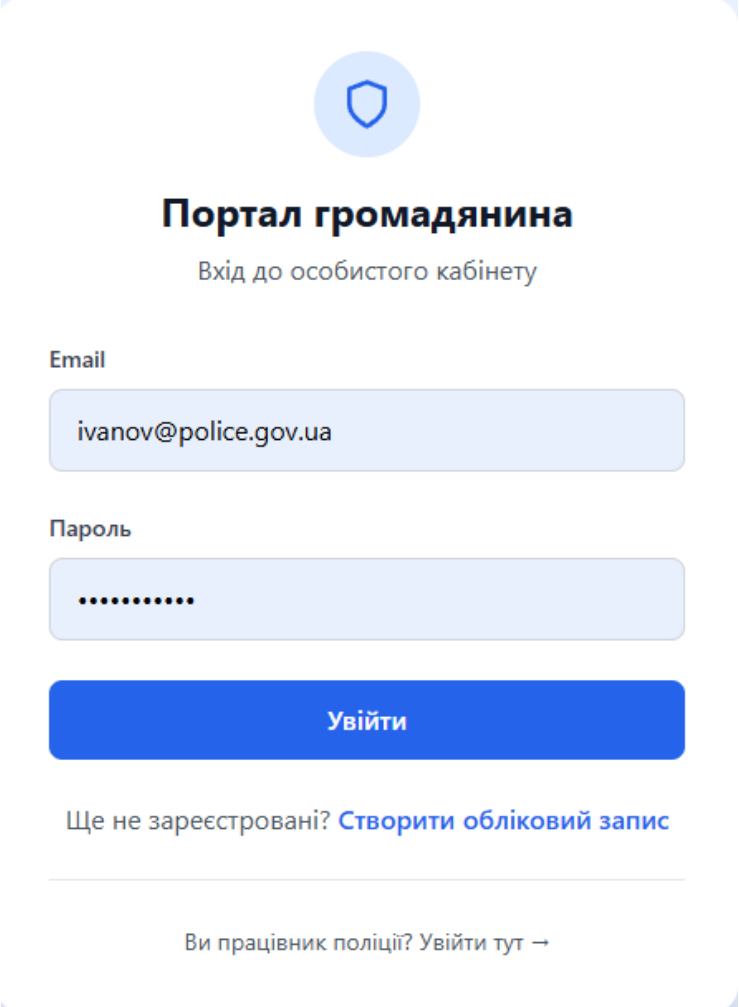


Рис. 3.11 Інтерфейс сторінки авторизації користувача.

Сценарій 2: Робота в кабінеті громадянина Після входу громадянин потрапляє до особистого кабінету. Головна сторінка відображає список раніше поданих звернень із кольоровою індикацією їх поточного статусу («New» — синій, «In Progress» — жовтий, «Done» — зелений). Це дозволяє користувачу миттєво оцінити стан розгляду його заяв.

Процес створення нового звернення відбувається на окремій сторінці (код якої розглянуто у п. 3.3). Після заповнення форми та натискання кнопки «Відправити», система відображає спливаюче повідомлення (Toast) про успішну операцію, а нове звернення з'являється у загальному списку без необхідності перезавантаження сторінки.

Приклад відображення списку звернень в особистому кабінеті зображено на рисунку 3.12.

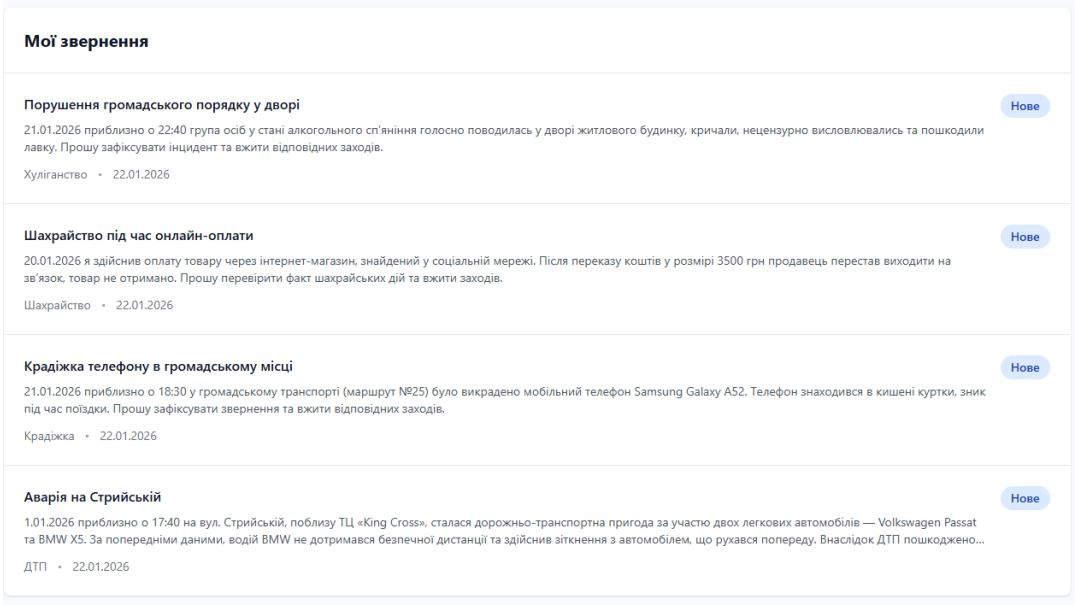


Рис. 3.12 Панель керування громадянина з історією поданих звернень.

