

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ
SOFTWARE FOR PARKING INSPECTION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-212сп
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

РАТУШНИЙ Р.В.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

РАТУШНОГО Романа Вікторовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення паркувальної інспекції.
Software for Parking Inspection

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Роман РАТУШНИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 102 с., 47 рис., 5 табл., 3 дод., 13 джерел, 17 лістингів.

ПАРКУВАННЯ, ПОРУШЕННЯ, ШТРАФ, ІНСПЕКТОР, ВЕБЗАСТОСУНОК, АВТОМАТИЗАЦІЯ, РЕЄСТР ПОРУШЕНЬ.

Об'єкт роботи – процес фіксації та обробки порушень правил паркування.

Предмет роботи – програмне забезпечення для автоматизації діяльності паркувальної інспекції.

Метою роботи є розробка вебзастосунку, який автоматизує процеси взаємодії між інспекторами, адміністраторами та користувачами у сфері контролю за паркуванням, з можливістю створення протоколів, додавання фотофіксацій, обробки оплат та адміністрування користувачів.

Матеріали, методи та технічні засоби: JavaScript, Node.js (Express.js), React, MongoDB, PostgreSQL, Sequelize, Mongoose, JWT, Docker.

Результати. У результаті виконання роботи створено повнофункціональний вебзастосунок для автоматизації процесів паркувальної інспекції. Реалізовано авторизацію та реєстрацію користувачів, призначення ролей, створення та перегляд протоколів, завантаження зображень порушень, обробку оплат, функції адміністрування, ведення логів та повний життєвий цикл взаємодії між учасниками системи. Система дозволяє контролювати та оптимізувати діяльність інспекторів, вести облік порушень і покращити зручність для звичайних користувачів.

Висновки. Розроблений застосунок забезпечує автоматизацію ключових процесів у сфері паркувальної інспекції, підвищує ефективність контролю за порушеннями, зменшує навантаження на адміністративний персонал та покращує взаємодію з користувачами. Програмне забезпечення

може бути впроваджене у муніципальні служби для модернізації контролю за паркуванням у містах.

Галузь використання. Муніципальні служби паркувального контролю, автоматизовані системи обліку порушень, Smart City-платформи.

ABSTRACT

Explanatory note to the bachelor's qualification thesis: 102 pages, 47 figures, 5 tables, 3 appendixes, 13 sources, 17 listings.

PARKING, VIOLATION, FINE, INSPECTOR, WEB APPLICATION, AUTOMATION, VIOLATION REGISTER.

Object of the work – the process of recording and processing parking rule violations.

Subject of the work – software for automating the activities of the parking inspection service.

The aim of the work is to develop a web application that automates the interaction processes between inspectors, administrators, and users in the field of parking control, with the ability to create violation reports, attach photographic evidence, process payments, and manage user accounts.

Materials, methods, and technical tools: JavaScript, Node.js (Express.js), React, MongoDB, PostgreSQL, Sequelize, Mongoose, JWT, Docker.

Results. As a result of the work, a full-featured web application was developed to automate the processes of parking inspection. User authentication and registration, role assignment, creation and viewing of violation reports, image upload, payment processing, administration functions, activity logging, and the full interaction cycle between system participants were implemented. The system allows for monitoring and optimizing inspector activities, maintaining a record of violations, and improving user convenience.

Conclusions. The developed application provides automation of key processes in the field of parking inspection, increases the efficiency of violation control, reduces the workload on administrative personnel, and improves interaction with users. The software can be implemented in municipal services to modernize parking control in urban areas.

Field of application. Municipal parking control services, automated violation recording systems, Smart City platforms.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	10
Вступ.....	11
1 Аналіз предметної області.....	13
1.1 Аналіз програмного забезпечення для автоматизації процесів паркувальної інспекції.....	13
1.2 Порівняльний аналіз розробки та аналогів	14
1.3 Висновки за розділом 1	16
2 Розробка архітектури програми.....	18
2.1 Вибір засобів проєктування	18
2.2 Сценарії використання програми	20
2.3 Алгоритм роботи програми	25
2.4 Структурна схема модулів програми.....	27
2.5 Концептуальна схема програми	29
2.6 Структура бази даних	31
2.6 Визначення порядку роботи програми	36
2.7 Висновки за розділом 2	37
3 Реалізація програми	38
3.1 Реалізація клієнтської частини (Frontend)	38
3.2 Реалізація бекенд-частини застосунку.....	41
3.3 Реалізація повного циклу функціоналу (Full Stack)	45
3.4 Розгортання застосунку (деплой).....	48
3.5 Висновки за розділом 3	52
4 Експлуатація програми.....	54
4.1 Призначення програми.....	54
4.2 Умови виконання програми.....	54
4.3 Виконання і тестування програми.....	55
4.4 Висновки за розділом 4	72
Висновки	74
Перелік джерел посилання	76

	9
Додаткок А Текст програми.....	78
Додаток Б Технічне завдання.....	83
Додаток В Слайди презентації.....	92

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ВФ – вебфреймворк.

ВСТУП

У сучасних містах проблема контролю за дотриманням правил паркування стає все більш актуальною. Неправильне паркування створює значні незручності для водіїв, пішоходів та громадського транспорту, що може призводити до заторів, аварійних ситуацій та ускладнення роботи екстрених служб. Для забезпечення порядку в цій сфері муніципальні органи запроваджують системи контролю та штрафкування порушників. Одним із ключових елементів такого контролю є діяльність паркувальних інспекторів, які фіксують правопорушення та складають відповідні адміністративні протоколи.

Проте процес фіксації порушень часто є трудомістким та потребує значних часових і людських ресурсів. У всіх випадках порушення мають бути задокументовані за допомогою фотофіксації, щоб уникнути безпідставних штрафів та забезпечити можливість доказової бази у разі оскарження штрафу в судовому порядку. Крім того, паркувальним інспекторам необхідно зберігати велику кількість інформації, що стосується кожного зафіксованого випадку, включаючи службові примітки та інші дані.

Одним із основних викликів у цій сфері є необхідність ефективного збору, зберігання та обробки такої інформації. Використання традиційних методів, таких як паперові протоколи або ручне ведення електронних таблиць, часто призводить до втрати даних, ускладнює доступ до інформації та збільшує ймовірність помилок.

Автоматизація роботи паркувальних інспекторів дозволяє підвищити ефективність фіксації порушень, забезпечити зручний доступ до фотофіксації та службових приміток, а також гарантувати безпеку та конфіденційність збережених даних. Використання сучасних технологій може значно зменшити кількість помилок, оптимізувати робочий процес інспекторів та сприяти покращенню взаємодії між різними службами, відповідальними за контроль паркування.

Метою даної роботи є розробка програмного забезпечення для автоматизації процесів паркувальної інспекції, яке дозволить спростити та вдосконалити роботу паркувальних інспекторів, забезпечуючи ефективний збір, збереження та обробку інформації про порушення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз програмного забезпечення для автоматизації процесів паркувальної інспекції

Під час дослідження було знайдено два застосунки, які так чи інакше стосуються сфери контролю за дотриманням правил паркування, при цьому не вирішуючи проблему в повній мірі.

Один зі знайдених застосунків - це програмний комплекс від Parkt Service Group [1].

Компанія Parkt Service Group спеціалізується на розробці та впровадженні програмних рішень для автоматизації процесів контролю за паркуванням у міських умовах. Їхній програмний комплекс спрямований на оптимізацію роботи інспекцій з паркування, забезпечуючи ефективне виявлення та обробку порушень правил зупинки, стоянки та паркування транспортних засобів [1].

Основні функції програмного комплексу від Parkt Service Group [1]:

- система автоматизує процес створення та обробки постанов, що зменшує час на адміністративні процедури та підвищує точність даних;
- програмне забезпечення включає модулі для координації та обліку процесів евакуації транспортних засобів, що порушують правила паркування, забезпечуючи прозорість та ефективність цих дій;
- рішення від Parkt Service Group може бути інтегровано з існуючими міськими інформаційними системами, що сприяє централізованому управлінню та моніторингу ситуації з паркуванням.

Також, у ході дослідження теми автоматизації контролю за паркуванням, вдалося виявити цікаву інформацію, що підтверджує актуальність цієї проблеми в Україні. Зокрема, одним із прикладів впровадження подібних рішень є тендер, оголошений на розробку програмного забезпечення для автоматизації контролю за дотриманням правил паркування в місті Дніпро [2].

Цей тендер свідчить про зацікавленість місцевих органів влади у впровадженні сучасних технологічних рішень для управління паркувальним простором. Визначене в тендері технічне завдання містить чіткий перелік вимог до програмного забезпечення, що демонструє комплексний підхід до вирішення проблеми порушень правил паркування [2].

Серед ключових аспектів технічного завдання можна виділити наступне [2]:

- програмне забезпечення повинно мати можливість взаємодії з державними реєстрами транспортних засобів, що дозволить ідентифікувати порушників та прискорить процес обробки даних;
- система має забезпечувати можливість автоматичної фіксації транспортних засобів, що припарковані з порушенням правил, із збереженням відповідних зображень та GPS-координат місця події;
- автоматизація процесу винесення штрафів і формування адміністративних постанов допоможе значно зменшити навантаження на інспекторів та підвищити ефективність контролю;
- передбачена можливість створення детальних звітів та аналітики щодо дотримання правил паркування, що допоможе муніципальним органам краще оцінювати ситуацію та приймати відповідні рішення.

Варто зазначити, що аналогічні рішення можуть бути впроваджені й в інших містах, що дозволить зменшити кількість порушень, покращити контроль за паркуванням і сприяти підвищенню комфорту для всіх учасників дорожнього руху.

1.2 Порівняльний аналіз розробки та аналогів

Обидва аналоги мають переваги і недоліки. Для зручності порівняння доцільно представити дані у вигляді таблиці, в якій будуть порівнюватись існуючі рішення та розробка (табл. 1.1).

Таблиця 1.1 — Порівняння програм для автоматизації процесів роботи паркувальної інспекції

Критерій	Розробка	Parkt Service Group	Автоматизація в м. Дніпро
1	2	3	4
Централізоване зберігання доказів	Так.	Так	Так
Фіксація правопорушень	Автоматизований процес з GPS-прив'язкою, що унеможлиблює підробку даних.	Є така можливість, але без прив'язки до GPS.	Автоматизований процес з GPS-прив'язкою, що унеможлиблює підробку даних.
Формування постанов та службових приміток	Автоматизоване створення постанов та можливість додавання службових приміток.	Постанови створюються, але примітки потрібно додавати вручну через сторонні системи.	Частково автоматизовано, але примітки не передбачені.
Контроль доступу	Гнучка система розмежування прав доступу, що унеможлиблює несанкціонований доступ.	Базова авторизація без розширеного управління ролями.	Відсутні детальні рівні доступу для інспекторів та адміністраторів.
Адмін панель	Повноцінна адмінка з можливістю керування користувачами та контролю процесів.	Немає	Є

Порівняння продемонструвало, що аналоги мають суттєві обмеження у функціоналі порівняно з представленою розробкою. Зокрема, рішення від Parkt Service Group не передбачає GPS-прив'язку при фіксації правопорушень, що може ускладнювати доказову базу в спірних випадках [1]. Крім того, їхня система не забезпечує автоматизованого додавання службових приміток [1], що змушує користувачів звертатися до сторонніх сервісів, ускладнюючи адміністрування процесу. Також їхній рівень безпеки залишається базовим, що створює ризики для захисту даних [1].

Автоматизована система контролю паркування в м. Дніпро, хоч і підтримує GPS-прив'язку для фіксації порушень, але не має повноцінної системи управління доступом [2], що може призвести до витоку або несанкціонованого використання даних. Крім того, вона не підтримує додавання службових приміток [2], що обмежує можливості фіксації додаткової інформації щодо кожного порушення.

На відміну від аналогів, запропонована розробка забезпечує високий рівень автоматизації, включаючи централізоване зберігання доказів, захист інформації за допомогою сучасних технологій шифрування, розширену адміністративну панель та контроль доступу на рівні ролей. Це дозволяє значно підвищити ефективність роботи інспекторів, мінімізувати ризики помилок та несанкціонованого доступу, а також покращити загальний рівень організації процесу контролю за паркуванням.

1.3 Висновки за розділом 1

Завдання роботи:

- проаналізувати програмне забезпечення для автоматизації процесів паркувальної інспекції;
- визначити засоби проєктування системи;
- сформулювати вимоги до програмного забезпечення для контролю за паркуванням;

– реалізувати систему, що відповідає поставленим вимогам.

Аналіз існуючих рішень у сфері автоматизації паркувальної інспекції дозволив виявити, що доступні програмні комплекси мають низку недоліків. Зокрема, вони або не забезпечують повноцінної автоматизації процесів фіксації порушень, або мають обмежену функціональність щодо адміністрування та безпеки даних. Деякі рішення, такі як система Parkt Service Group, не підтримують GPS-прив'язку при фіксації порушень і мають обмежені можливості для керування доступом. Інші, наприклад, система автоматизації контролю в м. Дніпро, хоча й включають GPS-прив'язку, не передбачають розширеного адміністрування та багаторівневого захисту даних.

На основі проведеного аналізу визначено ключові вимоги до розробки, які забезпечать високий рівень автоматизації, розширену адміністративну панель, безпечне зберігання доказів та надійний контроль доступу. Реалізація такого рішення дозволить підвищити ефективність роботи інспекторів, зменшити кількість порушень та покращити процес контролю за дотриманням правил паркування. Всі ці аспекти будуть детально розглянуті та реалізовані в наступних розділах роботи.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування

В умовах сучасної веброзробки доцільно використовувати фреймворки, які забезпечують високу швидкість розробки, масштабованість, активну спільноту та велику кількість готових рішень [3]-[5]. З огляду на це, для реалізації вебзастосунку було обрано React як фронтенд-фреймворк і Express.js для побудови серверної частини (бекенду).

Для обґрунтування вибору були розглянуті альтернативні технології: для клієнтської частини – Vue.js, для серверної – Koa.js, що також мають широку популярність. Порівняння засобів наведено в таблицях 2.1 та 2.2.

Таблиця 2.1 – Порівняння засобів для розробки фронтенд частини програми для паркувальної інспекції

Критерій	React	Vue.js
1	2	3
Архітектура	Гнучка, побудова на компонентах, повна свобода організації	Компонентна, з чіткішими структурами "з коробки"
Крива навчання	Потребує глибшого розуміння JavaScript та суміжних бібліотек	Простий в освоєнні для новачків
Масштабованість	Ідеально підходить для великих, складних проєктів	Підходить переважно для малих і середніх проєктів
Підтримка екосистеми	Широка екосистема, підтримка Next.js, Redux, React Query	Власна екосистема, Nuxt.js, Vuex

Продовження таблиці 2.1

1	2	3
Гнучкість у виборі інструментів	Висока	Нижча, більшість рішень інтегровані в сам Vue
Спільнота	Надзвичайно велика, постійно оновлювана	Менша, але активно розвивається

Таблиця 2.2 – Порівняння засобів для розробки бекенд частини програми для паркувальної інспекції

Критерій	Express.js	Koa.js
Складність	Простий, мінімалістичний фреймворк	Ще легший, але потребує більше налаштувань вручну
Швидкість старту	Висока, мінімум конфігурації	Потребує додаткових модулів для базової функціональності
Сумісність з middleware	Величезна кількість готових middleware	Підтримує сучасні async/await, але менш поширений
Документація та підтримка	Широка документація, багаторічна історія	Менш поширений, але зручний для досвідчених розробників
Ідеально підходить для	CRUD-додатків, REST API, швидкої побудови backend'у	Високонавантажених застосунків, потребує кастомних рішень

Враховуючи складність функціоналу, який необхідно реалізувати (робота з доказами правопорушень, адміністрування, контроль доступу, формування постанов тощо), обрано React як фронтенд-фреймворк, який надає повну гнучкість у створенні компонентів [6], глибоку інтеграцію з іншими інструментами [7], зокрема Redux або React Query для управління станом [8], а також має сильну спільноту та підтримку [9].

На серверній частині доцільним було використання Express.js як найпопулярнішого та перевіреного фреймворку для Node.js, що дозволяє швидко розгортати REST API [10], легко інтегрується з базами даних, має величезну кількість middleware для реалізації авторизації, логування, обробки файлів тощо. Хоча Koa.js більш сучасний і мінімалістичний [11], його використання потребує більшого досвіду, а також самостійної реалізації багатьох рішень, які в Express вже доступні «з коробки» [12]-[13].

Використання зв'язки React + Express дозволяє розробити надійний [14-15], масштабований і зручний для підтримки вебзастосунків із централізованим зберіганням даних [15], авторизацією, гнучкою панеллю адміністрування та зручною взаємодією інспекторів із системою [16]-[17].

2.2 Сценарії використання програми

У вебзастосунку реалізовано поділ користувачів на адміністративний персонал та працівників паркувальної інспекції, що є однією з ключових функціональних вимог до системи. Такий розподіл ролей забезпечує контроль доступу до окремих модулів застосунку, що підвищує гнучкість у керуванні правами користувачів і гарантує безпеку при роботі з конфіденційною інформацією. Функціональні вимоги також передбачають підтримку повного циклу обробки порушень правил паркування — від виявлення і фіксації правопорушення до збереження протоколу, відображення статусу його оплати та адміністрування даних користувачів системи.

Нижче наведено основні функціональні сценарії у форматі User Stories, що дозволяють описати очікувану поведінку системи з погляду кінцевого користувача.

Авторизація та керування профілем:

- як користувач, я хочу мати можливість авторизуватися, щоб отримати доступ до захищених розділів системи;

- як користувач, я хочу переглядати свій профіль, щоб контролювати персональну інформацію;

- як користувач, я хочу редагувати свій профіль, щоб оновити особисті дані.

Адміністративна панель:

- як адміністратор, я хочу переглядати список усіх зареєстрованих користувачів, щоб контролювати склад системи;

- як адміністратор, я хочу блокувати та розблоковувати користувачів, щоб реагувати на порушення;

- як адміністратор, я хочу переглядати сесії конкретного користувача, щоб виявляти підозрілу активність;

- як адміністратор, я хочу переглядати логи дій користувачів, щоб проводити аудит;

- як адміністратор, я хочу моніторити статистику, щоб оцінювати навантаження та ефективність системи;

- як адміністратор, я хочу керувати правами доступу, щоб обмежити або розширити можливості користувачів;

- як адміністратор, я хочу керувати системними процесами, щоб підтримувати стабільність роботи платформи;

- як адміністратор, я хочу переглядати список інспекторів, щоб контролювати кадровий склад;

- як адміністратор, я хочу переглядати деталі конкретного інспектора, щоб отримувати повну інформацію;

- як адміністратор, я хочу створити нового інспектора, щоб додати працівника до системи;

- як адміністратор, я хочу редагувати інформацію про інспектора, щоб підтримувати актуальність даних.