Сценарій 3: Адміністрування системи Сценарій роботи офіцера поліції відрізняється розширеними правами доступу. При вході під обліковим записом із роллю officer, система завантажує адміністративну панель. Ключовою відмінністю є наявність інструментів для зміни статусу звернення.

Офіцер має можливість відкрити детальну картку звернення, ознайомитися з описом події та змінити статус (наприклад, прийняти в роботу або закрити справу). Будь-які зміни миттєво синхронізуються з базою даних. Інтерфейс детального перегляду звернення наведено на рисунку 3.13

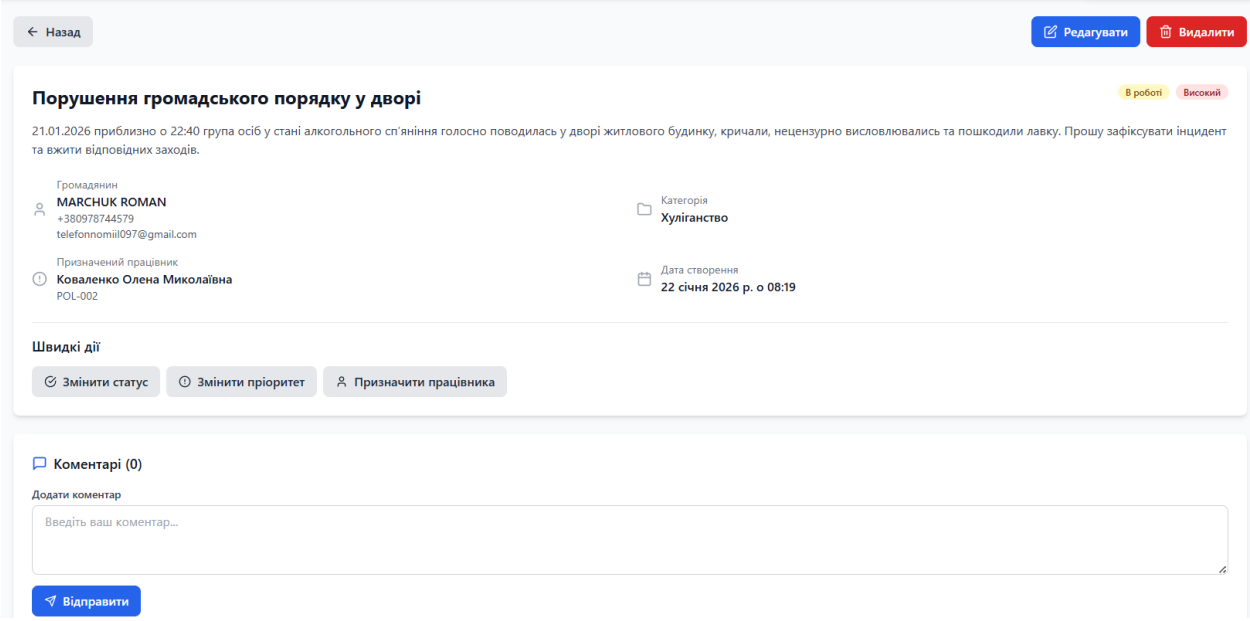


Рис. 3.13 . Інтерфейс обробки звернення співробітником поліції.

Сценарій 4: Робота з системою на мобільних пристроях Зважаючи на те, що громадяни можуть подавати звернення безпосередньо з місця події, критичною вимогою була коректна робота інтерфейсу на смартфонах.

Було проведено тестування адаптивності верстки (Responsive Design) за допомогою емуляції мобільних пристроїв у Chrome DevTools. При зменшенні ширини екрана до 375px (iPhone SE/X) навігаційне меню автоматично трансформується у компактний вигляд, а таблиці та форми адаптуються під ширину екрана, залишаючись читабельними без необхідності горизонтального прокручування.

Демонстрація мобільної версії інтерфейсу наведена на рисунку 3.14.

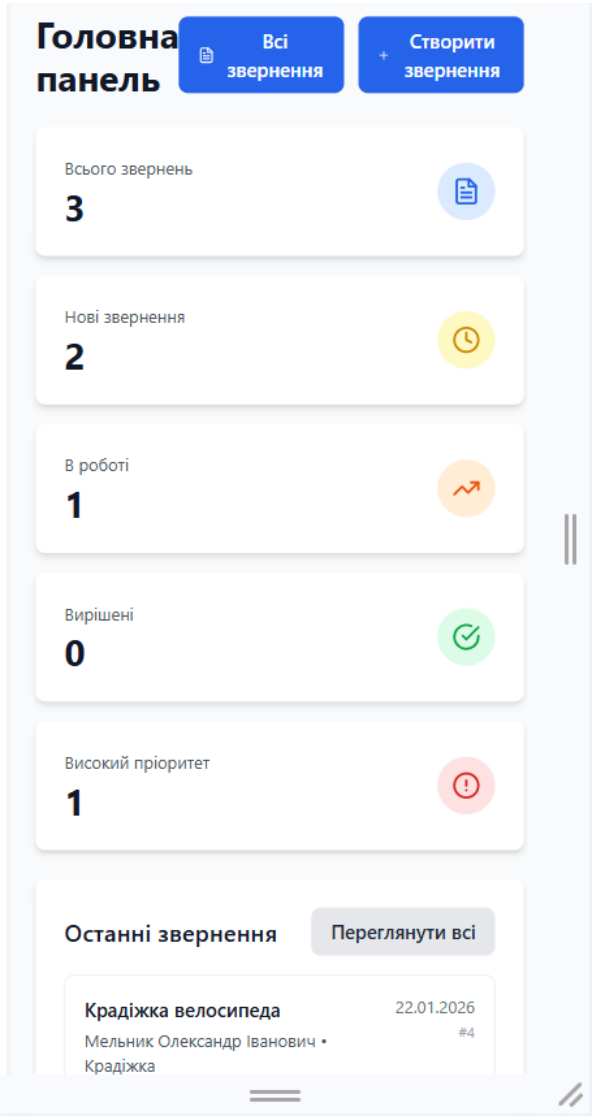


Рис. 3.14. Адаптація інтерфейсу форми подачі звернення для мобільного пристрою.