Робота з протоколами:

- як адміністратор/інспектор, я хочу переглядати загальний список протоколів, щоб мати огляд ситуації;

- як адміністратор/інспектор, я хочу бачити протоколи, які створив конкретний інспектор, щоб оцінити його роботу;
- як інспектор, я хочу створити новий протокол порушення, щоб зафіксувати факт правопорушення;
- як інспектор, я хочу додавати фотофіксацію до протоколу, щоб забезпечити доказову базу;
- як інспектор/адміністратор, я хочу редагувати дані протоколу, щоб виправляти помилки;
- як користувач, я хочу переглядати деталі свого протоколу, щоб бути в курсі справи;
- як користувач, я хочу оплатити штраф за протоколом, щоб уникнути подальшого переслідування.

Відновлення доступу:

- як користувач, я хочу відновити пароль, щоб знову отримати доступ до системи;
- як користувач, я хочу змінити пароль після відновлення, щоб убезпечити свій обліковий запис.

Обробка помилок:

- як користувач, я хочу бачити сторінку 404 у разі некоректного запиту, щоб зрозуміти, що сторінки не існує.

Нижче подано зведений список основних і альтернативних сценаріїв використання системи, які відповідають user stories, описаним вище. Ці сценарії слугують основою для формулювання функціональних вимог і охоплюють ключові аспекти взаємодії різних типів користувачів з платформою.

Основні сценарії (функціональні вимоги):

- а) авторизація користувача;
- б) перегляд та редагування власного профілю;
- в) робота з адміністративною панеллю:

- 1) перегляд усіх зареєстрованих користувачів;

- 2) керування заблокованими користувачами;
- 3) перегляд сесій конкретного користувача;
- 4) перегляд логів дій користувачів;
- г) робота з працівниками паркувальної інспекції:
 - 1) перегляд списку інспекторів;
 - 2) детальна інформація про конкретного інспектора;
 - 3) створення нового запису інспектора;
 - 4) редагування існуючого запису інспектора;
- г) робота з протоколами правопорушень:
 - 1) перегляд загального списку протоколів;
 - 2) перегляд протоколів конкретного інспектора;
 - 3) створення нового протоколу;
 - 4) додавання фотофіксації до протоколу;
 - 5) редагування інформації протоколу;
 - 6) перегляд деталей конкретного протоколу;
 - 7) оплата штрафу;
- д) робота з адміністративною панеллю:
 - 1) моніторинг статистики;
 - 2) контроль прав доступу;
 - 3) управління системними процесами;
- є) відновлення паролю та зміна пароля після втрати доступу;
- ж) обробка некоректних запитів через сторінку 404.

Альтернативні сценарії:

- скасування створення протоколу;
- повторне додавання фотофіксації у випадку неуспішного завантаження;
- перевизначення інформації інспектора після виявлення помилки;
- повторна активація заблокованого користувача адміністратором;

– ручне завершення активної сесії користувача з підозрілою активністю (рис. 2.1).

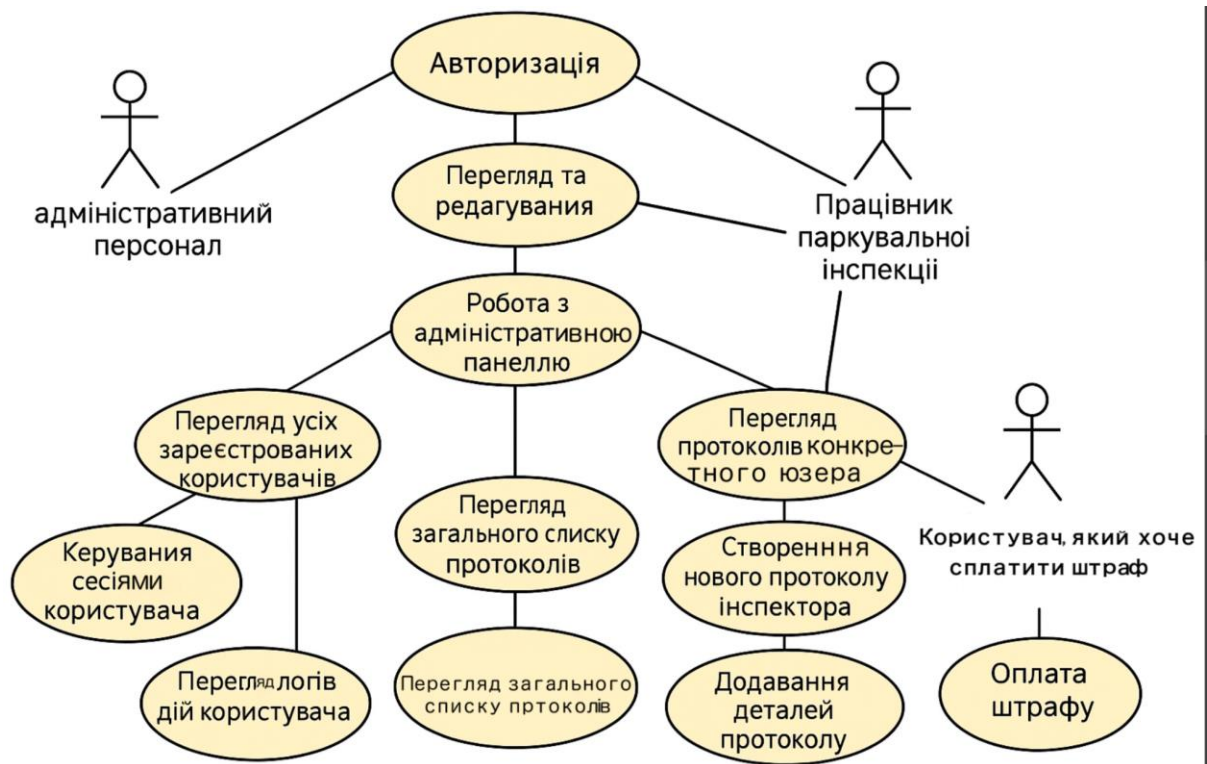


Рисунок 2.1 – Діаграма прецедентів програми

Завдяки широкому набору сценаріїв та модульності вебсистеми, забезпечено повноцінну цифрову підтримку ключових процесів паркувальної інспекції – від фіксації порушень до централізованого управління даними та доступом.

З метою забезпечення стабільної, безпечної та ефективної роботи вебзастосунку також сформульовано низку нефункціональних вимог, які враховуються на етапах розробки, тестування та впровадження системи:

- надійність: система повинна забезпечувати стабільну роботу без збоїв при одночасному доступі декількох користувачів;
- безпека: дані користувачів, протоколи та платіжна інформація повинні зберігатися у зашифрованому вигляді, доступ до системи має бути захищений аутентифікацією з різними рівнями прав доступу;

- продуктивність: сторінки застосунку мають завантажуватися не довше 2 секунд при стабільному інтернет-з'єднанні, запити до сервера повинні оброблятися за час не більше 1 секунди;
- масштабованість: система повинна бути готова до розширення бази даних та обробки збільшеної кількості користувачів і протоколів без втрати продуктивності;
- зручність у користуванні (юзабіліті): інтерфейс має бути інтуїтивно зрозумілим для користувачів з базовими комп'ютерними навичками. Інспектори повинні мати змогу створити протокол за кілька кліків;
- портативність: вебзастосунок має коректно працювати як на ПК, так і на мобільних пристроях (адаптивна верстка);
- підтримуваність: архітектура коду повинна бути модульною, щоб полегшити оновлення, виправлення помилок та додавання нового функціоналу;
- сумісність: застосунок має підтримуватись у всіх основних браузерах (Chrome, Firefox, Edge, Safari).

2.3 Алгоритм роботи програми

Алгоритм роботи програми забезпечує послідовне виконання дій, починаючи з авторизації користувача і завершуючи виходом із системи. Основна логіка програми передбачає взаємодію користувача з інтерфейсом через набір функціональних розділів, доступ до яких надається лише після успішного входу.

Послідовність роботи програми включає такі основні етапи:

- початок роботи: ініціалізується робота програми та виконується завантаження необхідних компонентів;
- авторизація користувача: користувач вводить облікові дані для входу в систему, програма перевіряє їх на коректність;

- перевірка успішності входу: у разі успішної авторизації користувач отримує доступ до основного функціоналу, а якщо авторизація неуспішна, користувача перенаправляє до модуля керування обліковими записами для повторної спроби;
- доступ до функціоналу: після входу користувач має змогу обрати один із доступних розділів системи;
- вибір розділу: користувач обирає потрібний функціональний розділ з-поміж наступних: Інспектори; Протоколи; Платежі; Користувачі;
- виконання дій у відповідному розділі: у вибраному модулі користувач може здійснювати відповідні операції згідно з наданими правами доступу;
- логування дій: усі дії користувача фіксуються в системі для забезпечення прозорості та можливості подальшого аудиту;
- завершення сесії: після завершення роботи з системою ініціюється завершення сесії користувача;
- вихід із програми: користувач виходить із системи, завершуючи роботу з програмним забезпеченням.

На рисунку нижче представлено блок-схему, яка детально ілюструє алгоритм роботи програми. Вона слугує візуальним доповненням до текстового опису, дозволяючи краще усвідомити логіку послідовного виконання дій у системі. Схема охоплює всі ключові етапи — від початкової ініціалізації та авторизації користувача до завершення сесії та виходу з програми.

Кожен блок на схемі відповідає окремому кроку або функціональному модулю, а стрілки вказують на можливі шляхи переходів між ними залежно від дій користувача або результатів перевірок. Такий підхід дозволяє не лише зрозуміти загальний принцип роботи, але й виявити логічні зв'язки між окремими компонентами програми. Візуалізація процесу особливо корисна під час розробки, тестування та навчання нових користувачів системи: юзерів, інспекторів, адміністраторів (рис. 2.2).

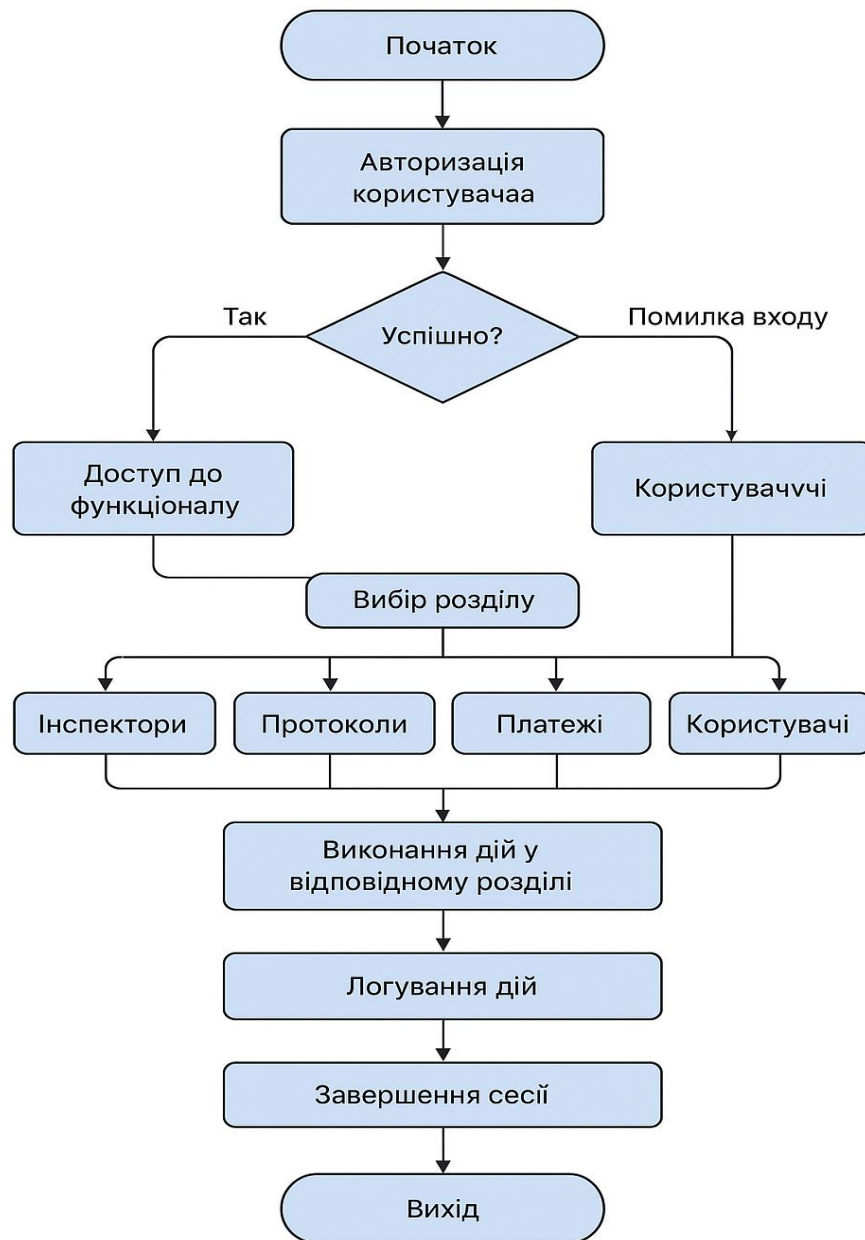


Рисунок 2.2 - Алгоритм роботи програми

2.4 Структурна схема модулів програми

Програмне забезпечення побудоване за модульним принципом, що забезпечує зручність у розробці, тестуванні, супроводі та масштабуванні системи. Кожен модуль виконує окрему функцію, взаємодіючи з іншими через чітко визначені інтерфейси.

Такий підхід дозволяє ізолювати окремі компоненти один від одного, що спрощує виявлення та усунення помилок. Крім того, це дає можливість

розширювати функціонал системи поступово, без ризику порушити вже реалізовану логіку роботи інших модулів.

Основні модулі системи:

- модуль авторизації: відповідає за автентифікацію користувача при вході в систему, забезпечує перевірку облікових даних, контроль доступу до функціоналу відповідно до ролі користувача;
- модуль інспекторів: реалізує функції перегляду, додавання, редагування та видалення інформації про інспекторів, дозволяє адміністраторам керувати записами інспекторів;
- модуль користувачів: здійснює управління обліковими записами користувачів, передбачає створення нових користувачів, зміну прав доступу, блокування/розблокування та інші дії;
- модуль логування: забезпечує запис усіх ключових дій користувачів у системі, це дозволяє здійснювати аудит, виявляти потенційні порушення та забезпечувати безпеку системи;
- модуль платежів: дозволяє здійснювати операції з платежами: перегляд історії платежів, додавання нових записів, звітність за платежами;
- модуль адміністратора: призначений для загального управління системою: контроль за модулями, налаштування системних параметрів, управління правами доступу, резервне копіювання тощо.

Додаткова перевага модульної архітектури полягає у можливості незалежного оновлення та вдосконалення окремих компонентів без необхідності змін у всій системі. Це значно спрощує процес впровадження нових функцій або виправлення помилок, адже зміни в одному модулі не впливають на роботу інших, за умови збереження узгодженості інтерфейсів.

Нижче подано рисунок, на якому представлено загальну структуру системи у вигляді діаграми, яка наочно демонструє взаємозв'язки між модулями. Така візуалізація дозволяє краще зрозуміти логіку побудови програмного забезпечення, а також полегшує аналіз архітектури під час розробки, технічної документації та подальшої експлуатації (рис. 2.3).

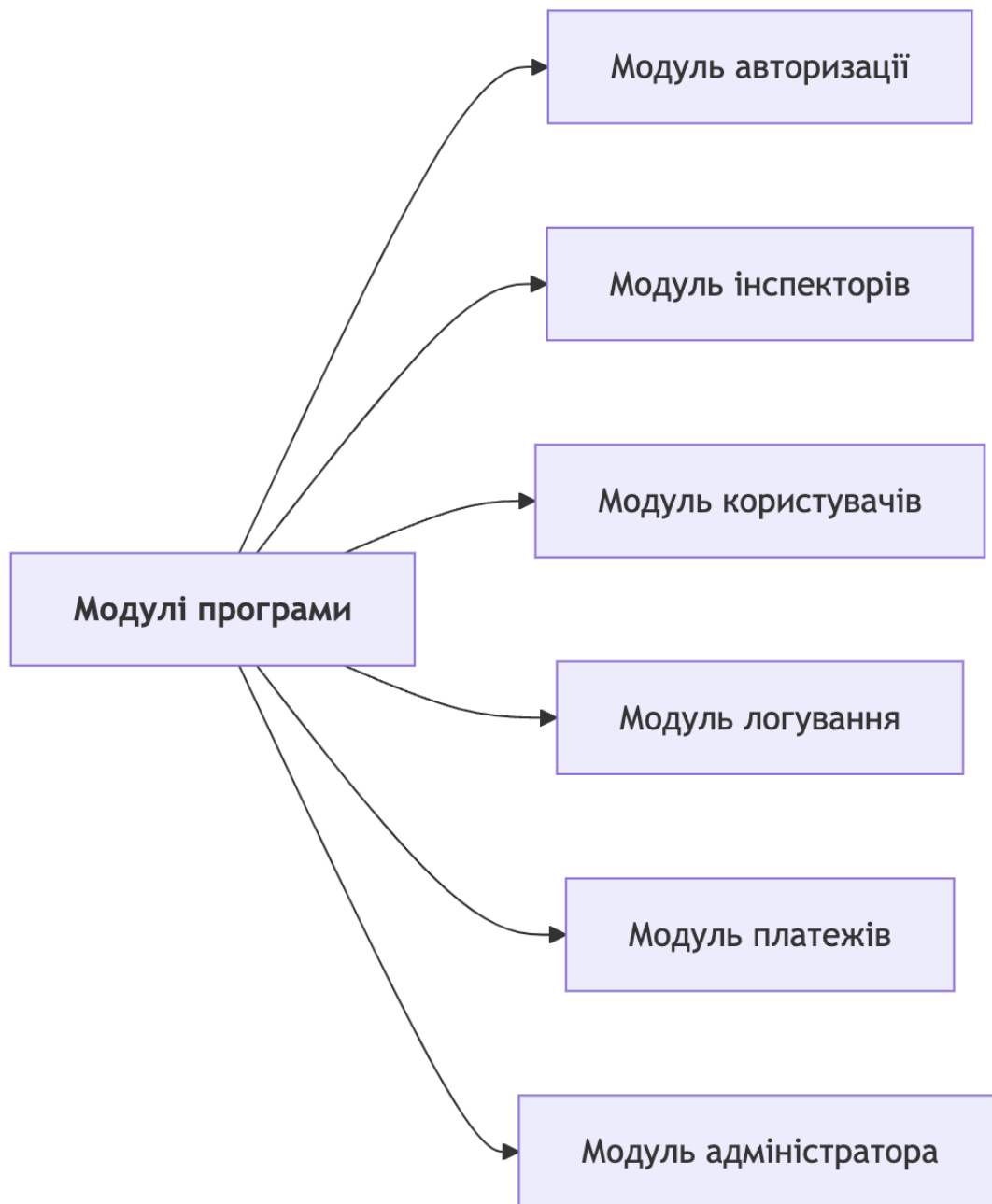


Рисунок 2.3 - Структурна схема модулів програми

2.5 Концептуальна схема програми

Концептуальна схема програми автоматизації процесів паркувальної інспекції відображає основні сутності системи та взаємозв'язки між ними. Основними об'єктами, які відображені на схемі, є: користувачі,

адміністратори, інспектори, протоколи, зображення, платежі, токени, журнали дій.