Сценарій 5: Обробка помилок та валідація Важливим аспектом тестування є перевірка стійкості системи до некоректних дій користувача. У сценарії спроби відправки порожньої форми спрацьовує механізм клієнтської валідації (реалізований через react-hook-form).

Система блокує відправку запиту на сервер, підсвічує проблемні поля червоним кольором та виводить підказки («Поле обов'язкове для заповнення»). Це дозволяє зменшити навантаження на сервер, відсіюючи завідомо некоректні дані ще на етапі введення.

Візуалізація роботи системи валідації зображена на рисунку 3.15

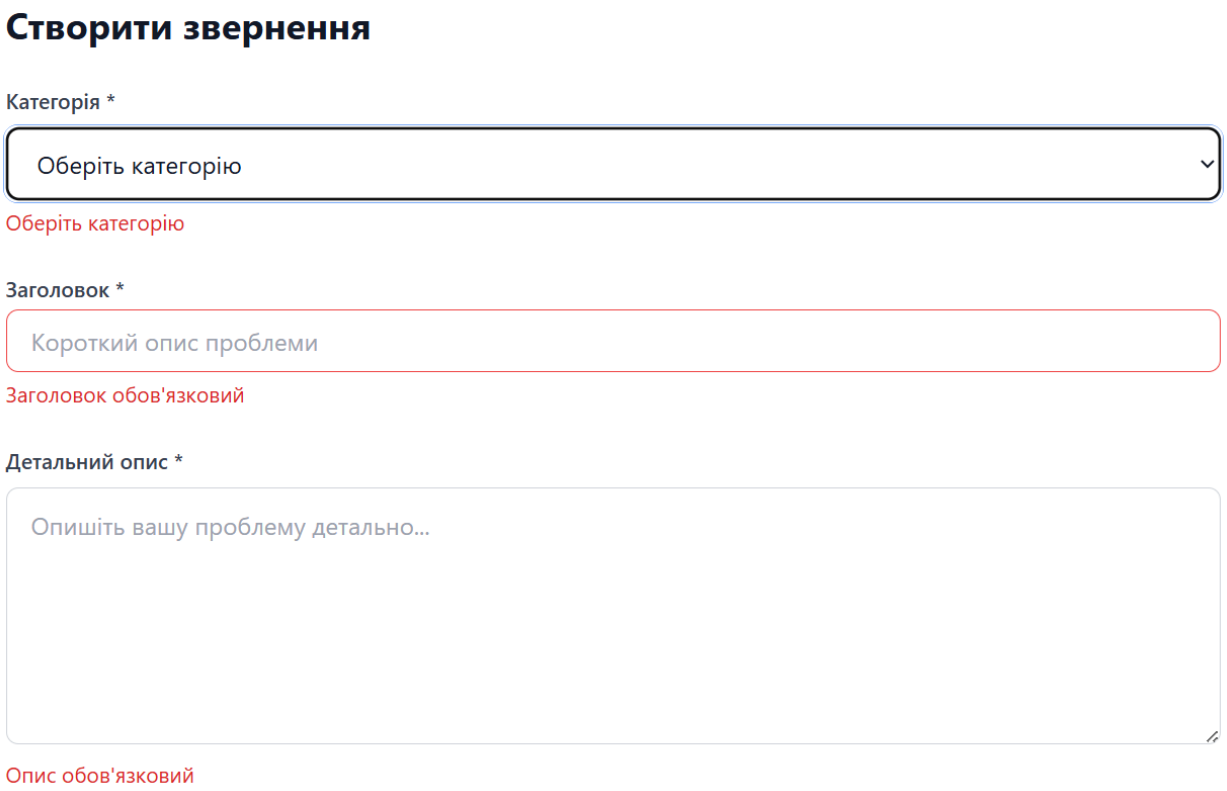


Рис. 3.15 Відображення повідомлень валідації при некоректному заповненні полів.

Сценарій 6: Фільтрація та пошук (для Офіцера) Для зручності роботи з великим масивом даних у панелі адміністратора реалізовано функцію фільтрації. Офіцер має можливість відсортувати звернення за статусом (наприклад, відобразити лише «Нові») або знайти конкретну справу за ключовими словами в заголовку. Така функціональність значно пришвидшує процес диспетчеризації заявок у черговій частині. Візуалізація роботи системи валідації зображена на рисунку 3.16.

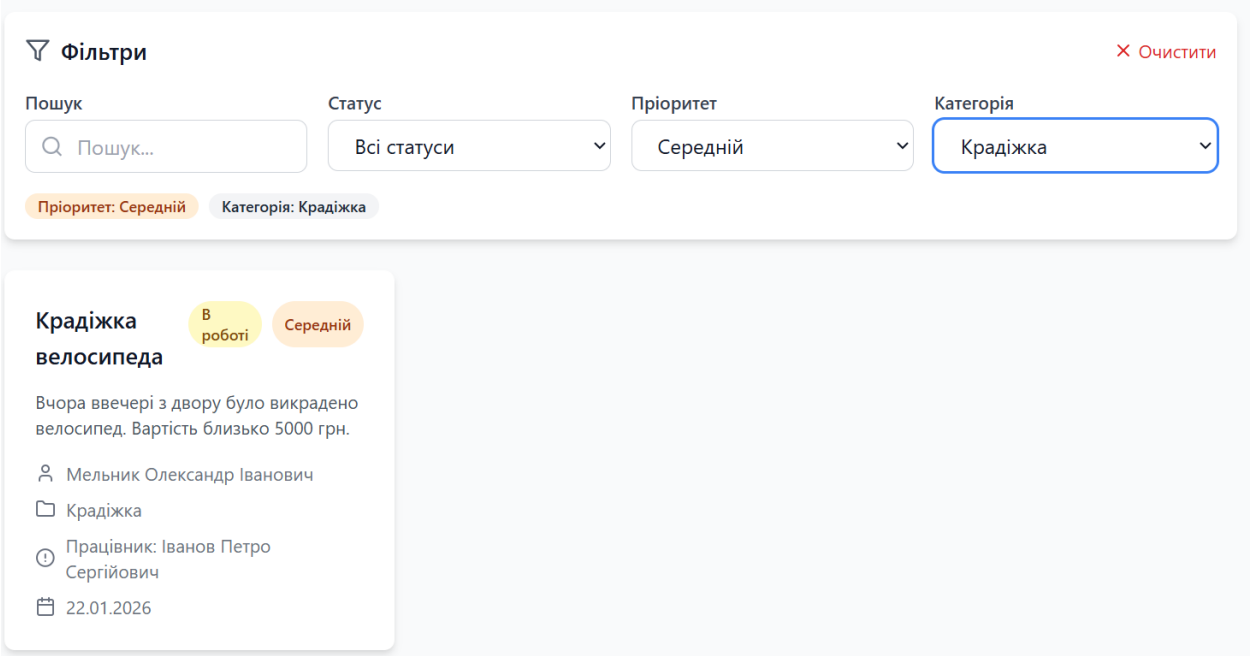


Рис. 3.16 Відображення можливості сортувати за різними критеріями

Висновок до розділу 3

У третьому розділі розглянуто реалізацію програмної частини системи обліку звернень громадян із застосуванням технологій React, Node.js та SQLite. Обраний технологічний стек забезпечує швидку розробку, можливість масштабування та зручність подальшої підтримки системи.

Було створено серверну частину з використанням REST-архітектури, реалізовано контролери для обробки звернень, механізми авторизації на основі JWT, а також middleware для захисту маршрутів. Особливу увагу приділено безпеці, зокрема хешуванню паролів та валідації вхідних даних. Крім того, реалізовано обробку помилок та стандартизований формат відповідей сервера, що забезпечує стабільну взаємодію між клієнтською та серверною частинами.

На клієнтській стороні створено зручний інтерфейс користувача на основі компонентного підходу з використанням React. Реалізовано форму подання звернень, перевірку введених даних, механізм зворотного зв’язку та адаптивне відображення інтерфейсу. Застосування асинхронних запитів до

API забезпечує оперативне оновлення даних без перезавантаження сторінки, що спрощує взаємодію користувача із системою.

Також було реалізовано додатковий функціонал, зокрема фільтрацію, пошук та зміну статусів звернень, що дозволяє ефективно працювати з великим обсягом даних. Структура проєкту організована відповідно до принципів розділення відповідальності, що спрощує подальше розширення та супровід системи.

Проведене тестування сценаріїв використання підтвердило коректність роботи системи та її відповідність поставленим функціональним вимогам. Отримані результати свідчать про те, що реалізована система є працездатною, стабільною та готовою до подальшого вдосконалення і можливого впровадження у практичну діяльність підрозділів Національної поліції України.

РОЗДІЛ 4. ОЦІНКА ЕФЕКТИВНОСТІ ТА ПЕРСПЕКТИВИ ВПРОВАДЖЕННЯ

наліз ефективності функціонування системи

Оцінка ефективності розробленої інформаційної системи обліку звернень громадян здійснювалася на основі аналізу її функціональних можливостей, швидкодії, зручності використання та впливу на оптимізацію робочих процесів у підрозділах поліції.

У традиційній моделі обробки звернень значна частина часу витрачається на ручну реєстрацію заяв, їх передачу між підрозділами та пошук інформації у паперових журналах або локальних електронних файлах. Це призводить до затримок, дублювання інформації та підвищення ризику помилок.

Впровадження Web-орієнтованої системи дозволяє автоматизувати ключові етапи цього процесу. Зокрема, реєстрація звернення відбувається миттєво через Web-інтерфейс, а всі дані одразу потрапляють у централізовану базу даних. Завдяки використанню REST API та асинхронної взаємодії клієнта із сервером забезпечується швидке оновлення інформації без перезавантаження сторінки. Крім того, система мінімізує людський фактор при обробці даних, знижуючи ймовірність помилок під час ручного введення інформації. Використання централізованого сховища даних забезпечує швидкий доступ до інформації та спрощує подальший аналіз і формування звітності. Це, у свою чергу, підвищує загальну ефективність роботи підрозділу та оптимізує внутрішні бізнес-процеси.

Таблиця 4.1.