Користувач (User) є базовою сутністю системи. Він може реєструватися, входити до системи, ініціювати відновлення паролю, а також оплачувати протоколи. Деякі користувачі мають роль адміністратора (Admin), яка дозволяє виконувати дії з управління користувачами, логами, токенами та надання/позбавлення адміністративних прав.

Інспектор (ParkOfficer) є окремою сутністю, яка пов'язана з протоколами порушень. Кожен протокол (Protocol) створюється інспектором і містить інформацію про порушення правил паркування. До протоколу можуть прикріплюватися зображення (Image), що фіксують факт порушення.

Оплата за протокол виконується через сутність Payment, яка пов'язана з конкретним протоколом. Система перевіряє факт здійснення оплати за кожен протокол.

Схема описує ключові зв'язки між цими сутностями, такі як:

- один користувач може бути адміністраторами або бути звичайним юзером;
- один інспектор створює багато протоколів;
- кожен протокол може мати багато зображень;
- кожен протокол може бути оплачений одним платежем.

Концептуальна схема, дозволяє побачити логічну структуру системи в узагальненому вигляді, що є надзвичайно корисним під час проектування бази даних та розробки програмної логіки. Вона демонструє, як сутності пов'язані між собою, і дозволяє зрозуміти, які саме дані зберігаються в системі та яким чином вони взаємодіють.

Завдяки такій схемі можна ефективніше координувати свою роботу, виявляти потенційні логічні помилки ще на етапі проектування та забезпечувати узгодженість структури даних із функціональними вимогами системи. Це сприяє підвищенню якості кінцевого продукту та зменшенню витрат на його підтримку (рис. 2.4).

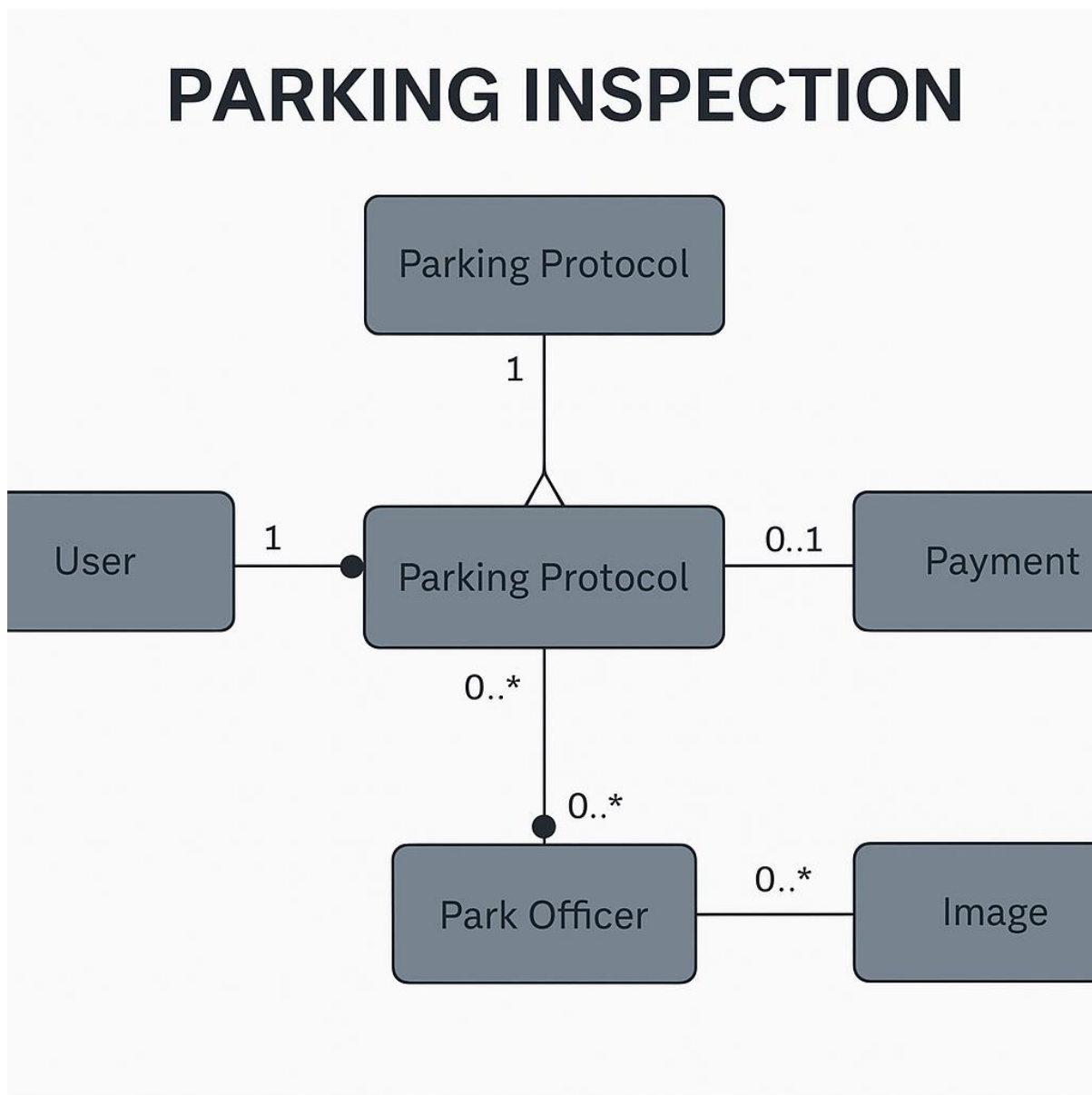


Рисунок 2.4 - Концептуальна схема програми

2.6 Структура бази даних

У процесі розробки вебзастосунку було використано дві бази даних: PostgreSQL та MongoDB. PostgreSQL відповідає за збереження основних доменних сутностей, пов'язаних із діяльністю паркувальної інспекції, таких як протоколи, працівники інспекції, платежі та фотофіксація. Такий вибір обумовлений надійністю, високою продуктивністю, широкими можливостями щодо реляційного моделювання даних, а також підтримкою ACID-властивостей.

Окрім реляційної бази даних PostgreSQL, для зберігання даних, пов'язаних з аутентифікацією, адмініструванням та журналюванням дій, було використано MongoDB. Ця база даних забезпечує гнучку схему зберігання документів, що особливо зручно для роботи з динамічними структурами користувачів, логів, токенів та різних адміністративних дій, які не мають чітко визначеної схеми або можуть змінюватися з часом.

Модель Image (PostgreSQL).

Таблиця images зберігає дані про фотофіксацію правопорушень. Має такі поля:

- path – шлях до зображення, збереженого на сервері; обов'язкове поле;
- protocolId – є зовнішнім ключем до таблиці protocols, забезпечуючи зв'язок типу багато-до-одного, згідно з яким одне або більше зображень належать до одного протоколу.

Модель ParkOfficer (PostgreSQL).

Таблиця park_officers зберігає інформацію про працівників паркувальної інспекції. Має такі поля:

- fullName – повне ім'я інспектора;
- badgeNumber – унікальний номер жетона інспектора;
- district – район, у якому закріплений інспектор;
- isWorked – логічне поле, що вказує, чи працює інспектор у паркувальній інспекції на даний момент;
- id – є зовнішнім ключем до таблиці protocols, що формує зв'язок типу один-до-багатьох, де один інспектор може мати багато протоколів.

Модель Payment (PostgreSQL).

Таблиця payments зберігає інформацію про оплату штрафів. Має такі поля:

- amount – сума штрафу;
- status – статус платежу (pending, completed, failed);
- userId – ідентифікатор користувача, який здійснював оплату.

– `protocolId` – виступає зовнішнім ключем до таблиці `protocols`, дозволяючи зв'язати конкретний платіж із відповідним протоколом.

Модель Protocol (PostgreSQL).

Таблиця `protocols` містить детальну інформацію про правопорушення.

Має такі поля:

- `serviceNotes` – службові примітки інспектора щодо ситуації;
- `fineAmount` – сума накладеного штрафу;
- `violatorFullName` – ПІБ правопорушника;
- `violatorPassportNumber` – паспортний номер правопорушника.

Кожен запис зв'язаний з одним працівником інспекції (через поле `officerId`) та може мати багато фотографій, завантажених як докази (через зв'язок з `images`). Таким чином реалізовано зв'язки типу багато-до-одного з таблицями `park_officers` та один-до-багатьох з `images`.

У результаті інтеграції двох різних типів баз даних було досягнуто балансу між структурованим зберіганням критично важливої інформації та гнучкістю у веденні змінних або слабо структурованих даних. Такий підхід дозволив забезпечити високу ефективність, масштабованість і зручність у подальшій підтримці вебзастосунку, враховуючи специфіку різних компонентів системи. PostgreSQL ефективно обробляє транзакційні запити та забезпечує цілісність даних завдяки реляційній природі, у той час як MongoDB дає змогу швидко впроваджувати нові функції, пов'язані з управлінням користувачами та веденням журналів, без необхідності складного рефакторингу схеми. Така архітектурна гнучкість підвищує адаптивність вебзастосунку до майбутніх змін і розширень. Схема реляційної БД зображена на рисунку нижче. Візуалізація допомагає краще зрозуміти логіку організації даних і спрощує навігацію під час подальшої розробки та обслуговування застосунку (рис. 2.5).

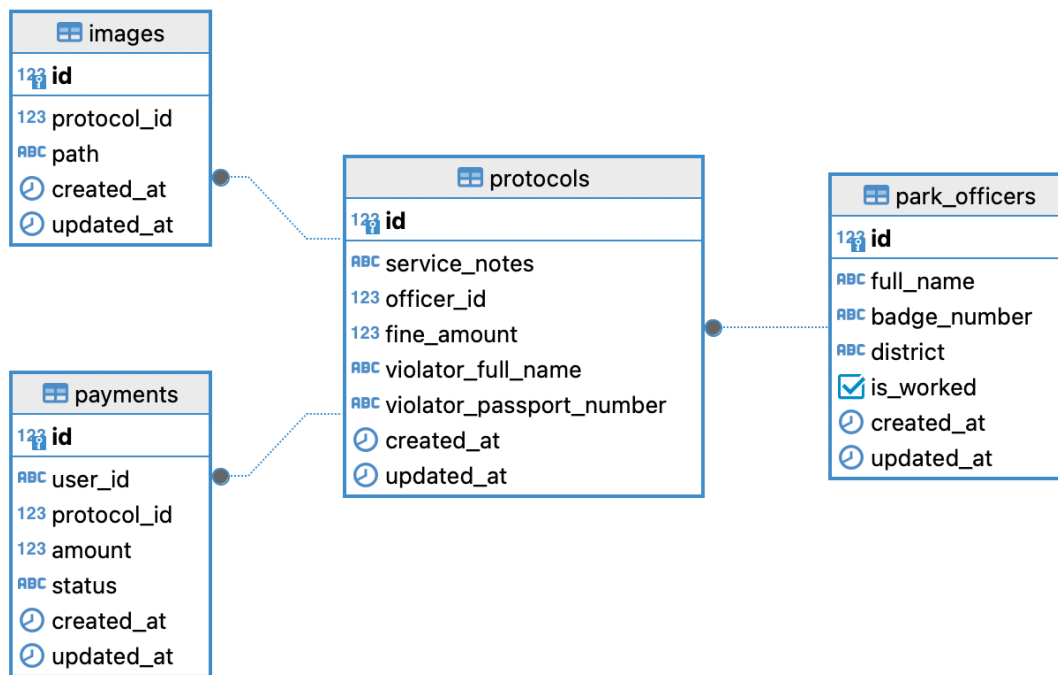


Рисунок 2.5 – Схема реляційної БД програми для автоматизації процесів паркувальної інспекції

Модель User (MongoDB).

Колекція users зберігає дані зареєстрованих користувачів системи.

Має такі поля:

- nickname – унікальний псевдонім користувача;
- email – унікальна електронна адреса;
- passwordHash – хеш пароля (відсутній, якщо використовується Google-авторизація);
- role – роль користувача в системі (наприклад, admin, user);
- card – об'єкт, що зберігає платіжні дані користувача (номер картки, термін дії, CVC).

Ця модель є базовою для автентифікації та контролю доступу користувачів.

Модель RefreshToken (MongoDB).

Колекція refresh_tokens зберігає інформацію про видані токени оновлення. Має такі поля:

- userId – посилання на користувача, якому належить токен;
- token – сам токен оновлення;
- createdAt – дата створення токена;
- geolocation, ipAddress, operatingSystem, browser – допоміжна інформація для безпеки (геолокація, IP, ОС, браузер);
- isRevoked – логічне поле, яке показує, чи токен було відкликано.

Цей функціонал дозволяє реалізувати безпечну авторизацію за допомогою JWT.

Модель LogActionType (MongoDB).

Колекція log_action_types містить довідкову інформацію про типи дій, які можуть бути зафіксовані у журналі (наприклад, LOGIN, DELETE_USER, EDIT_ROLE). Має єдине поле:

- type – тип дії, фіксується як рядок, що належить до переліку дозволених значень.

Ця модель слугує джерелом для фільтрації та уніфікації типів логів у системі.

Модель Log (MongoDB).

Колекція logs зберігає всі адміністративні дії, виконані в системі. Має такі поля:

- description – опис дії;
- actionType – тип дії, що відповідає одному зі значень з log_action_types;
- performedBy – користувач, що виконав дію;
- timestamp – час виконання дії.

Модель логування дозволяє вести повноцінний аудит взаємодії з системою.

Модель Banlist (MongoDB).

Колекція banlist зберігає інформацію про заблокованих користувачів.

Має такі поля:

- userId – користувач, який був заблокований;
- adminId – адміністратор, який здійснив блокування;
- reason – причина блокування;
- createdAt – дата блокування.

2.6 Визначення порядку роботи програми

Функціональна схема програми автоматизації процесів паркувальної інспекції базується на взаємодії користувача з меню, яке передбачає логічно структуровані етапи роботи — від перегляду загального рейтингу порушень до конкретних дій з перегляду протоколів, взаємодії з інспекторами, аналітики та історії платежів. Схема підтримує зворотний перехід на кожному кроці, що дозволяє користувачу ефективно перемикатися між функціональними модулями системи.

Особливу увагу в схемі приділено інтуїтивній навігації та логічному групуванню функцій, що дозволяє зменшити час на навчання користувачів та знижує ймовірність помилок при роботі з системою. Кожен функціональний модуль має чітко визначене призначення, а структура інтерфейсу сприяє послідовному виконанню задач, зберігаючи загальну узгодженість і зручність користування.

Функціональна схема, представлена на рисунку нижче, виступає основою для побудови логіки інтерфейсу користувача та відображає ключові сценарії використання системи. Вона дозволяє розробникам та аналітикам легко відстежувати послідовність взаємодій і виявляти потенційні вузькі місця або надлишкові переходи. Завдяки такій структурованій організації програми забезпечується не лише зручність для кінцевого користувача, а й спрощується подальше масштабування системи (рис. 2.6).

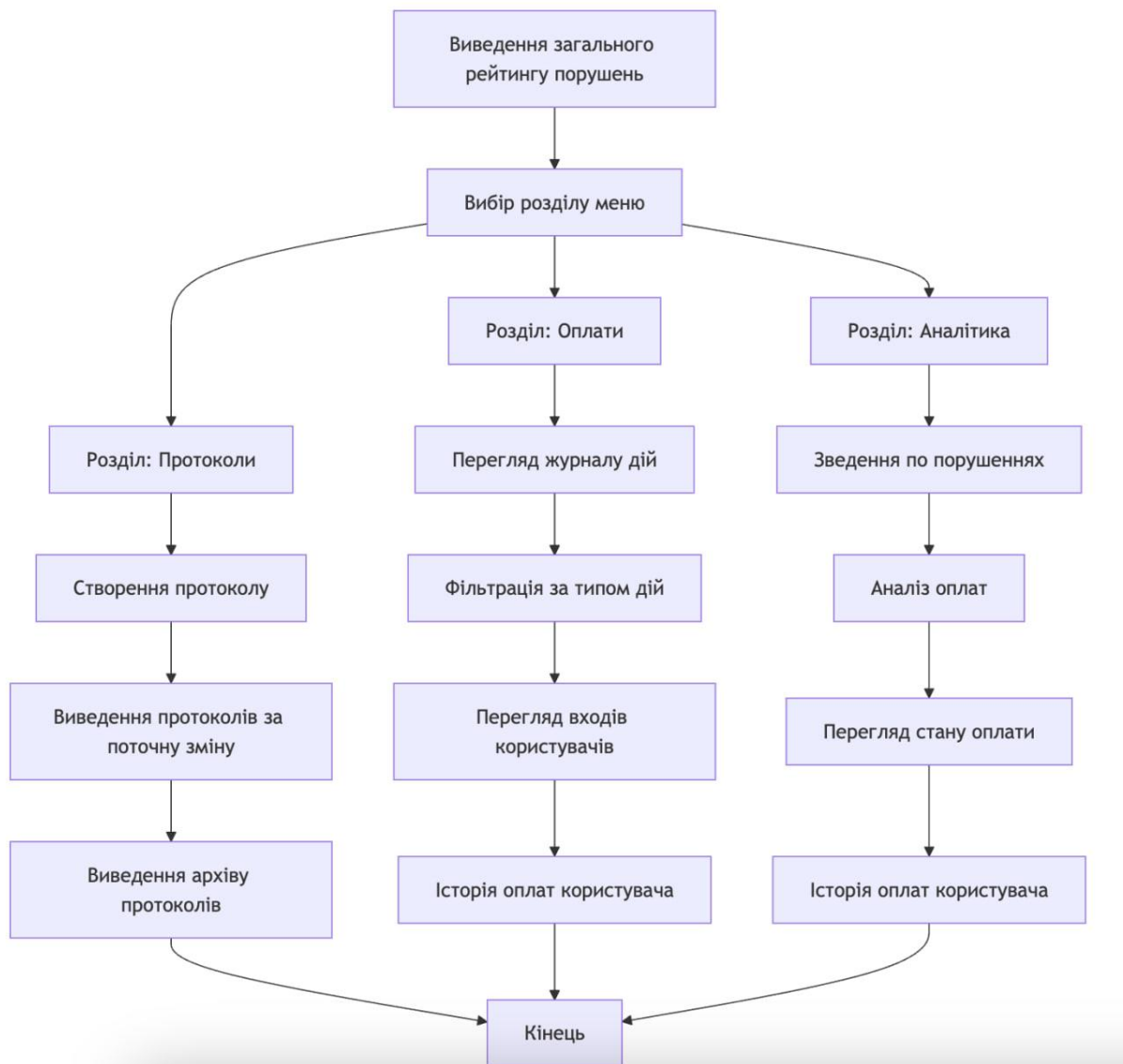


Рисунок 2.6 – Функціональна схема програми для автоматизації процесів паркувальної інспекції

2.7 Висновки за розділом 2

Обґрунтовано прийняття рішення, пов'язаного з вибором ВФ React в якості основи розробки, описано сценарії використання програми для автоматизації процесів паркувальної інспекції з визначенням діаграми прецедентів, представлено структуру баз даних. Також було розроблено алгоритм роботи програми, структурну схему модулів програми а також концептуальну схему. Крім того, було визначено порядок роботи програми.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Реалізація клієнтської частини (Frontend)

Клієнтська частина вебзастосунку для автоматизації процесів паркувальної інспекції реалізована з використанням бібліотеки React. Такий вибір було обґрунтовано потребою у створенні динамічного, інтерактивного інтерфейсу, що забезпечує швидку взаємодію користувача з даними без необхідності постійного перезавантаження сторінки.

Одразу після запуску застосунку користувач потрапляє на сторінку входу. Для автентифікації потрібно ввести email (який є логіном) та пароль. Пароль не зберігається у відкритому вигляді, а хешується на сервері за допомогою алгоритму bcrypt із засалюванням у 5 раундів.

Лістинг 3.1 – Фрагмент коду форми логіну

```
function LoginForm() {  
  const [email, setEmail] = useState("");  
  const [password, setPassword] = useState("");  
  
  const handleLogin = async () => {  
    try {  
      const response = await axios.post('/api/sign-in', { email, password });  
      localStorage.setItem('accessToken', response.data.accessToken);  
      navigate('/inspectors');  
    } catch (error) {  
      alert('Невірні облікові дані');  
    }  
  };  
  
  return (  
    <form onSubmit={handleLogin}>
```

```

    <input type="email" value={email} onChange={e => setEmail(e.target.value)}
  />
    <input type="password" value={password} onChange={e =>
setPassword(e.target.value)} />
    <button type="submit">Увійти</button>
  </form>
);
}

```

У разі втрати пароля користувач може перейти до сторінки його відновлення. Введений email надсилається на сервер, після чого користувачу приходить лист з посиланням на форму встановлення нового пароля.

Користувач також має можливість зареєструватись — або заповнивши форму, або через інтеграцію з Google OAuth.

Після входу або реєстрації користувач потрапляє на сторінку перегляду паркувальних інспекторів. Тут реалізовано систему фільтрації та пошуку за такими параметрами: ім'я інспектора, номер жетону, район, статус роботи (активний або звільнений).

Лістинг 3.2 – Фрагмент коду фільтру інспекторів

```

const filteredInspectors = inspectors.filter(inspection =>
  inspection.fullName.includes(searchTerm) ||
  inspection.badgeNumber.includes(searchTerm) ||
  inspection.district.includes(searchTerm)
);

```

На сторінці конкретного інспектора доступна функція генерації PDF-документу з його інформацією (з використанням бібліотеки jsPDF) та перегляду протоколів, складених ним.

Таким чином, реалізовано основні сценарії взаємодії користувача з клієнтською частиною вебзастосунку: автентифікація, перегляд даних, фільтрація, генерація звітів.

Для зручності навігації та розуміння структури клієнтської частини вебзастосунку доцільно представити перелік основних сторінок інтерфейсу користувача разом із коротким описом їх призначення. Це дозволяє чітко окреслити функціональні можливості, що доступні на кожному етапі взаємодії користувача із системою.

Таблиця 3.1 – Таблиця відповідності сторінок інтерфейсу користувача (фронтенду) та їх призначення

Назва сторінки	Призначення сторінки
1	2
AdminPanelPage	Головна сторінка адміністратора з доступом до керування користувачами та даними
AllUsersPage	Відображення повного списку зареєстрованих користувачів системи
BannedUsersPage	Перегляд та керування заблокованими користувачами
ForgotPassword	Форма для надсилання запиту на скидання пароля за адресою електронної пошти
HomePage	Домашня сторінка застосунку; початкова точка входу для всіх користувачів
MyProfile	Сторінка перегляду та редагування особистої інформації користувача
NotFoundPage	Сторінка, що відображається при переході на неіснуючий маршрут (помилка 404)

Продовження таблиці 3.1

1	2
ParkOfficerDetailsPage	Детальний перегляд інформації про конкретного працівника паркувальної інспекції
ParkOfficersPage	Перегляд списку працівників паркувальної інспекції
ProtocolDetailsPage	Перегляд повної інформації щодо конкретного протоколу порушення
ProtocolsPage	Список усіх протоколів з можливістю фільтрації, пошуку та сортування
ResetPassword	Сторінка для відновлення паролю у разі втрати
UserLogsPage	Перегляд логів активності конкретного користувача
UserSessionsPage	Перегляд активних сесій користувача з можливістю їх завершення

3.2 Реалізація бекенд-частини застосунку

Бекенд-частина вебзастосунку реалізована з використанням фреймворку Express.js. Вона відповідає за обробку HTTP-запитів, керування бізнес-логікою, взаємодію з базою даних, перевірку автентичності та авторизації користувачів, а також логування подій.

Основні модулі та функціональність.

Користувачі та автентифікація.

Використано модуль User. Користувачі проходять реєстрацію, вхід, відновлення пароля та перевірку токенів.

Лістинг 3.3 – Фрагмент коду модулю реєстрації

```
async function registrationUser(req, res) {
```

```

const { email, password } = req.body;
const hashedPassword = await bcrypt.hash(password, 5);
const user = await User.create({ email, password: hashedPassword });
res.status(201).json(user);
}

```

Пароль хешується за допомогою bcrypt із солінням у 5 раундів для безпечного зберігання. Після реєстрації створюється новий запис у колекції користувачів.

Авторизація та токени.

Для перевірки автентичності використовуються JWT-токени.

Лістинг 3.4 – Фрагмент коду функції перевірки токенів

```

function checkToken(req, res, next) {
  const token = req.headers.authorization?.split(" ")[1];
  const payload = jwt.verify(token, process.env.ACCESS_SECRET);
  req.user = payload;
  next();
}

```

Функція middleware перевіряє дійсність токена доступу. Якщо токен коректний, до req.user додається payload користувача.

Лістинг 3.5 – Фрагмент коду функції створення інспектора

```

async function createParkOfficer(req, res) {
  const { fullName, badgeNumber, district } = req.body;
  const officer = await ParkOfficer.create({ fullName, badgeNumber, district });
  res.status(201).json(officer);
}

```

Адміністратор може додавати інспекторів. Перевірки у middleware гарантують, що ці дії доступні лише авторизованим адміністраторам.

Протоколи порушень.

Лістинг 3.6 – Фрагмент коду функції створення протоколу

```
async function createProtocol(req, res) {  
  const { officerId, violationData } = req.body;  
  const protocol = await Protocol.create({ officerId, violationData });  
  res.status(201).json(protocol);  
}
```

Збереження інформації про порушення, що пов'язані з конкретним інспектором. Протокол може містити зображення та дані про оплату.

Ведеться запис подій у лог.

Лістинг 3.7 – Фрагмент коду функції логування

```
async function logAction(userId, actionType) {  
  await Log.create({ userId, actionType, timestamp: new Date() });  
}
```

Це дозволяє адміністраторам бачити історію дій користувачів у системі та забезпечити аудит.

Усі контролери організовані модульно. Для зберігання даних використовується база MongoDB, для логування — окрема колекція Logs.

У таблиці нижче наведено перелік основних методів контролерів разом із їх функціональним призначенням.

Це дозволяє підтримувати чисту архітектуру та спрощує обслуговування окремих частин системи. Такий підхід також сприяє гнучкості у розширенні функціоналу без необхідності значних змін у наявному коді.

Таблиця 3.2 – Таблиця відповідності методів на бекенд-частині

Назва методу контролера	Призначення методу
1	2
loginUser	Обробка входу користувача до системи
registerUser	Реєстрація нового користувача
logoutUser	Вихід користувача із системи та завершення сесії
getCurrentUser	Отримання інформації про поточного користувача
banUser	Блокування користувача адміністратором
unbanUser	Розблокування користувача
getAllUsers	Отримання списку всіх користувачів
getBannedUsers	Отримання списку заблокованих користувачів
getUserLogs	Отримання логів дій конкретного користувача
getUserSessions	Отримання всіх сесій конкретного користувача
endUserSession	Завершення окремої сесії користувача
createProtocol	Створення нового протоколу порушення
getAllProtocols	Отримання списку всіх протоколів
getAllProtocolsByOfficerID	Отримання всіх протоколів, створених певним працівником паркувальної інспекції
getImageByID	Отримання зображення за його ідентифікатором

Продовження таблиці 3.2

1	2
getProtocolByID	Отримання повної інформації про конкретний протокол
uploadImage	Завантаження зображення, пов'язаного з протоколом
checkIfProtocolIsPaid	Перевірка статусу оплати протоколу
createPayment	Створення нового платежу за протоколом
getPaymentsByUserID	Отримання всіх платежів, здійснених певним користувачем
getParkOfficers	Отримання списку всіх працівників паркувальної інспекції
getParkOfficerByID	Отримання інформації про конкретного інспектора

Результатом реалізації є надійний бекенд, який забезпечує:

- захист і перевірку користувачів;
- обробку CRUD-операцій для інспекторів і протоколів;
- можливість керування платіжною інформацією;
- централізоване логування для адміністраторів.

3.3 Реалізація повного циклу функціоналу (Full Stack)

Повний цикл реалізації функціоналу передбачає узгоджену роботу клієнтської (frontend) та серверної (backend) частин програми, а також їхню взаємодію через API. Один із прикладів такого повного циклу — створення, перегляд і оплата протоколу порушення.

Створення протоколу порушення.

На стороні інспектора у вебінтерфейсі доступна форма створення нового протоколу.

Лістинг 3.8 – Фрагмент коду Реакт—компонента на фронтенді

```
const handleSubmit = async () => {  
  const formData = new FormData();  
  formData.append("officerId", selectedOfficerId);  
  formData.append("violationData", JSON.stringify(violationDetails));  
  images.forEach((img) => formData.append("images", img));  
  
  await axios.post("/api/protocols", formData, {  
    headers: { Authorization: `Bearer ${token}` }  
  });  
};
```

За допомогою FormData здійснюється надсилання як текстової інформації, так і зображень порушення.

Лістинг 3.9 – Фрагмент коду обробки на бекенді (Express.js)

```
app.post("/api/protocols", uploadImages, async (req, res) => {  
  const { officerId, violationData } = req.body;  
  const images = req.files.map(f => f.filename);  
  const protocol = await Protocol.create({ officerId, violationData, images });  
  res.status(201).json(protocol);  
});
```

Сервер приймає дані з фронтенду, зберігає їх у базі, а зображення — на файлову систему або в хмарне сховище.

Далі буде розглянуто, як відбувається перегляд протоколу.

Лістинг 3.10 – Фрагмент коду обробки перегляду протоколів на фронті

```
useEffect(() => {
  axios.get(`/api/protocols/${id}`).then(res => setProtocol(res.data));
}, []);
```

Лістинг 3.11 – Фрагмент коду перегляду протоколів на бекенді

```
app.get("/api/protocols/:id", async (req, res) => {
  const protocol = await Protocol.findById(req.params.id);
  res.json(protocol);
});
```

Завантажується повна інформація про конкретне порушення, зокрема прив'язка до інспектора та додані фото.

Далі буде розглянуто, як відбувається оплата протоколу

Лістинг 3.12 – Фрагмент коду обробника кнопки оплати на фронті

```
const handlePayment = async () => {
  await axios.post(`/api/protocols/${protocolId}/pay`, { cardId });
};
```

Лістинг 3.13 – Фрагмент коду обробника оплати на бекенді

```
app.post("/api/protocols/:protocolId/pay", async (req, res) => {
  const { protocolId } = req.params;
  const { cardId } = req.body;
  await Payment.create({ protocolId, cardId, paidAt: new Date() });
  await Protocol.findByIdAndUpdate(protocolId, { status: "paid" });
  res.status(200).send("Оплата пройшла успішно");
});
```

Бекенд створює запис про оплату, оновлює статус протоколу, а фронтенд відображає відповідне повідомлення.

Таким чином, повний стек реалізації забезпечує:

- ефективну взаємодію між користувачем і сервером;
- автоматичну обробку форм, валідацію, завантаження файлів і оплату;
- зручний UX через React-інтерфейс;
- надійне збереження даних і логіку опрацювання на Express-бекенді.

3.4 Розгортання застосунку (деплой)

Для забезпечення постійного доступу до вебзастосунку автоматизації процесів паркувальної інспекції було виконано розгортання (деплой) фронтенд- та бекенд-частин застосунку на віддаленому сервері.

Технологічний стек:

- серверна частина (backend): Node.js та Express;
- клієнтська частина (frontend): React;
- база даних: MongoDB, PostgreSQL;
- сервер: Ubuntu 22.04.

Сервер додатку запускається через: pm2.

Фронтенд-білд обслуговується через: nginx.

Розгортання в Docker (опційно): передбачено можливість запуску контейнерів з Node.js, MongoDB, nginx.

Далі будуть розглянуті кроки розгортання.

Спочатку виконується білд фронтенду.

Результат з build/ копіюється в каталог, який обслуговується nginx.

Далі відбувається запуск бекенду з pm2.

Далі налаштовується nginx-конфігурація.

При розгортанні не обов'язково дотримуватись саме такої послідовності кроків розгортання.

Лістинг 3.14 – Фрагмент коду nginx-конфігурації

```
server {  
    listen 80;  
    server_name parking-inspection.com;  
  
    root /var/www/parking-app/build;  
    index index.html;  
  
    location /api/ {  
        proxy_pass http://localhost:5000/;  
        proxy_http_version 1.1;  
        proxy_set_header Upgrade $http_upgrade;  
        proxy_set_header Connection 'upgrade';  
        proxy_set_header Host $host;  
        proxy_cache_bypass $http_upgrade;  
    }  
  
    location / {  
        try_files $uri /index.html;  
    }  
}
```

Далі підіймаються бази даних: PostgreSQL та MongoDB.

У вебзастосунку використовуються дві різні бази даних, кожна з яких виконує окрему роль у загальній архітектурі:

PostgreSQL — основна реляційна база даних для зберігання структурованої інформації (користувачі, протоколи, платежі тощо).

MongoDB — документно-орієнтоване сховище для зберігання додаткових, менш структурованих даних, зокрема метаданих зображень, логів та сесій користувачів.

PostgreSQL використовується як основне джерело даних для критичних бізнес-об'єктів. Вона забезпечує цілісність даних, підтримку транзакцій і складні запити.

База запускається як окремий контейнер у Docker-композиції або може бути підключена ззовні.

Параметри підключення зберігаються у .env файлі.

Лістинг 3.15 – Фрагмент .env файлу

```
PG_HOST=localhost  
PG_PORT=5432  
PG_DATABASE=parking_db  
PG_USER=admin  
PG_PASSWORD=admin123
```

Підключення до бази даних виконується через бібліотеку pg.

Лістинг 3.16 – Фрагмент коду підключення до бази даних

```
import { Pool } from 'pg';  
import dotenv from 'dotenv';  
  
dotenv.config();  
  
const pool = new Pool({  
  host: process.env.PG_HOST,  
  port: process.env.PG_PORT,  
  database: process.env.PG_DATABASE,  
  user: process.env.PG_USER,  
  password: process.env.PG_PASSWORD,  
});
```

export default pool;

Усі основні контролери (протоколи, користувачі, платежі) працюють саме з PostgreSQL.

MongoDB використовується як допоміжна база даних для зберігання неструктурованих або об'ємних даних, які не потребують жорсткої схеми.

Переважно застосовується для таких сутностей:

- зображення, пов'язані з протоколами (наприклад, фото порушень);
- логи користувачів, що фіксують важливі події;
- сесії користувачів, якщо використовується механізм авторизації зберігання в БД.

Підключення реалізовано через mongoose.

Лістинг 3.17 – Фрагмент коду підключення до MongoDB

```
import mongoose from 'mongoose';
import dotenv from 'dotenv';

dotenv.config();

const connectMongo = async () => {
  try {
    await mongoose.connect(process.env.MONGODB_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.error('MongoDB connection error:', err);
    process.exit(1);
  }
}
```

};

export default connectMongo;

Опціонально: Docker-інфраструктура.

За допомогою docker-compose можливо підняти всі сервіси в контейнерах:

- backend;
- frontend;
- mongo;
- postgresql;
- nginx.

У результаті деплою користувачі отримують доступ до застосунку за доменним іменем, можуть авторизуватись, працювати з протоколами, переглядати інформацію в реальному часі, а адміністратори — керувати інспекторами та журналами дій.

3.5 Висновки за розділом 3

У даному розділі було представлено функціональну схему вебзастосунку для автоматизації процесів паркувальної інспекції, що відображає основні компоненти системи та зв'язки між ними. Реалізовано всі ключові функції, необхідні для повноцінної роботи застосунку: реєстрація та автентифікація користувачів, керування обліковими записами, створення та обробка протоколів порушень, прикріплення фотофіксацій, оплата штрафів, а також адміністрування системи.

Описано загальну архітектуру системи, включаючи функціональну схему, структурну модель модулів, концептуальну схему бази даних та алгоритм роботи програми. Розглянуто реалізацію серверної та клієнтської

частин, а також взаємодію між ними. Здійснено повний цикл впровадження — від розробки до розгортання застосунку в робоче середовище.