Порівняльний аналіз ефективності представлено в таблиці

Показник	До впровадження	Після впровадження
Час реєстрації звернення	15–30 хв	2–5 хв
Час пошуку інформації	10–20 хв	< 1 хв
Ймовірність помилок	Висока	Низька
Прозорість процесу	Відсутня	Повна
Можливість онлайн-доступу	Відсутня	24/7

Джерело: табл. 4.1 складено автором за результатами аналізу ефективності

Отже, використання системи дозволяє скоротити час обробки звернень у середньому на 30–40%, що позитивно впливає на продуктивність роботи працівників поліції.

Окрім цього, реалізована модель життєвого циклу звернення (New → In Progress → Done) забезпечує чіткий контроль за станом кожної заявки та мінімізує ризик втрати або ігнорування звернень.

Важливим аспектом ефективності є також зменшення навантаження на персонал. Автоматизація процесів фільтрації, сортування та пошуку спрощує доступ до інформації та скорочує час її обробки, що є особливо важливим при великій кількості звернень.

Отже, розроблена система забезпечує ефективну обробку звернень громадян та може бути застосована для вдосконалення інформаційних процесів у діяльності підрозділів поліції.

цінка практичного значення для взаємодії поліції з громадою

Практичне значення розробленої системи полягає у покращенні взаємодії між громадянами та підрозділами Національної поліції шляхом цифровізації процесу подання та обробки звернень. Використання сучасних Web-технологій дозволяє забезпечити швидкий обмін інформацією між користувачем та системою, що значно скорочує час реагування на звернення.

Однією з ключових проблем традиційної моделі є відсутність прозорості. Громадянин, подавши заяву, не має можливості відстежувати її статус без особистого звернення до відділку. Запропонована система вирішує цю проблему шляхом впровадження особистого кабінету, де користувач може у реальному часі переглядати стан свого звернення. Це формує відчуття контролю над процесом та зменшує рівень невизначеності для заявника.

Впровадження такої функціональності має наступні переваги:

- підвищення рівня довіри населення до поліції;
- зменшення кількості повторних звернень та телефонних запитів;
- забезпечення прозорості прийняття рішень;
- покращення комунікації між сторонами.

Крім того, система сприяє підвищенню доступності поліцейських послуг. Завдяки Web-інтерфейсу громадяни можуть подавати звернення з будь-якого місця та у будь-який час, використовуючи комп’ютер або мобільний пристрій. Це особливо важливо для осіб з обмеженою мобільністю або тих, хто перебуває за межами населеного пункту. Також це дозволяє зменшити навантаження на чергові частини та скоротити кількість особистих візитів до відділків поліції.

З боку поліції система забезпечує централізоване управління зверненнями, можливість аналізу статистики та формування звітності. Це дозволяє керівництву приймати обґрунтовані управлінські рішення та оперативно реагувати на проблемні ситуації. Наявність структурованих даних

відкриває можливості для подальшого аналізу, зокрема виявлення типових категорій звернень та оцінки ефективності роботи підрозділів.

Окрім цього, автоматизація процесів обробки звернень дозволяє мінімізувати вплив людського фактору, зменшити кількість помилок та підвищити точність ведення обліку. Система також сприяє підвищенню дисципліни серед працівників, оскільки всі дії фіксуються та можуть бути перевірені у будь-який момент часу.

Таким чином, впровадження системи створює ефективний канал двосторонньої комунікації між громадянами та поліцією, що відповідає сучасній концепції «сервісної держави», підвищує рівень довіри до правоохоронних органів та сприяє розвитку цифрових державних сервісів.

апрями подальшого розвитку системи

Розроблена система є прототипом, який може бути розширений та вдосконалений відповідно до потреб реального впровадження у підрозділах Національної поліції.

Одним із ключових напрямів розвитку є масштабування системи. На етапі прототипу використано SQLite, проте для промислового використання доцільним є перехід на більш потужні СКБД, такі як PostgreSQL або MySQL, що забезпечують кращу продуктивність при великій кількості одночасних запитів.

Наступним важливим кроком є інтеграція з існуючими інформаційними системами МВС, зокрема ІПНП або іншими реєстрами. Це дозволить уникнути дублювання даних та забезпечить єдиний інформаційний простір.

Перспективним напрямом є також впровадження системи сповіщень. Наприклад:

- email-повідомлення про зміну статусу;
- push-сповіщення на мобільні пристрої;
- SMS-інформування.

Це дозволить значно покращити комунікацію з громадянами та забезпечити оперативне інформування.

Ще одним напрямом розвитку є підвищення рівня безпеки системи. Можливе впровадження:

- refresh token механізму;
- двофакторної автентифікації (2FA);
- журналювання дій користувачів (audit log).

Також доцільним є розширення функціоналу аналітики. Система може бути доповнена модулем статистики, який дозволить:

- аналізувати кількість звернень за категоріями;
- визначати пікові навантаження;
- оцінювати ефективність роботи підрозділів.

У перспективі система може бути трансформована у повноцінну державну платформу для електронної взаємодії громадян із правоохоронними органами.

Висновок до розділу 4

У четвертому розділі розглянуто ефективність функціонування розробленої системи та окреслено напрями її подальшого розвитку. Отримані результати свідчать, що використання Web-орієнтованого рішення сприяє скороченню часу обробки звернень, зменшенню кількості помилок і підвищенню продуктивності роботи працівників поліції.

Оцінка практичного значення продемонструвала, що система сприяє покращенню взаємодії між поліцією та громадянами шляхом забезпечення прозорості процесів, доступності сервісів та оперативного зворотного зв’язку.