Описані програмні одиниці демонструють чітку структурування функціоналу відповідно до ролей користувачів (інспектори, адміністратори, звичайні користувачі) та розподіл відповідальності між окремими модулями. Кожен програмний компонент реалізує окрему бізнес-логіку відповідно до загального призначення застосунку, забезпечуючи надійність, масштабованість та безпеку системи.

Таким чином, на основі розробленої функціональної структури вдалося реалізувати програмні засоби та одиниці.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для автоматизації процесів паркувальної інспекції. Основною метою є спрощення роботи інспекторів з фіксації порушень правил паркування, обробки відповідної інформації, взаємодії з адміністративною частиною системи та забезпечення зручного механізму для користувачів щодо перегляду і оплати штрафів.

Застосунок забезпечує створення та керування протоколами порушень, прикріплення фотофіксацій до кожного випадку, перевірку статусу оплат, роботу з платіжними картками, а також надає адміністраторам можливості контролю за діями користувачів, інспекторів і власне процесом винесення штрафів. Завдяки розподілу ролей, кожна група користувачів взаємодіє лише з релевантним для неї функціоналом.

Система також дозволяє ефективно відслідковувати історію дій користувачів, адмініструвати доступи, блокувати порушників і забезпечує прозору, надійну та зручну взаємодію між усіма учасниками процесу паркувального контролю.

4.2 Умови виконання програми

Обов'язковою умовою експлуатації програми для автоматизації процесів паркувальної інспекції є наявність адміністратора системи або відповідального користувача, який має повноваження на адміністрування вебзастосунку.

Мінімальні вимоги до технічного забезпечення сервера для ефективного функціонування системи включають:

- процесор Intel Xeon E5440 (частота 2,83 ГГц) або еквівалентний;
- жорсткий диск обсягом не менше 100 Гб;
- оперативна пам'ять обсягом не менше 4 Гб.

Крім того, для роботи вебзастосунку необхідне встановлене серверне програмне забезпечення, зокрема середовище Node.js, система керування базами даних (MongoDB, PostgreSQL).

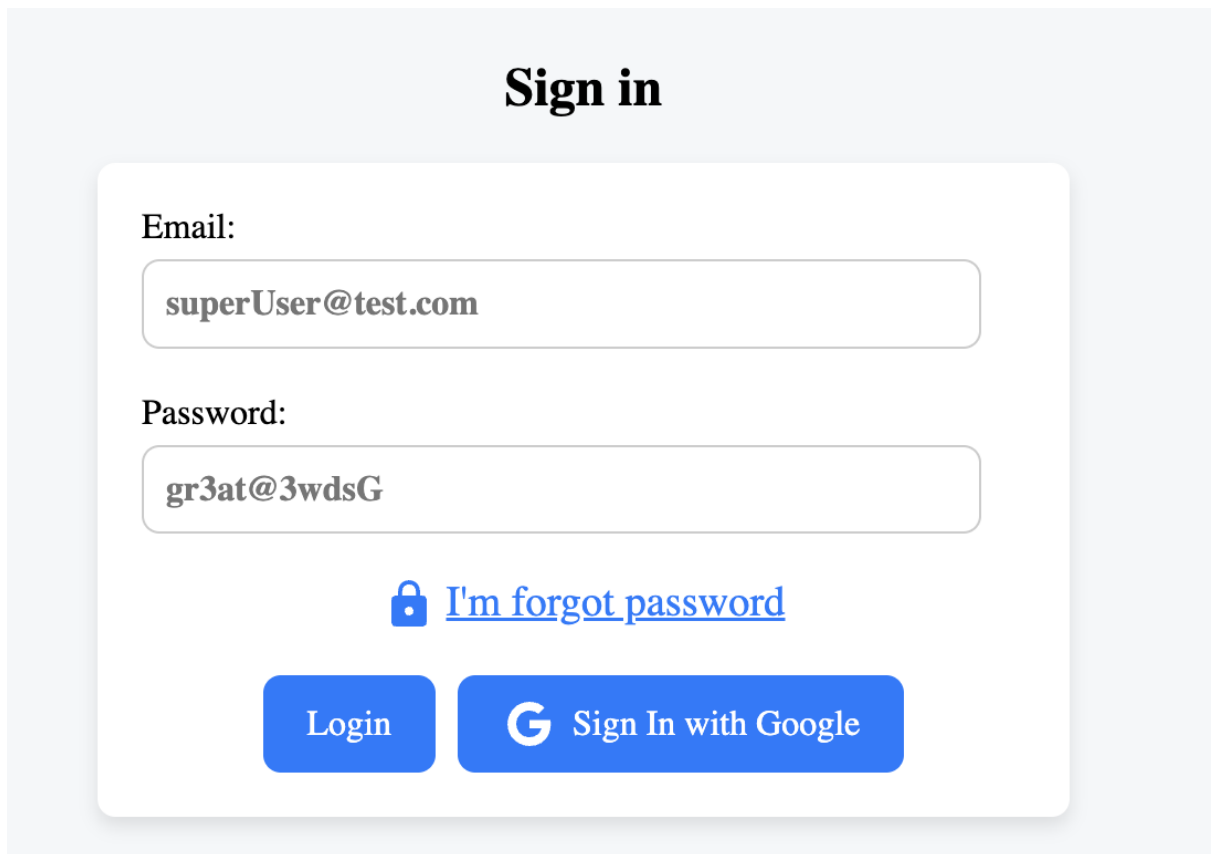
Також є можливість підняти додаток на Docker, що значно спрощує процес розгортання та забезпечує ізоляцію середовища виконання. Завдяки контейнеризації усі необхідні залежності, конфігурації та сервіси можуть бути об'єднані в одному середовищі, що гарантує стабільну роботу застосунку на різних хост-системах.

Docker дозволяє автоматизувати процес налаштування середовища за допомогою Dockerfile та docker-compose, що є особливо корисним під час розгортання на віддалених серверах або в хмарних середовищах. Крім того, контейнеризований підхід забезпечує масштабованість та полегшує оновлення компонентів без впливу на загальну працездатність системи.

4.3 Виконання і тестування програми

Реалізацію тестування забезпечено в межах перевірки набору функціональних можливостей, які повинні були бути забезпеченими для роботи користувача. Тестування продемонструвало успішні результати забезпечення всіх запланованих функцій. Далі на основі результатів тестування описано приклади і логіку роботи з програмою.

Після запуску додатку користувач бачить форму входу, яка передбачає авторизацію за допомогою електронної пошти (email) та пароля. Пароль не зберігається у відкритому вигляді — він хешується за допомогою алгоритму bcrypt із засалюванням у 5 раундів, що забезпечує безпечне зберігання облікових даних у базі даних (рис. 4.1).



The image shows a 'Sign in' form on a light blue background. The form is white with rounded corners and contains the following elements:

- Sign in** (Section Header)
- Email:** (Label) followed by a text input field containing `superUser@test.com`.
- Password:** (Label) followed by a text input field containing `gr3at@3wdsG`.
- [I'm forgot password](#) (Link with a lock icon).
- Login** (Blue button).
- Sign In with Google** (Blue button with the Google 'G' logo).

Рисунок 4.1 – Форма входу у додаток

Крім того, додаток передбачає можливість відновлення паролю. Відновлення паролю передбачає надсилання спеціального посилання для відновлення паролю на зареєстрований користувачем email. Перейшовши за ним, користувач потрапляє на сторінку, де може встановити новий пароль.

Також користувач має можливість перейти у режим реєстрації. Реєстрація може здійснюватися шляхом заповнення відповідної форми з введенням електронної пошти та паролю або за допомогою облікового запису Google через OAuth-автентифікацію.

Після успішної реєстрації дані нового користувача зберігаються в базі, і йому автоматично надається роль, що відповідає типу стандартного користувача. У разі використання реєстрації через Google, система самостійно отримує необхідну інформацію з облікового запису, що значно спрощує процес входу для користувача та підвищує зручність взаємодії із застосунком (рис. 4.2).

Sign up

Nickname:

Email:

Password:

Registration

 Sign Up with Google

Рисунок 4.2 – Форма реєстрації

Після успішного входу до системи або завершення процедури реєстрації користувач автоматично перенаправляється на сторінку, що містить інформацію про паркувальних інспекторів. На цій сторінці представлено повний перелік усіх інспекторів, які є або були залучені до роботи в паркувальній інспекції. Інтерфейс сторінки розроблений таким чином, щоб забезпечити зручну навігацію та швидкий доступ до потрібної інформації: реалізована функція фільтрації, яка дозволяє відокремити активних інспекторів від тих, що вже не перебувають на службі. Окрім цього, система підтримує зручний пошук за кількома критеріями — користувач може знайти необхідного інспектора, ввівши його повне ім'я, унікальний номер жетона або район, у якому працівник виконував свої обов'язки. Це

значно спрощує процес роботи з великими масивами даних і дозволяє оперативно отримувати точну інформацію (рис. 4.3).

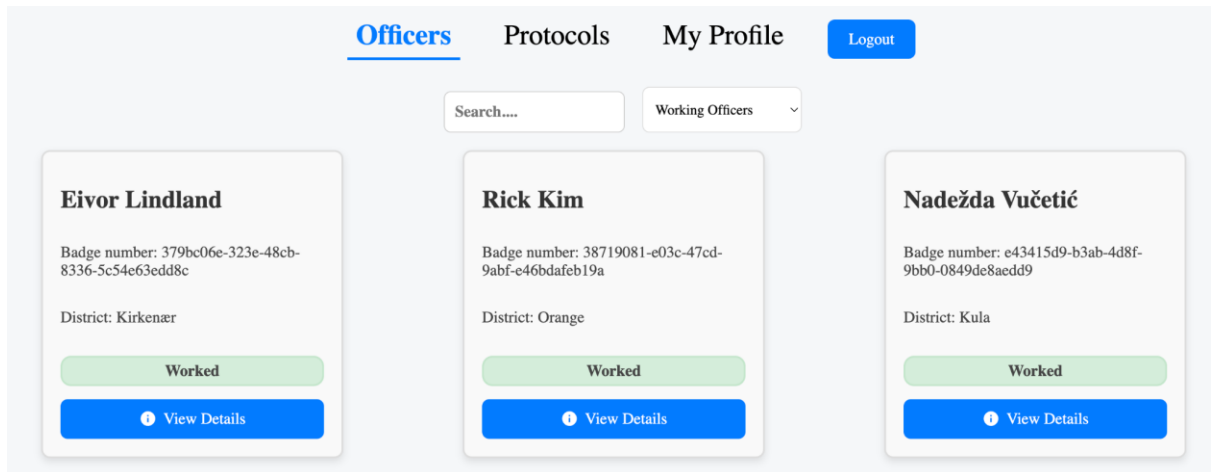


Рисунок 4.3 – Сторінка інспекторів

Натиснення на кнопку "View details" відкриває сторінку окремого інспектора (рис. 4.4).

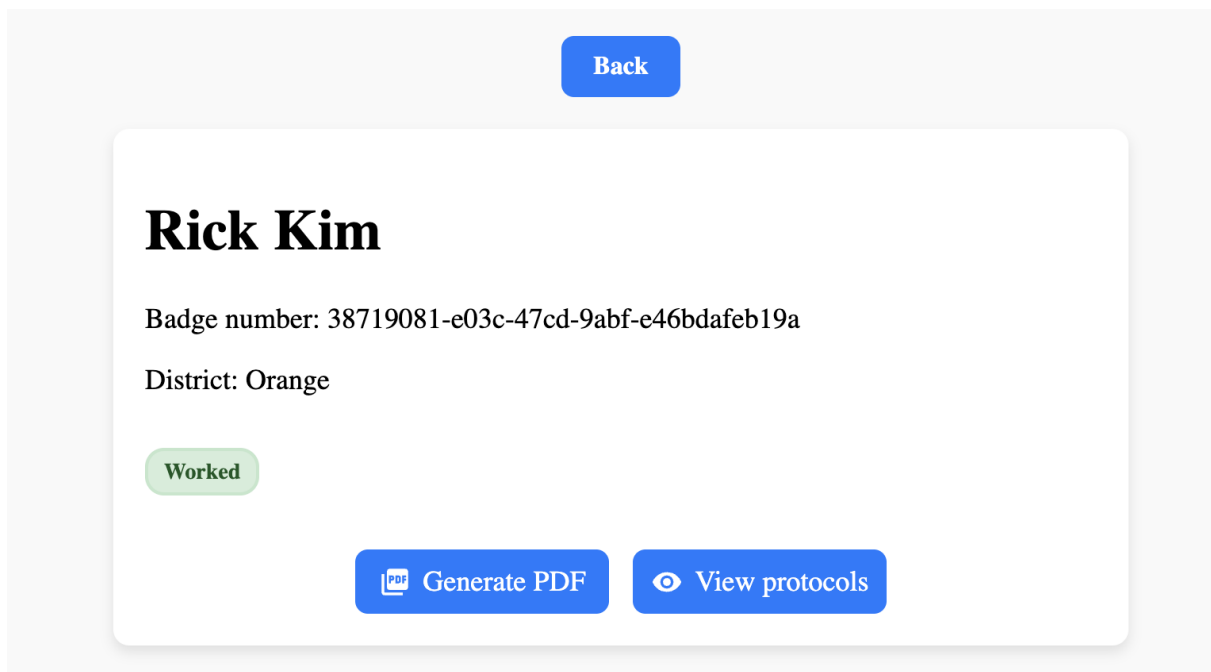


Рисунок 4.4 – Сторінка окремого інспектора

На сторінці окремого інспектора звичайний користувач (який не є адміністратором) має можливість згенерувати PDF-документ із інформацією про інспектора (рис. 4.5).

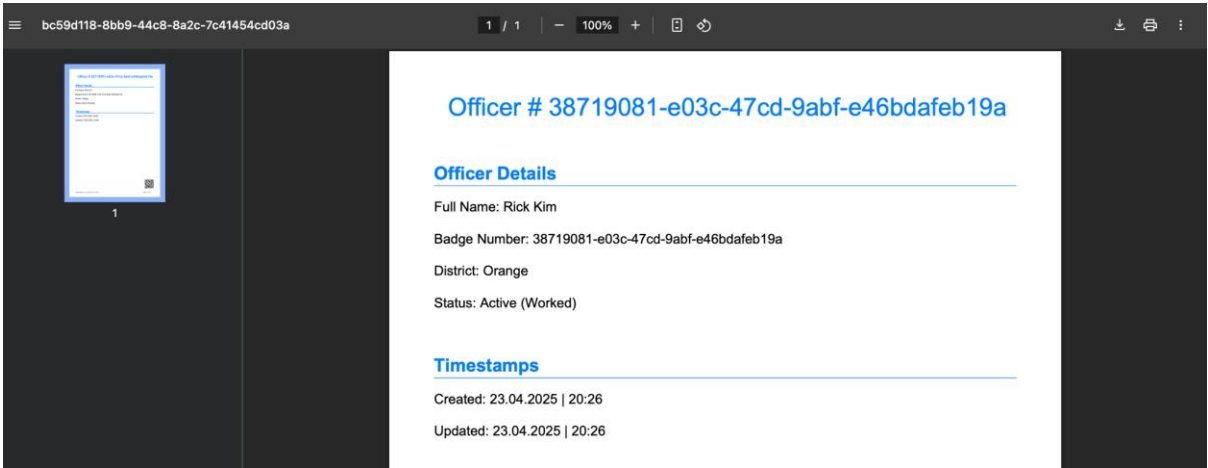


Рисунок 4.5 – PDF документ з інформацією про інспектора

Крім того, як зазначено вище, у нас є можливість переглянути протоколи.

Ось, наприклад, протоколи конкретного інспектора (рис. 4.6).

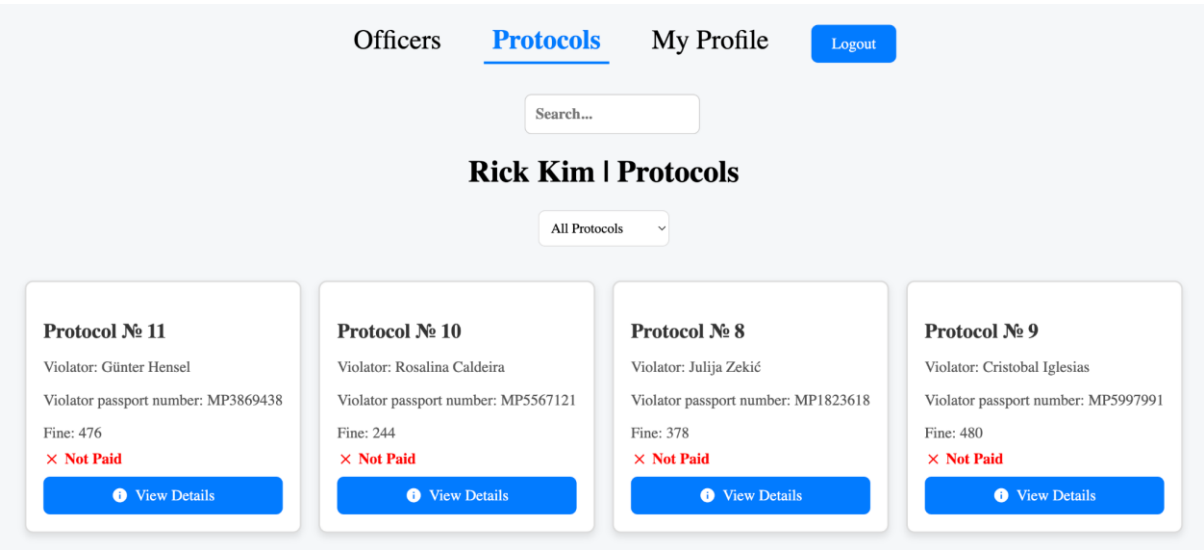


Рисунок 4.6 – Протоколи створені конкретним інспектором

На цій сторінці ми можемо переглядати інформацію про протоколи. Крім того, реалізовано фільтр, за допомогою якого ми можемо знайти якийсь певний потрібний нам протокол.

Пошук протоколів можна здійснювати за ID протоколу, розміром штрафу (наприклад, >50, <100, =75) або іншими критеріями: прізвище порушника, номер паспорта порушника, прізвище інспектора, номер жетону інспектора.

Натиснення на кнопку "View details" призведе до відкриття сторінки конкретного протоколу. На цій сторінці ми можемо побачити основну інформацію про протокол. Користувач може сплатити цей протокол (можливість сплати буде описана далі). Крім того, користувач може сформувати PDF з інформацією про протокол (рис. 4.7).