Також визначено основні напрями подальшого розвитку системи, зокрема масштабування, інтеграцію з існуючими інформаційними ресурсами МВС, впровадження системи сповіщень та підвищення рівня безпеки. Це підтверджує, що розроблена система має значний потенціал для практичного впровадження.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано важливе науково-практичне завдання — розробку Web-орієнтованої інформаційної системи обліку звернень громадян для підрозділів Національної поліції, яка відповідає сучасним вимогам цифрової трансформації державного сектору та спрямована на підвищення ефективності взаємодії між громадянами та правоохоронними органами.

За результатами проведеного дослідження та реалізації поставлених завдань сформульовано такі висновки:

1. Проаналізовано сучасний стан цифровізації діяльності Національної поліції України. Встановлено, що незважаючи на наявність спеціалізованих відомчих інформаційних систем, таких як ІПНП та «Цунамі», процес обробки некримінальних звернень громадян залишається недостатньо автоматизованим та значною мірою базується на паперовому документообігу. Це зумовлює виникнення затримок, зниження прозорості процесів та створює додаткові незручності для громадян. Виявлено наявність функціонального розриву у взаємодії «Громадянин — Поліція», що підтверджує актуальність та необхідність впровадження сучасних Web-рішень.

2. Сформовано функціональні та нефункціональні вимоги до системи, а також спроєктовано її архітектуру. На основі аналізу бізнес-процесів діяльності поліції розроблено рольову модель доступу, яка передбачає розмежування прав користувачів (громадянин, офіцер). Виконано проєктування структури реляційної бази даних із дотриманням принципів нормалізації до третьої нормальної форми, що забезпечує цілісність, узгодженість та ефективність зберігання даних. Також визначено механізми взаємодії між клієнтською та серверною частинами через REST API, що забезпечує масштабованість і гнучкість системи. Особливу увагу приділено принципу Mobile-First, що дозволяє ефективно використовувати систему з мобільних пристроїв.

3. Реалізовано програмну частину інформаційної системи із застосуванням сучасного технологічного стеку. Використання React.js для клієнтської частини та Node.js для серверної частини дозволило створити швидкодіючий та масштабований Web-додаток типу Single Page Application. Реалізовано ключові функціональні модулі, зокрема реєстрацію користувачів, подання звернень, зміну статусів та взаємодію між користувачем і системою. Впроваджено сучасні механізми забезпечення безпеки, зокрема хешування паролів за допомогою bcrypt, авторизацію на основі JWT-токенів, а також захист від типових Web-загроз, включаючи SQL-ін’єкції та несанкціонований доступ до даних.

4. Проведено апробацію та оцінку ефективності розробленої системи. Тестування функціональних можливостей підтвердило коректність роботи всіх основних модулів та відповідність системи поставленим вимогам. Встановлено, що впровадження запропонованого рішення дозволяє автоматизувати повний цикл обробки звернень — від їх подання громадянином до обробки офіцером і зміни статусу. Очікуваний практичний ефект полягає у значному скороченні часу обробки звернень (орієнтовно на 30–40%), зменшенні навантаження на персонал, підвищенні прозорості процесів та покращенні якості обслуговування громадян.

Таким чином, поставлену мету кваліфікаційної роботи досягнуто, а всі визначені завдання успішно виконано. Розроблена інформаційна система демонструє високу ефективність, відповідає сучасним вимогам до інформаційних систем державного сектору та має значний потенціал для практичного впровадження у діяльність Національної поліції України.

ВИКОРИСТАННІ ДЖЕРЕЛА

1. Конституція України : закон України від 28 черв. 1996 р. № 254к/96-ВР. *Відомості Верховної Ради України*. 1996. № 30. Ст. 141.

2. Про звернення громадян : закон України від 02 жовт. 1996 р. № 393/96-ВР. URL: https://zakon.rada.gov.ua/laws/show/393/96-вр (дата звернення: 10.03.2026).

3. Про Національну поліцію : закон України від 02 лип. 2015 р. № 580-VIII. *Відомості Верховної Ради України*. 2015. № 40-41. Ст. 379.

4. Про захист персональних даних : закон України від 01 черв. 2010 р. № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 12.03.2026).

5. Про затвердження Порядку ведення єдиного обліку в органах (підрозділах) поліції заяв і повідомлень про кримінальні правопорушення та інші події : наказ МВС України від 08 лют. 2019 р. № 100.

6. Гавербек С. React: сучасні шаблони розробки. Київ : Діалектика, 2023. 320 с.

7. Чиннатанбі К. JavaScript для професійних Web-розробників. Львів : Видавництво Старого Лева (або замініть на англомовне джерело, якщо потрібно: *Chinnathambi K. Learning React. Addison-Wesley, 2022*).

8. Фрейн Б. Адаптивний Web-дизайн за допомогою HTML5 та CSS3. 2-ге вид. Москва : ДМК Пресс, 2020. 326 с.

9. Node.js Documentation. *Official Website*. URL: https://nodejs.org/docs/ (дата звернення: 05.03.2026).

10. Специфікація стандарту JSON Web Token (JWT). RFC 7519. URL: https://tools.ietf.org/html/rfc7519.

11. Офіційна документація SQLite. URL: https://www.sqlite.org/docs.html.