Protocol № 10

Details

Service notes: Violation of parking rules

Fine amount: 244

Violator full name: Rosalina Caldeira

Violator passport number: MP5567121

Timestamps

Created: 23.04.2025 | 20:27 | 14 minutes ago

Updated: 23.04.2025 | 20:27 | 14 minutes ago

Officer Details

Officer full name: Rick Kim

Officer badge number: 38719081-e03c-47cd-9abf-e46bdafeb19a

Payment Status

× Not Paid

☑ Pay protocol

PDF Generate PDF

Рисунок 4.7 – Сторінка конкретного протоколу

Також ми можемо сформувати PDF з інформацією про протокол (рис. 4.8).

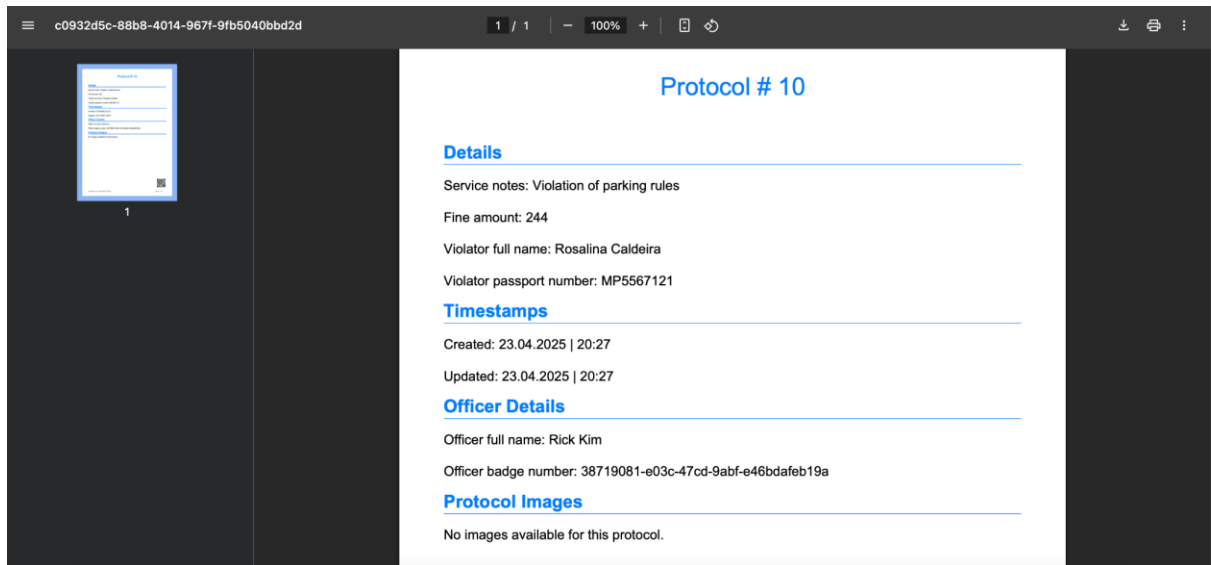


Рисунок 4.8 – PDF документ з інформацією про протокол

Як було описано вище, ще у нас є можливість сплатити протокол. Для того, щоб користувач міг сплатити за протокол, нам необхідно прив'язати до свого профілю банківську картку. Далі буде описано, як це робити.

У користувача в меню є вкладка "My profile". При переході на цю вкладку ми побачимо основну інформацію про профайл, яка зберігається в СКБД (рис. 4.9).



Рисунок 4.9 – Вкладка "My profile"

Натиснення на кнопку Add card призводить до відкриття форми додавання картки. Користувач може прив'язати картку до свого профілю (рис. 4.10). Картка валідуються.

My Profile

Nickname: Роман Ратушний

Email: remenok9@gmail.com

Role: user

Add Your Card



4111 1111 1111 1111

RATUSHNYI

valid thru 08/28

MongoDB Co

Рисунок 4.10 – Вигляд форми додавання картки

Після додавання картки ми зможемо побачити її у своєму профілі (рис. 4.11). Крім того, там же ми можемо цю картку видалити.

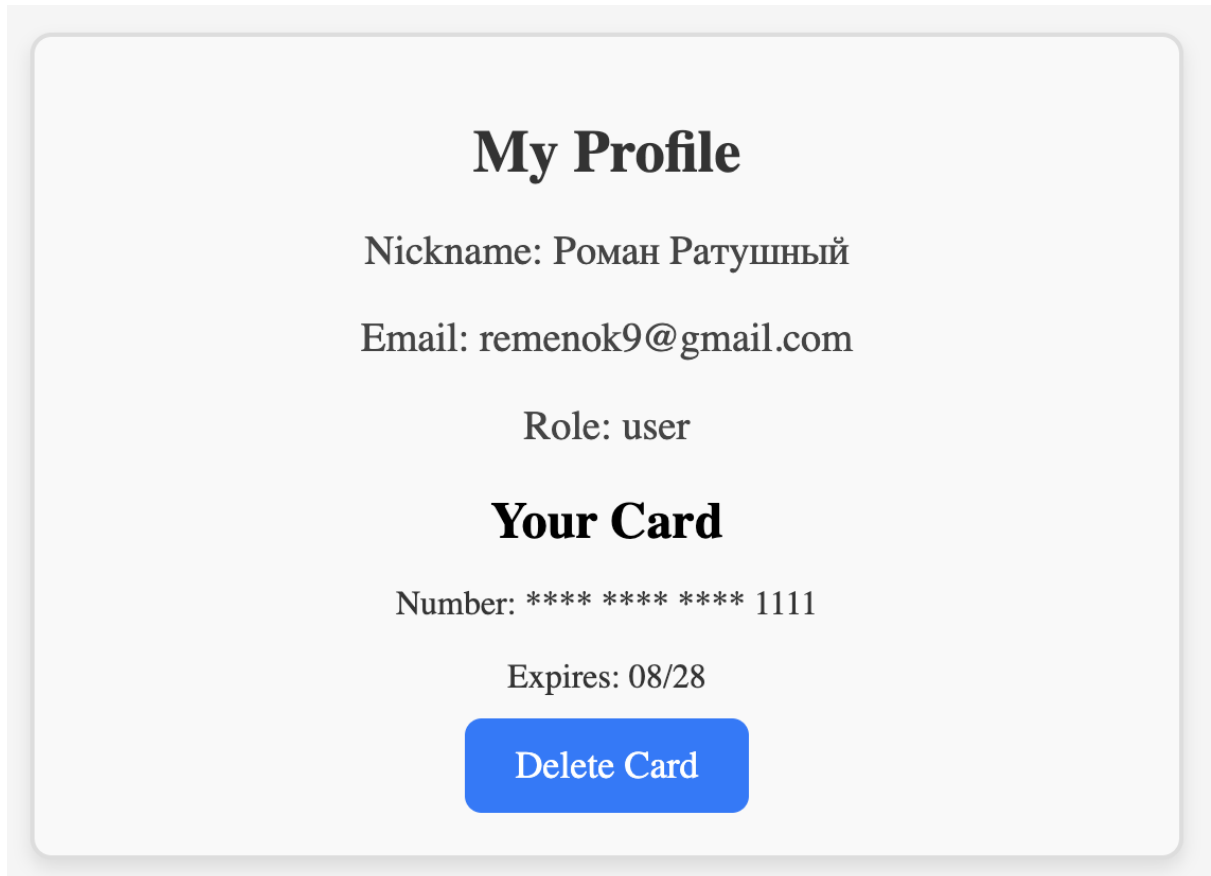


Рисунок 4.11 – Вигляд вкладки "My profile" після додавання картки

Якщо користувач є адміністратором, цьому користувачеві доступна спеціальна вкладка — "A-panel". У цьому розділі адміністратор може переглядати список усіх зареєстрованих користувачів, включаючи тих, які були заблоковані. Крім того, доступна можливість перегляду журналів дій (логів), що дозволяє здійснювати аудит змін та контролювати активність у системі.

Цей функціонал сприяє підвищенню прозорості роботи та забезпечує ефективне управління користувачами й безпекою системи (рис. 4.12).

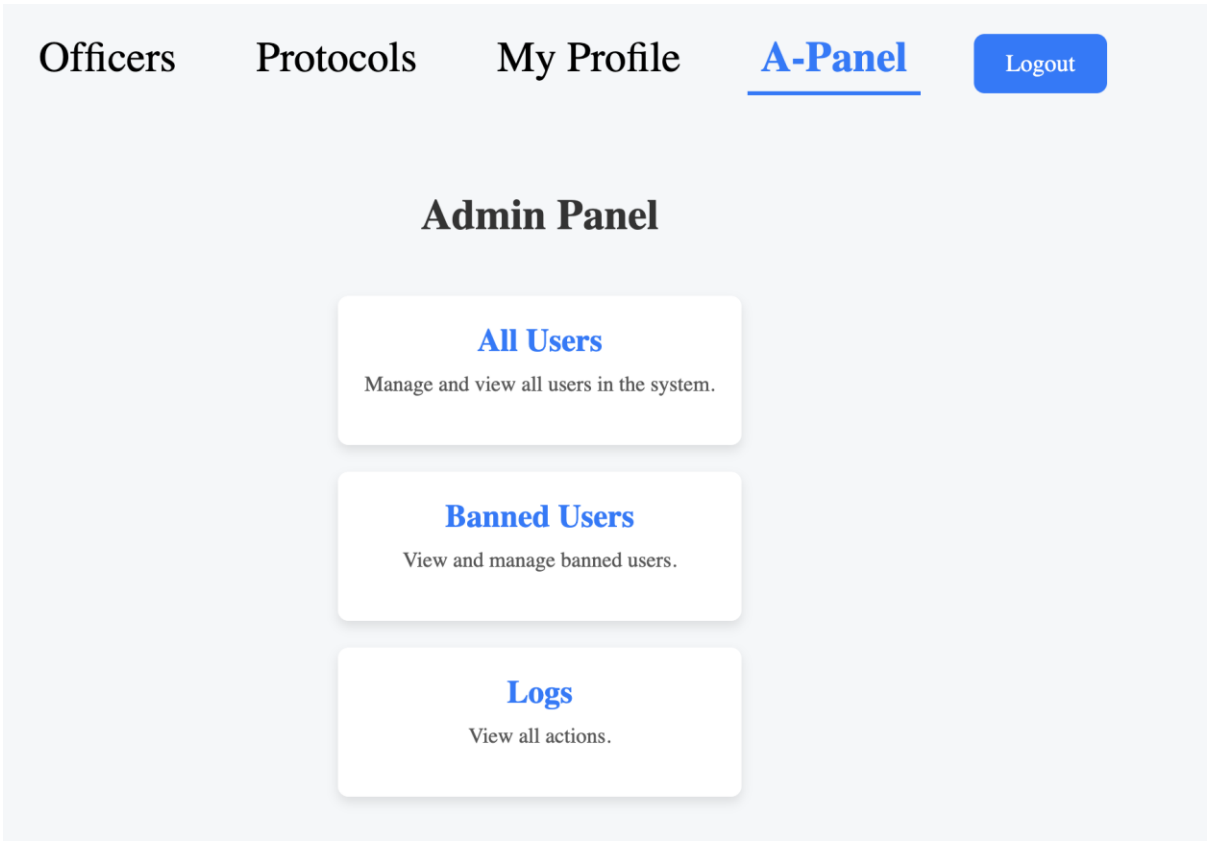


Рисунок 4.12 – Вкладка "A-panel"

Як вже було зазначено вище, тут адміністратору доступно 3 опції: може переглянути взагалі всіх користувачів, переглянути забаних користувачів та переглянути логи (рис. 4.13).

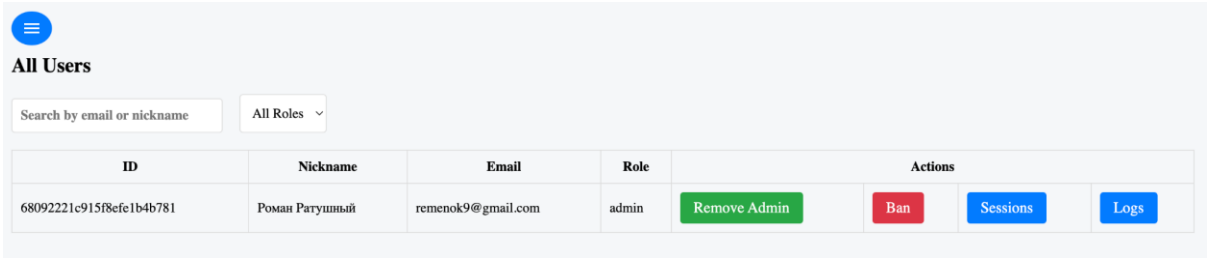


Рисунок 4.13 – Сторінка "All users"

На сторінці All users ми можемо побачити взагалі всіх користувачів, які зареєстровані в системі. Крім того, ми можемо побачити системну інформацію.

На цій сторінці ми можемо призначити/зняти адміністративну роль, заблокувати користувача, переглянути сесії користувача та переглянути логи конкретного користувача.

Крім того, на цій сторінці ми можемо здійснювати пошук користувачів за роллю користувача та/або нікнеймом або адресою електронної пошти (рис. 4.14).

Back

User Sessions

Роман Ратушний | Sessions

Session Created At	IP Address	Provider Info	Geolocation	Geolocation Info	Operating System	Browser	Session Status
23.04.2025 21:16 less than a minute ago	146.120.165.159	Ukraine, Zaporizhzhya, Dianet Ltd.	47.88267000658994 35.07153020297373	Ukraine, Zaporizhia Oblast, Zaporizhzhia, Petra Sahaidachnoho street	Mac OS	Chrome 135	End session
23.04.2025 21:00 16 minutes ago	146.120.165.159	Unknown	Unknown	Unknown	Mac OS	Chrome 135	Revoked
23.04.2025 20:23 about 1 hour ago	146.120.165.159	Unknown	Unknown	Unknown	Mac OS	Chrome 135	Revoked

Load More

Load All

Рисунок 4.14 – Вигляд сторінки сесії користувача

На цій сторінці ми можемо переглянути інформацію про всі активні та неактивні сесії користувача. Крім того, ми можемо примусово завершити сесію (рис. 4.15).

User Logs

Filter by Action Type: All Actions

Timestamp	Action Description	Action Type	Performed By
23.04.2025 21:20 less than a minute ago	ID: 68092221c915f8efe1b4b781 get park officer with ID: 2	get park officer by id	68092221c915f8efe1b4b781
23.04.2025 21:20 less than a minute ago	ID: 68092221c915f8efe1b4b781 dismiss park officer. Park officer ID: 2	dismiss park officer	68092221c915f8efe1b4b781
23.04.2025 21:20 less than a minute ago	ID: 68092221c915f8efe1b4b781 get park officer with ID: 2	get park officer by id	68092221c915f8efe1b4b781
23.04.2025 21:20 less than a minute ago	ADMIN ID: 68092221c915f8efe1b4b781 get all users	get all users	68092221c915f8efe1b4b781
23.04.2025 21:20 less than a minute ago	ADMIN ID: 68092221c915f8efe1b4b781 get all user logs. User ID: 68092221c915f8efe1b4b781	get all user logs	68092221c915f8efe1b4b781

Load More

Load All

Рисунок 4.15 – Вигляд сторінки логів

На цій сторінці ми можемо переглянути взагалі всі логи конкретного користувача. Крім того, ми можемо фільтрувати логи саме за тою подією, яка нас цікавить.

Крім адміністративної панелі, адміністратор має розширені можливості на вкладках "Protocols" та "Officers".

На вкладці "Officers" з'являється додаткова кнопка, за натисненням на яку ми можемо додати нового офіцера (рис. 4.16).

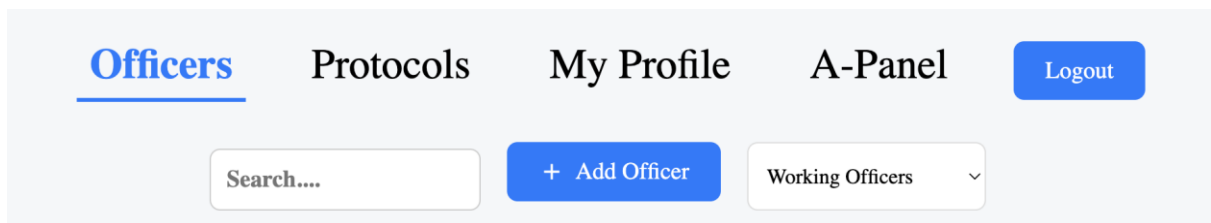


Рисунок 4.16 – Вигляд кнопки Add officer

Після натиснення на цю кнопку відкриється форма додавання нового офіцера (рис. 4.17).

The image shows a modal form titled 'Add officer' in bold black text. The form contains three input fields: 'Fullname:', 'Badge number:', and 'District:'. Each field is represented by a white rounded rectangle with a thin gray border. At the bottom of the form, there are two blue buttons: 'Add officer' and 'Cancel'.

Рисунок 4.17 – Вигляд форми додавання нового офіцера

Крім того, на сторінці конкретного офіцера також з'являються додаткові можливості (рис. 4.18).