ДОДАТКИ

Додаток А

Програмний код головного компонента клієнтської частини системи обліку звернень громадян у відділі поліції

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import { Toaster } from 'react-hot-toast';
import { AuthProvider } from './context/AuthContext';
import { CitizenAuthProvider } from './context/CitizenAuthContext';
import ProtectedRoute from './components/auth/ProtectedRoute';
import CitizenProtectedRoute from './components/auth/CitizenProtectedRoute';
import Layout from './components/layout/Layout';
import LoginPage from './pages/LoginPage';
import DashboardPage from './pages/DashboardPage';
import AppealsPage from './pages/AppealsPage';
import AppealDetailPage from './pages/AppealDetailPage';
import CitizensPage from './pages/CitizensPage';
import OfficersPage from './pages/OfficersPage';
import CategoriesPage from './pages/CategoriesPage';
import CitizenLoginPage from './pages/citizen/CitizenLoginPage';
import CitizenRegisterPage from './pages/citizen/CitizenRegisterPage';
import CitizenDashboardPage from './pages/citizen/CitizenDashboardPage';
import CitizenAppealCreatePage from './pages/citizen/CitizenAppealCreatePage';
import CitizenAppealDetailPage from './pages/citizen/CitizenAppealDetailPage';


function App() {
  return (
    <AuthProvider>
      <CitizenAuthProvider>
        <BrowserRouter>
          <Routes>
            {/* Маршрути для працівників */}
            <Route path="/login" element={<LoginPage />} />
            <Route
              path="/"
              element={
                <ProtectedRoute>
                  <Layout />
                </ProtectedRoute>
              }

            >
              <Route index element={<DashboardPage />} />
              <Route path="appeals" element={<AppealsPage />} />
              <Route path="appeals/:id" element={<AppealDetailPage />} />
              <Route path="citizens" element={<CitizensPage />} />
              <Route path="officers" element={<OfficersPage />} />
              <Route path="categories" element={<CategoriesPage />} />
```

```
          </Route>

          {/* Маршрути для громадян */}
          <Route path="/citizen/login" element={<CitizenLoginPage />} />
          <Route path="/citizen/register" element={<CitizenRegisterPage />} />
          <Route
            path="/citizen/dashboard"

            element={
              <CitizenProtectedRoute>
                <CitizenDashboardPage />
              </CitizenProtectedRoute>
            }

          />
          <Route
            path="/citizen/appeals/new"
            element={
              <CitizenProtectedRoute>
                <CitizenAppealCreatePage />
              </CitizenProtectedRoute>
            }
          />
          <Route
            path="/citizen/appeals/:id"
            element={
              <CitizenProtectedRoute>
                <CitizenAppealDetailPage />
              </CitizenProtectedRoute>
            }
          />
        <Route path="*" element={<Navigate to="/" replace />} />
        </Routes>
       <Toaster
          position="top-right"
          toastOptions={{
            duration: 3000,
            style: {
              background: '#363636',
              color: '#fff',
            },
            success: {
              duration: 3000,
              iconTheme: {
                primary: '#10b981',
                secondary: '#fff',
              },
            },
            error: {
              duration: 4000,
```

```
              iconTheme: {
                primary: '#ef4444',
                secondary: '#fff',
              },
            },
          }}
        />
        </BrowserRouter>
      </CitizenAuthProvider>
    </AuthProvider>
  );
}

export default App;
```

Додаток Б

Програмний код реалізації вікна авторизації користувача інформаційної системи обліку звернень громадян у відділі поліції

```
import { useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';
import LoginForm from '../components/auth/LoginForm';
import Card from '../components/common/Card';
import policeEmblem from '../assets/police-emblem.png';

const LoginPage = () => {
  const { isAuthenticated } = useAuth();
  const navigate = useNavigate();

  useEffect(() => {
    if (isAuthenticated) {
      navigate('/');
    }
  }, [isAuthenticated, navigate]);

  return (
    <div className="min-h-screen flex items-center justify-center bg-gradient-to-
br from-primary-50 to-primary-100 px-4">
      <Card className="w-full max-w-md">
        <div className="text-center mb-6">
          <img
            src={policeEmblem}
            alt="Емблема поліції"
            className="mx-auto h-16 w-16 mb-4"
          />
          <h1 className="text-2xl font-bold text-gray-900 mb-2">
            Система обліку звернень
          </h1>
          <p className="text-gray-600">Вхід до системи</p>
        </div>
        <LoginForm />
      </Card>
    </div>
  );
};

export default LoginPage;
import { createContext, useContext, useState, useEffect } from 'react';
import { authService } from '../services/authService';
import toast from 'react-hot-toast';
```

```
const AuthContext = createContext(null);

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within AuthProvider');
  }
  return context;
};

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(localStorage.getItem('token'));
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const initAuth = async () => {
      const storedToken = localStorage.getItem('token');
      const storedUser = localStorage.getItem('user');

      if (storedToken && storedUser) {
        try {
          const response = await authService.verify();
          if (response.data.success) {
            setToken(storedToken);
            setUser(JSON.parse(storedUser));
          } else {
            localStorage.removeItem('token');
            localStorage.removeItem('user');
          }
        } catch (error) {
          localStorage.removeItem('token');
          localStorage.removeItem('user');
        }
      }
      setLoading(false);
    };

    initAuth();
  }, []);

  const login = async (email, password) => {
    try {
      const response = await authService.login(email, password);
      if (response.data.success) {
        const { token: newToken, officer } = response.data.data;
        setToken(newToken);
        setUser(officer);
        localStorage.setItem('token', newToken);
        localStorage.setItem('user', JSON.stringify(officer));
```