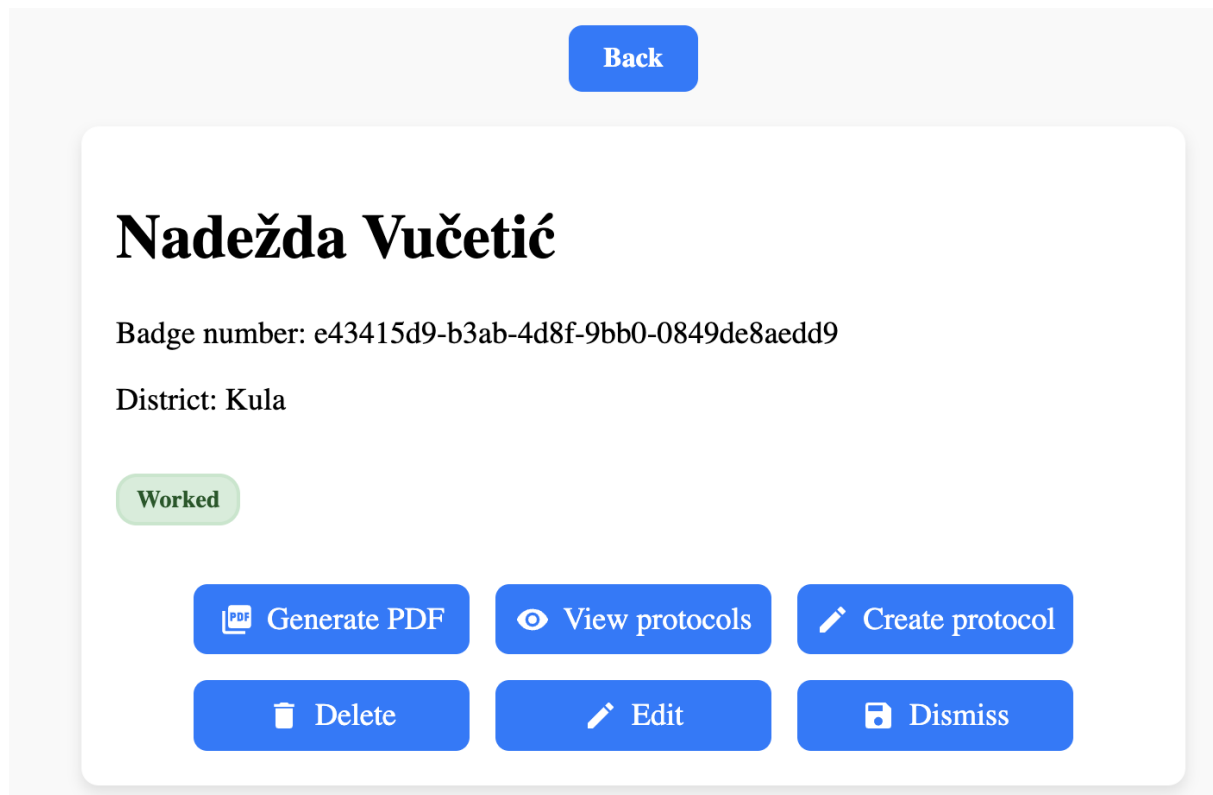


Рисунок 4.18 – Додаткові кнопки на сторінці конкретного офіцера

Система надає можливість виконувати широкий спектр дій щодо управління працівниками паркувальної інспекції. Зокрема, для обраного інспектора можна створити новий протокол про порушення, що дозволяє зафіксувати факт правопорушення безпосередньо в системі. Крім того, адміністратор має змогу редагувати персональні дані інспектора, вносити зміни до його службової інформації або оновлювати дані відповідно до актуального статусу (рис. 4.19).

У разі припинення трудових відносин передбачено функцію звільнення інспектора, яка деактивує його обліковий запис у системі без остаточного видалення історії дій.

Nadežda Vučetić | Create protocol

Service notes:

Fine amount:

Violator full name:

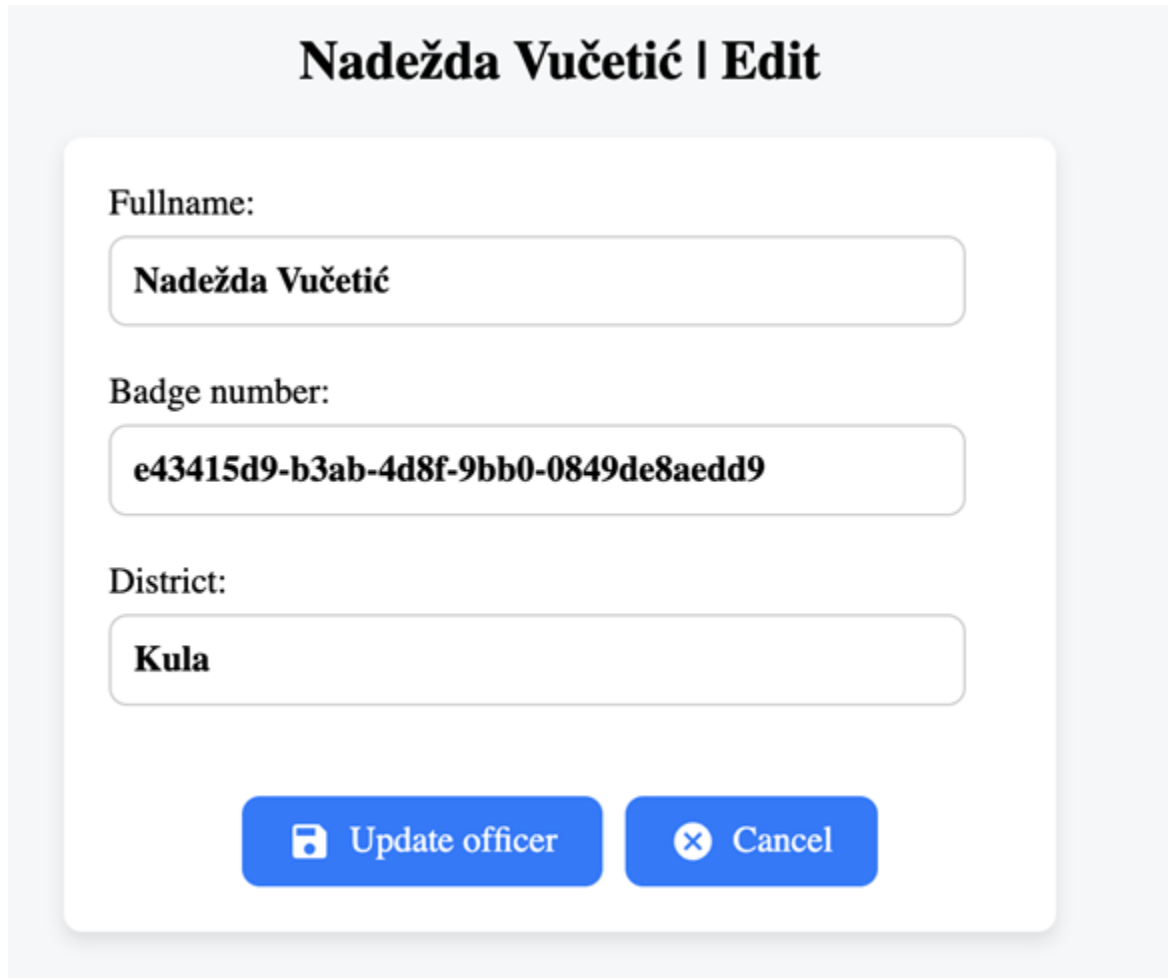
Violator passport number:

+ Create protocol

× Cancel

Рисунок 4.19 – Форма створення нового протоколу для конкретного офіцера

Як зазначено вище, є можливість редагувати мінформацію про існуючого інспектора. Нижче зображено форму редагування інформації про інспектора (рис. 4.20).



Nadežda Vučetić | Edit

Fullname:
Nadežda Vučetić

Badge number:
e43415d9-b3ab-4d8f-9bb0-0849de8aedd9

District:
Kula

 Update officer  Cancel

Рисунок 4.20 – Форма редагування інформації про конкретного офіцера

Крім того, як було зазначено раніше, адміністратор має додаткові функціональні можливості, які доступні безпосередньо на сторінці кожного конкретного протоколу. Зокрема, адміністратор може завантажити і додати до протоколу відповідне зображення, що є важливим для наочності та підтвердження факту правопорушення. Це дозволяє наочно фіксувати порушення та забезпечувати доказову базу для подальшої обробки.

Окрім цього, адміністратор може редагувати вже існуючі протоколи, змінюючи їх деталі чи доповнюючи новою інформацією, якщо це необхідно

для коректного відображення події. Важливою функцією є можливість видалення протоколу, що дозволяє прибрати будь-які дублікати або помилково створені записи, забезпечуючи таким чином чистоту та правильність даних у системі (рис. 4.21).

Protocol № 25

Details

Service notes: Violation of parking rules

Fine amount: 246

Violator full name: Naomi Guillaume

Violator passport number: MP1477502

Timestamps

Created: 23.04.2025 | 20:27 | about 1 hour ago

Updated: 23.04.2025 | 20:27 | about 1 hour ago

Officer Details

Officer full name: Andre Marie

Officer badge number: 6d9f5492-ac69-4eab-adde-ecf9f309cdc0

Payment Status

✕ Not Paid

 Pay protocol

 Generate PDF

 Delete

 Edit

 Add Image(s)

Рисунок 4.21 – Додаткові можливості адміністратора на сторінці конкретного протоколу

Як зазначено вище, на сторінці протоколу адміністратор може виконувати кілька важливих операцій. Зокрема, є можливість видалити

протокол, редагувати його інформацію. Ці функції забезпечують зручність управління протоколами та коректність даних у системі (рис. 4.22).

Protocol № 25 | Edit

Service Notes:

Violation of parking rules

Fine amount:

246

Violator full name:

Naomi Guillaume

Violator passport number:

MP1477502

 Update protocol

 Cancel

Рисунок 4.22 – Форма редагування інформації про протокол

Також система передбачає можливість додавання зображень безпосередньо до протоколу, що є важливим елементом у процесі документування правопорушення. Завдяки цій функції адміністратор або

інспектор може прикріпити фотофіксацію, яка слугує візуальним підтвердженням факту порушення правил паркування, що значно підвищує обґрунтованість і достовірність складеного протоколу (рис. 4.23).

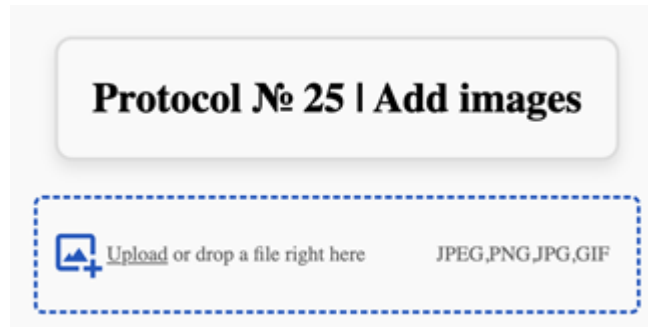


Рисунок 4.23 – Додавання зображення до протоколу

При додаванні зображень може використовуватись Drag & Drop.

4.4 Висновки за розділом 4

У ході експлуатації та тестування програмного забезпечення для автоматизації процесів паркувальної інспекції було підтверджено відповідність системи визначеним функціональним вимогам. Реалізоване рішення продемонструвало стабільну роботу в умовах типових сценаріїв використання та коректне реагування на нестандартні ситуації, включно з обробкою помилок. Особливу увагу було приділено перевірці адміністративного функціоналу, зокрема моніторингу сесій, керуванню доступом, перегляду логів дій та контролю за активністю інспекторів, що дало змогу впевнено оцінити надійність системи з точки зору інформаційної безпеки.

У результаті функціонального та інтеграційного тестування підтверджено повноту реалізації ключових можливостей, таких як реєстрація та авторизація користувачів, створення та редагування протоколів порушень, управління інспекторами, додавання фотофіксацій, а також можливість взаємодії між ролями адміністратора, інспектора та звичайного користувача.

Система продемонструвала очікувану поведінку при навмисному введенні некоректних даних, що свідчить про наявність необхідних перевірок і захисту від помилок користувача.

Завдяки структурованому підходу до тестування, включаючи як ручну перевірку інтерфейсів, так і автоматизоване тестування окремих модулів, вдалося досягти високого рівня якості програмного продукту. Таким чином, можна зробити висновок, що програмне забезпечення є готовим до практичного використання, забезпечує повний набір необхідних функцій і відповідає вимогам, які були поставлені на етапі проєктування системи.

ВИСНОВКИ

Виходячи з отриманих результатів, у роботі було досягнуто мету, якою є розробка програмного забезпечення для автоматизації діяльності паркувальної інспекції, що сприяє підвищенню ефективності контролю за дотриманням правил паркування.

Виконання завдання аналізу наявного програмного забезпечення у сфері контролю за паркуванням дозволило виявити, що більшість існуючих рішень або є частиною закритих систем без гнучких можливостей для модернізації, або не забезпечують повного циклу роботи інспекції — від фіксації порушення до оплати штрафу. На усунення цих недоліків була спрямована дана розробка.

Для визначення функціональних вимог до майбутнього застосунку було побудовано діаграму прецедентів, що охоплює основні сценарії взаємодії користувачів: інспекторів, адміністраторів і звичайних користувачів. Це дозволило сформулювати чітке бачення архітектури та розробити ефективну логіку взаємодії системи.

У результаті виконання завдань з вибору технологічного стеку було обрано JavaScript як основну мову, а також фреймворки Express.js і React для створення серверної та клієнтської частин відповідно. Для зберігання даних застосовувалися MongoDB та PostgreSQL, а також ORM-бібліотеки Sequelize та Mongoose. Для автентифікації використано JWT, а для розгортання — Docker.

Загальним результатом є створений вебзастосунок, який забезпечує повний життєвий цикл обробки порушень паркування: реєстрація та автентифікація користувачів, створення протоколів, прикріплення фотофіксацій, обробка оплат, адміністрування ролей та облікових записів. Застосунок створено з урахуванням можливості масштабування, захисту даних і зручного інтерфейсу для кожної ролі користувача.

У підсумку, програмне забезпечення призначене для автоматизації процесів паркувальної інспекції, оптимізації роботи інспекторів, зменшення навантаження на адміністративний персонал та покращення користувацького досвіду при взаємодії з системою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Як це працює – Парксервіс Груп [Електрон. ресурс]. – Режим доступу: <https://psg.ua/yak-se-prasue/>.
2. Prozorro – Закупівля автоматизованої системи паркування [Електрон. ресурс]. – Режим доступу: <https://prozorro.gov.ua/tender/UA-2023-12-08-009963-a>.
3. Київ – Інспекція з паркування [Електрон. ресурс]. – Режим доступу: https://kyivcity.gov.ua/dorohy_transport_ta_parkovky/parkuvannia/upravlinnya_in_spektsiya_z_parkuvannya/.
4. Smart Parking Systems and Solutions [Electronic resource]. – Access mode: <https://www.smartparking.com/>.
5. EU Urban Mobility – Parking Management [Electronic resource]. – Access mode: <https://www.eltis.org/mobility-plans/parking-management>.
6. Parkopedia – Parking API for Cities and Developers [Electronic resource]. – Access mode: <https://business.parkopedia.com/>.
7. Recommendations for the Inspection, Maintenance and Management of Car Park Structures / National Steering Committee on the Inspection of Multi-storey Car Parks. – Thomas Telford Publishing, 2002. – 112 p. – [Electronic resource]. – Access mode: <https://archive.org/details/recommendationsf0000inst>.
8. Car Parking System Standard Requirements / Gerardus Blokdyk. – 5STARCook, 2022. – 308 p. – [Electronic resource]. – Access mode: <https://www.lodige.com/fileadmin/Downloads/doc-cps-e/Respace/Lodige-CarParkSystem-RESPACE-Planning-guide-en-w.pdf>.
9. Вагомі підстави і принципи управління паркуванням / Robert Pressl, Tom Rye. – 2020. – 40 с. – Режим доступу: https://park4sump.eu/sites/default/files/2020-07/PARK4SUMP_reasons_15072020_UA_web.pdf.
10. React – JavaScript library for building user interfaces [Electronic resource]. – Access mode: <https://reactjs.org/>.

11. Express – Fast, unopinionated, minimalist web framework for Node.js [Electronic resource]. – Access mode: <https://expressjs.com/>.
12. Koa – Next generation web framework for Node.js [Electronic resource]. – Access mode: <https://koajs.com/>.
13. NGINX – High performance load balancer, web server, & reverse proxy [Electronic resource]. – Access mode: <https://nginx.org/>.

ДОДАТОК А
Технічне завдання

Вступ

У сучасних містах проблема контролю за дотриманням правил паркування стає все більш актуальною. Неправильне паркування створює значні незручності для водіїв, пішоходів та громадського транспорту, що може призводити до заторів, аварійних ситуацій та ускладнення роботи екстрених служб. Для забезпечення порядку в цій сфері муніципальні органи запроваджують системи контролю та штрафування порушників. Одним із ключових елементів такого контролю є діяльність паркувальних інспекторів, які фіксують правопорушення та складають відповідні адміністративні протоколи.

Проте процес фіксації порушень часто є трудомістким та потребує значних часових і людських ресурсів. У всіх випадках порушення мають бути задокументовані за допомогою фотофіксації, щоб уникнути безпідставних штрафів та забезпечити можливість доказової бази у разі оскарження штрафу в судовому порядку. Крім того, паркувальним інспекторам необхідно зберігати велику кількість інформації, що стосується кожного зафіксованого випадку, включаючи службові примітки та інші дані.

Метою даної роботи є розробка програмного забезпечення для автоматизації процесів паркувальної інспекції, яке дозволить спростити та вдосконалити роботу паркувальних інспекторів, забезпечуючи ефективний збір, збереження та обробку інформації про порушення.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності та на підставі Наказу № 209 від 28 квітня 2025 р. за Національним університетом «Запорізька політехніка», яким визначено тему роботи – «Програмне забезпечення паркувальної інспекції».

A.2 Призначення розробки

Програма призначена для автоматизації процесів паркувальної інспекції, фіксації та обробки порушень правил паркування, ведення електронного реєстру порушень, взаємодії між інспекторами, адміністраторами та користувачами, формування та перегляду протоколів, прикріплення фотофіксацій, адміністрування облікових записів, обробки оплат штрафів та контролю за виконанням постанов.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні вимоги програми для паркувальної інспекції повинні включати:

Основні сценарії (функціональні вимоги):

- а) авторизація користувача;
- б) перегляд та редагування власного профілю;
- в) робота з адміністративною панеллю:
 - 1) перегляд усіх зареєстрованих користувачів;
 - 2) керування заблокованими користувачами;
 - 3) перегляд сесій конкретного користувача;
 - 4) перегляд логів дій користувачів;
- г) робота з працівниками паркувальної інспекції:
 - 1) перегляд списку інспекторів;
 - 2) детальна інформація про конкретного інспектора;
 - 3) створення нового запису інспектора;
 - 4) редагування існуючого запису інспектора;
- г) робота з протоколами правопорушень:
 - 1) перегляд загального списку протоколів;
 - 2) перегляд протоколів конкретного інспектора;

- 3) створення нового протоколу;
- 4) додавання фотофіксації до протоколу;
- 5) редагування інформації протоколу;
- 6) перегляд деталей конкретного протоколу;
- 7) оплата штрафу;

д) робота з адміністративною панеллю:

- 1) моніторинг статистики;
- 2) контроль прав доступу;
- 3) управління системними процесами;

є) відновлення паролю та зміна пароля після втрати доступу;

ж) обробка некоректних запитів через сторінку 404.

Альтернативні сценарії:

- скасування створення протоколу;
- повторне додавання фотофіксації у випадку неуспішного завантаження;
- перевизначення інформації інспектора після виявлення помилки;
- повторна активація заблокованого користувача адміністратором;
- ручне завершення активної сесії користувача з підозрілою активністю.