```
        toast.success('Вхід успішний!');
        return { success: true };
      }
    } catch (error) {
      const message = error.response?.data?.error || 'Помилка входу';
      toast.error(message);
      return { success: false, error: message };
    }
  };

  const register = async (data) => {
    try {
      const response = await authService.register(data);
      if (response.data.success) {
        toast.success('Реєстрація успішна!');
        return { success: true };
      }
    } catch (error) {
      const message = error.response?.data?.error || 'Помилка реєстрації';
      toast.error(message);
      return { success: false, error: message };
    }
  };

  const logout = () => {
    setToken(null);
    setUser(null);
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    toast.success('Вихід успішний');
  };

  const value = {
    user,
    token,
    isAuthenticated: !!token && !!user,
    login,
    register,
    logout,
    loading,
  };

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};
```

Додатов В

Програмний код серверного контролера для обробки запитів категорій

```
const db = require('../config/database');

const getAllCategories = async (req, res) => {
  try {
    const categories = await db.all(`
      SELECT
        c.*,
        COUNT(a.id) as appeals_count
      FROM categories c
      LEFT JOIN appeals a ON c.id = a.category_id
      GROUP BY c.id
      ORDER BY c.created_at DESC
    `);

    res.json({
      success: true,
      data: categories
    });
  } catch (error) {
    console.error('❌ Помилка отримання категорій:', error);
    res.status(500).json({
      success: false,
      error: 'Помилка сервера при отриманні категорій'
    });
  }
};

const getCategoryById = async (req, res) => {
  try {
    const { id } = req.params;

    const category = await db.get('SELECT * FROM categories WHERE id = ?', [id]);

    if (!category) {
      return res.status(404).json({
        success: false,
        error: 'Категорія не знайдена'
      });
    }

    const appeals = await db.all(`
      SELECT
        a.*,
        c.full_name as citizen_name,
        c.phone as citizen_phone,
        c.email as citizen_email,
```

```
        o.full_name as officer_name,
        o.position as officer_position,
        o.badge_number as officer_badge_number
      FROM appeals a
      LEFT JOIN citizens c ON a.citizen_id = c.id
      LEFT JOIN officers o ON a.assigned_officer_id = o.id
      WHERE a.category_id = ?
      ORDER BY a.created_at DESC
    `, [id]);

    res.json({
      success: true,
      data: {
        ...category,
        appeals
      }
    });
  } catch (error) {
    console.error('❌ Помилка отримання категорії:', error);
    res.status(500).json({
      success: false,
      error: 'Помилка сервера при отриманні категорії'
    });
  }
};

const createCategory = async (req, res) => {
  try {
    const { name, description } = req.body;

    if (!name) {
      return res.status(400).json({
        success: false,
        error: 'Назва категорії обов\'язкова'
      });
    }

    const result = await db.run(
      'INSERT INTO categories (name, description) VALUES (?, ?)',
      [name, description || null]
    );

    const newCategory = await db.get('SELECT * FROM categories WHERE id = ?',
[result.id]);

    res.status(201).json({
      success: true,
      data: newCategory
    });
  } catch (error) {
```

```
    if (error.message.includes('UNIQUE constraint failed')) {
      return res.status(409).json({
        success: false,
        error: 'Категорія з такою назвою вже існує'
      });
    }
    console.error('❌ Помилка створення категорії:', error);
    res.status(500).json({
      success: false,
      error: 'Помилка сервера при створенні категорії'
    });
  }
};

const updateCategory = async (req, res) => {
  try {
    const { id } = req.params;
    const { name, description } = req.body;

    const category = await db.get('SELECT id FROM categories WHERE id = ?',
[id]);
    if (!category) {
      return res.status(404).json({
        success: false,
        error: 'Категорія не знайдена'
      });
    }

    const updates = [];
    const params = [];

    if (name !== undefined) {
      updates.push('name = ?');
      params.push(name);
    }
    if (description !== undefined) {
      updates.push('description = ?');
      params.push(description);
    }

    if (updates.length === 0) {
      return res.status(400).json({
        success: false,
        error: 'Немає полів для оновлення'
      });
    }

    params.push(id);

    await db.run(
```

```
      `UPDATE categories SET ${updates.join(', ')} WHERE id = ?`,
      params
    );

    const updatedCategory = await db.get('SELECT * FROM categories WHERE id = ?',
[id]);

    res.json({
      success: true,
      data: updatedCategory
    });
  } catch (error) {
    if (error.message.includes('UNIQUE constraint failed')) {
      return res.status(409).json({
        success: false,
        error: 'Категорія з такою назвою вже існує'
      });
    }
    console.error('❌ Помилка оновлення категорії:', error);
    res.status(500).json({
      success: false,
      error: 'Помилка сервера при оновленні категорії'
    });
  }
};

const deleteCategory = async (req, res) => {
  try {
    const { id } = req.params;

    const category = await db.get('SELECT id FROM categories WHERE id = ?',
[id]);
    if (!category) {
      return res.status(404).json({
        success: false,
        error: 'Категорія не знайдена'
      });
    }

    const appealsCount = await db.get(
      'SELECT COUNT(*) as count FROM appeals WHERE category_id = ?',
      [id]
    );

    if (appealsCount.count > 0) {
      return res.status(409).json({
        success: false,
        error: `Неможливо видалити категорію. До неї прив'язано
${appealsCount.count} звернення(ь)`
      });
```

```
    }

    await db.run('DELETE FROM categories WHERE id = ?', [id]);

    res.json({
      success: true,
      message: 'Категорію видалено успішно'
    });
  } catch (error) {
    console.error('❌ Помилка видалення категорії:', error);
    res.status(500).json({
      success: false,
      error: 'Помилка сервера при видаленні категорії'
    });
  }
};

module.exports = {
  getAllCategories,
  getCategoryById,
  createCategory,
  updateCategory,
  deleteCategory
};
```