A.3.2 Умови експлуатації

Обов'язковою умовою експлуатації програми для автоматизації процесів паркувальної інспекції є наявність адміністратора системи або відповідального користувача, який має повноваження на адміністрування вебзастосунку. Також адміністратор повинен виконати розгортання додатку на сервері.

Окрім цього, адміністратор відповідає за первинне налаштування параметрів системи, створення облікових записів користувачів та

призначення ролей. Його роль є ключовою для забезпечення безперебійної роботи вебзастосунку, контролю доступу та підтримки безпеки даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Мінімальні вимоги до технічного забезпечення сервера для ефективного функціонування системи включають:

- процесор Intel Xeon E5440 (частота 2,83 ГГц) або еквівалентний;
- жорсткий диск обсягом не менше 100 Гб;
- оперативна пам'ять обсягом не менше 4 Гб.

A.4 Порядок контролю та приймання

Починаючи з дати видачі завдання – 28 квітня, протягом наступних 8 тижнів має бути забезпечено виконання роботи за календарним планом, що містить в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом – строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів формується пояснювальна записка до дипломної кваліфікаційної роботи. Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б
Текст програми

```

const { Protocol, ParkOfficer, Image } = require('../models');
const { Log } = require('../models/MongoDB');
const createHttpError = require('http-errors');
const { deleteImageFromDisk } = require('../utils');
const { LOG_ACTION_TYPES } = require('../config/logActionTypes');
const include = [
  { model: ParkOfficer, attributes: ['id', 'full_name', 'badge_number'], as:
'parkOfficer' },
  { model: Image, attributes: ['id', 'path'], as: 'images' },
];
module.exports.getAllProtocols = async (req, res, next) => {
  try {
    const { pagination } = req;
    const data = await Protocol.findAll({ include, order: [['updated_at',
'DESC']], ...pagination });
    const totalProtocolsCount = await Protocol.count();
    res.status(200).send({ data, totalProtocolsCount });
  } catch (e) { next(e); }
};
module.exports.getProtocolById = async (req, res, next) => {
  try {
    const { id } = req.params;
    const data = await Protocol.findOne({ where: { id }, include });
    res.status(200).send({ data });
  } catch (e) { next(e); }
};
module.exports.getAllProtocolsByOfficerID = async (req, res, next) => {
  try {
    const { officerId } = req.params, { pagination } = req;
    const data = await Protocol.findAll({ where: { officerId }, include, order:
[['updated_at', 'DESC']], ...pagination });
    const totalProtocolsCount = await Protocol.count({ where: { officerId }
});
    res.status(200).send({ data, totalProtocolsCount });
  } catch (e) { next(e); }
};
module.exports.createProtocol = async (req, res, next) => {
  try {
    const { officerId } = req.params, { body, files, tokenPayload: { userId }
} = req;
    const officer = await ParkOfficer.findByPk(officerId);
    if (!officer || !officer.isWorked) return next(createHttpError(400, 'Park
officer is not active or does not exist'));
    const created = await Protocol.create({ ...body, officerId });
    if (!created) return next(createHttpError(400, 'Protocol not created'));
  }

```

```

    if (files?.length) await Image.bulkCreate(files.map(f => ({ path:
f.filename, protocolId: created.id })));
    const data = await Protocol.findAll({ where: { id: created.id }, include,
order: [['updated_at', 'DESC']] });
    await Log.createLog({ actionType:
LOG_ACTION_TYPES.CREATE_PROTOCOL, description: `ID: ${userId}
create protocol. Protocol ID: ${created.id}`, performedBy: userId });
    res.status(201).send({ data });
  } catch (e) { next(e); }
};
module.exports.updateProtocolByID = async (req, res, next) => {
  try {
    const { id } = req.params, { body, files, tokenPayload: { userId } } = req;
    const [count, [updated]] = await Protocol.update(body, { where: { id },
returning: true });
    if (files?.length) await Image.bulkCreate(files.map(f => ({ path:
f.filename, protocolId: updated.id })));
    if (!count) return next(createHttpError(404, 'Protocol not found'));
    const data = await Protocol.findAll({ where: { id: updated.id }, include,
order: [['updated_at', 'DESC']] });
    await Log.createLog({ actionType:
LOG_ACTION_TYPES.UPDATE_PROTOCOL, description: `ID: ${userId}
update protocol. Protocol ID: ${id}`, performedBy: userId });
    res.status(200).send({ data });
  } catch (e) { next(e); }
};
module.exports.deleteProtocolByID = async (req, res, next) => {
  try {
    const { id } = req.params, { userId } = req.tokenPayload;
    const protocol = await Protocol.findPk(id, { include });
    if (!protocol) return next(createHttpError(404));
    for (const img of protocol.images) await
deleteImageFromDisk(img.path);
    const count = await Protocol.destroy({ where: { id } });
    if (!count) return next(createHttpError(404, 'Protocol not found'));
    await Log.createLog({ actionType:
LOG_ACTION_TYPES.DELETE_PROTOCOL, description: `ID: ${userId}
delete protocol. Protocol ID: ${id} (protocol deleted)`, performedBy: userId });
    res.status(200).end();
  } catch (e) { next(e); }
};
module.exports.getProtocolsByViolatorPassportNumber = async (req, res,
next) => {
  try {
    const { passportNumber } = req.params;

```

```

    const data = await Protocol.findAll({ where: { violatorPassportNumber:
passportNumber }, include, order: [['updated_at', 'DESC']] });
    res.status(200).send({ data });
  } catch (e) { next(e); }
};
const log = (actionType, description, performedBy) => Log.createLog({
actionType, description, performedBy });
module.exports = {
  getAllParkOfficers: async (req, res, next) => {
    try {
      const parkOfficers = await ParkOfficer.findAll();
      res.status(200).send({ data: parkOfficers });
    } catch (e) { next(e); }
  },

  getParkOfficerByID: async (req, res, next) => {
    try {
      const { id } = req.params, { userId } = req.tokenPayload;
      const officer = await ParkOfficer.findAll({ where: { id } });
      if (!officer) return next(createHttpError(404, 'Park officer not found'));
      await log(LOG_ACTION_TYPES.GET_PARK_OFFICER_BY_ID,
`ID: ${userId} get park officer with ID: ${id}`, userId);
      res.status(200).send({ data: officer });
    } catch (e) { next(e); }
  },

  createParkOfficer: async (req, res, next) => {
    try {
      const { body, tokenPayload: { userId } } = req;
      const officer = await ParkOfficer.create(body);
      if (!officer) return next(createHttpError(400, 'Park officer not created'));
      await log(LOG_ACTION_TYPES.CREATE_PARK_OFFICER, `ID:
${userId} create park officer. Officer ID: ${officer.id}`, userId);
      res.status(201).send({ data: officer });
    } catch (e) { next(e); }
  },

  updateParkOfficerByID: async (req, res, next) => {
    try {
      const { id } = req.params, { body, tokenPayload: { userId } } = req;
      const [count, [officer]] = await ParkOfficer.update(body, { where: { id
}, returning: true });
      if (!count) return next(createHttpError(404, 'Park officer not found'));
      await log(LOG_ACTION_TYPES.UPDATE_PARK_OFFICER, `ID:
${userId} update park officer. Park officer ID: ${id}`, userId);
      res.status(200).send({ data: officer });
    } catch (e) { next(e); }
  }
};

```

```

    },
    deleteParkOfficerByID: async (req, res, next) => {
      try {
        const { id } = req.params, { userId } = req.tokenPayload;
        const officer = await ParkOfficer.findOne({ where: { id }, include: {
model: Protocol, as: 'protocols', attributes: ['id'] } });
        if (officer.protocols.length) return next(createHttpError(400, 'You
cannot delete this officer. The officer has protocols.));
        const count = await ParkOfficer.destroy({ where: { id } });
        if (!count) return next(createHttpError(404, 'Park officer not found'));
        await log(LOG_ACTION_TYPES.DELETE_PARK_OFFICER, `ID:
${userId} delete park officer. Park officer ID: ${id}`, userId);
        res.status(200).end();
      } catch (e) { next(e); }
    },
    dismissParkOfficerByID: async (req, res, next) => {
      try {
        const { id } = req.params, { userId } = req.tokenPayload;
        const [count, [officer]] = await ParkOfficer.update({ isWorked: false },
{ where: { id }, returning: true });
        if (!count) return next(createHttpError(404, 'Park officer not found'));
        await log(LOG_ACTION_TYPES.DISMISS_PARK_OFFICER, `ID:
${userId} dismiss park officer. Park officer ID: ${id}`, userId);
        res.status(200).send({ data: officer });
      } catch (e) { next(e); }
    },
    restoreParkOfficerByID: async (req, res, next) => {
      try {
        const { id } = req.params, { userId } = req.tokenPayload;
        const [count, [officer]] = await ParkOfficer.update({ isWorked: true },
{ where: { id }, returning: true });
        if (!count) return next(createHttpError(404, 'Park officer not found'));
        await log(LOG_ACTION_TYPES.RESTORE_PARK_OFFICER, `ID:
${userId} restore park officer. Park officer ID: ${id}`, userId);
        res.status(200).send({ data: officer });
      } catch (e) { next(e); }
    }
  };

```

```

import React, { useEffect, useState, useRef } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { getAllProtocols, getAllProtocolsByOfficerID } from
'../redux/slices/protocolSlice';
import { isProtocolsPaid } from '../redux/slices/paymentSlice';
import { useParams, Link } from 'react-router-dom';

```

```

import { getParkOfficerByID } from '../API';
import styles from './ProtocolsPage.module.scss';
import Pagination from '../components/Pagination/Pagination';
import NavBar from '../components/NavBar/NavBar';
import UserSidebar from '../components/UserSidebar/UserSidebar';
import NotFound from '../components/NotFound/NotFound';
import filterProtocols from '../utils/filterProtocols';
import { Info, Done, Close } from '@mui/icons-material';
import CONSTANTS from '../constants';
const { LIMIT, MOBILE_WIDTH } = CONSTANTS;
const ProtocolsPage = () => {
  const { parkOfficerID } = useParams();
  const dispatch = useDispatch();
  const { protocols, error, totalProtocolsCount } = useSelector(state =>
state.protocols);
  const { protocolsPaymentStatuses } = useSelector(state =>
state.payments);

  const [searchValue, setSearchValue] = useState("");
  const [searchType, setSearchType] = useState('default');
  const [filter, setFilter] = useState('all');
  const [pageNumber, setPageNumber] = useState(1);
  const [totalPages, setTotalPages] = useState(1);
  const [showPagination, setShowPagination] = useState(true);
  const [isMobile, setIsMobile] = useState(window.innerWidth <
MOBILE_WIDTH);
  const [isSidebarOpen, setIsSidebarOpen] = useState(false);
  const [showSelect, setShowSelect] = useState(false);
  const [parkOfficerFullName, setParkOfficerFullName] = useState(null);
  const searchContainerRef = useRef(null);
  useEffect(() => {
    if (parkOfficerID) {
      getParkOfficerByID(parkOfficerID)
        .then(({ data: { data } }) =>
setParkOfficerFullName(data[0].fullName))
        .catch(console.error);
    }
  }, [parkOfficerID]);

  useEffect(() => {
    const fetchData = () => {
      const fetchFn = parkOfficerID ? getAllProtocolsByOfficerID :
getAllProtocols;
      const args = parkOfficerID ? { parkOfficerID, page: pageNumber } :
pageNumber;

```

```

        dispatch(showPagination ? fetchFn(args) : fetchFn(parkOfficerID ? {
parkOfficerID } : undefined));
    };
    fetchData();
    setTotalPages(Math.ceil(totalProtocolsCount / LIMIT));
    }, [pageNumber, totalProtocolsCount, dispatch, parkOfficerID,
showPagination]);
    useEffect(() => {
        if (protocols.length) {
            const ids = protocols.map(p => p.id);
            dispatch(isProtocolsPaid(ids));
        }
    }, [protocols, dispatch]);
    useEffect(() => {
        const resizeHandler = () => setIsMobile(window.innerWidth <
MOBILE_WIDTH);
        window.addEventListener('resize', resizeHandler);
        return () => window.removeEventListener('resize', resizeHandler);
    }, []);

    useEffect(() => {
        const clickOutside = (e) => {
            if (searchContainerRef.current &&
!searchContainerRef.current.contains(e.target)) {
                setShowSelect(false);
            }
        };
        document.addEventListener('mousedown', clickOutside);
        return () => document.removeEventListener('mousedown',
clickOutside);
    }, []);
    if (error) return <div>ERROR HAPPENED</div>;
    const filtered = filterProtocols(protocols, searchValue, searchType,
protocolsPaymentStatuses, filter);
    return (
        <>
            {isMobile ? <UserSidebar isOpen={isSidebarOpen} toggleSidebar={()
=> setIsSidebarOpen(!isSidebarOpen)} /> : <NavBar />}
            <section>
                <div className={styles['flex-center']}>
                    <div ref={searchContainerRef} className={styles['search-
container']}>
                        <div className={styles['flex-center-column']}>
                            <input
                                value={searchValue}

```

```

        onChange={e => setSearchValue(e.target.value)}
        onFocus={() => setShowSelect(true)}
        placeholder="Search..."
      />
      {showSelect && (
        <select value={searchType} onChange={e =>
setSearchType(e.target.value)} className={styles['filter-select']}>
          <option value="default">Default Search</option>
          <option value="byID">Search by Protocol ID</option>
        </select>
      )}
    </div>
    <div className={styles.tooltip}>
      {searchType === 'byID'
        ? 'Search by Protocol ID (e.g., 12345)'
        : 'Search by fine or violator/officer info'}
    </div>
  </div>
</div>
<h1 className={styles.header}>
  {parkOfficerFullName ? `${parkOfficerFullName} | Protocols` : 'All
protocols'}
</h1>
<div className={styles['filter-container']}>
  <select value={filter} onChange={e => setFilter(e.target.value)}
className={styles['filter-select']}>
    <option value="all">All Protocols</option>
    <option value="paid">Paid Protocols</option>
    <option value="unpaid">Unpaid Protocols</option>
  </select>
</div>
<div className={styles['protocols-list']}>
  {filtered.map(protocol => {
    const paid = protocolsPaymentStatuses.find(p => p.protocolId ===
protocol.id)?.isPaid;
    return (
      <div key={protocol.id} className={styles['protocol-card']}>
        <h3>Protocol № {protocol.id}</h3>
        <p>Violator: {protocol.violatorFullName}</p>
        <p>Passport: {protocol.violatorPassportNumber}</p>
        <p>Fine: {protocol.fineAmount}</p>
        <div className={styles['payment-status']}>
          {paid ? <span className={styles['status-paid']}><Done
fontSize="small" /> Paid</span>

```

```

      : <span className={styles['status-unpaid']}><Close
fontSize="small" /> Not Paid</span>}
    </div>
    <Link to={`/protocols/details/${protocol.id}`}>
      <button className={styles.button}><Info fontSize="small" />
View Details</button>
    </Link>
  </div>
  );
  })}
</div>
  {!protocols.length && <NotFound />}
  {!searchValue && showPagination && totalPages > 1 && (
    <Pagination currentPage={pageNumber} totalPages={totalPages}
onPageChange={setPageNumber} />
  )}
  {totalPages > 1 && (
    <div className={styles['pagination-toggle-wrapper']}>
      <button onClick={() => setShowPagination(!showPagination)}>
        {showPagination ? 'Show All Protocols' : 'Return pages'}
      </button>
    </div>
  )}
</section>
</>
);
};
export default ProtocolsPage;

```

ДОДАТОК В
Слайди презентації

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЗАПОРІЗЬКА ПОЛІТЕХНІКА”
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ**

**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ
SOFTWARE FOR PARKING INSPECTION**

Виконавець: Роман РАТУШНИЙ
Керівник: к.т.н., доцент Степан СКРУПСЬКИЙ

2025

Рисунок В.1 – Перший слайд

ОБ’ЄКТ, ПРЕДМЕТ, МЕТА РОБОТИ

- об’єкт роботи – процес фіксації та обробки порушень правил паркування;
- предмет роботи – програмне забезпечення для автоматизації діяльності паркувальної інспекції;
- метою роботи є розробка вебзастосунку, який автоматизує процеси взаємодії між інспекторами, адміністраторами та користувачами у сфері контролю за паркуванням, з можливістю створення протоколів, додавання фотофіксацій, обробки оплат та адміністрування користувачів.

2

Рисунок В.2 – Об’єкт, предмет, мета роботи

ЗАВДАННЯ РОБОТИ

- проаналізувати програмне забезпечення для автоматизації процесів паркувальної інспекції;
- визначити засоби проектування системи;
- сформулювати вимоги до програмного забезпечення для контролю за паркуванням;
- реалізувати систему, що відповідає поставленим вимогам.

3

Рисунок В.3 – Завдання роботи

ПОРІВНЯННЯ ПРОГРАМ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ РОБОТИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ

Критерій	Розробка	Parkt Service Group	Автоматизація в м. Дніпро
Фіксація правопорушень	Автоматизований процес з GPS-прив'язкою.	Є така можливість, але без прив'язки до GPS.	Автоматизований процес з GPS-прив'язкою.
Контроль доступу	Гнучка система розмежування прав доступу.	Базова авторизація без розширеного управління ролями.	Відсутні детальні рівні доступу для інспекторів та адміністраторів.
Адміністративна панель	Повноцінна адмінка з можливістю керування користувачами.	Немає	Є

4

Рисунок В.4 – Порівняння програм для автоматизації процесів роботи паркувальної інспекції

ВИБІР ЗАСОБІВ ДЛЯ РОЗРОБКИ ФРОНТЕНД ЧАСТИНИ ПРОГРАМИ ДЛЯ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ

Критерій	React	Vue.js
Крива навчання	Потребує глибшого розуміння JavaScript та суміжних бібліотек	Простий в освоєнні для новачків
Масштабованість	Ідеально підходить для великих, складних проєктів	Підходить переважно для малих і середніх проєктів
Підтримка екосистеми	Широка екосистема, підтримка Next.js, Redux, React Query	Власна екосистема, Nuxt.js, Vuex
Спільнота	Надзвичайно велика, постійно оновлювана	Менша, але активно розвивається

5

Рисунок В.5 – Вибір засобів для розробки фронтенд частини програми для паркувальної інспекції

ВИБІР ЗАСОБІВ ДЛЯ РОЗРОБКИ БЕКЕНД ЧАСТИНИ ПРОГРАМИ ДЛЯ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ

Критерій	Express.js	Koa.js
Складність	Простий, мінімалістичний фреймворк	Ще легший, але потребує більше налаштувань вручну
Швидкість старту	Висока, мінімум конфігурації	Потребує додаткових модулів для базової функціональності
Документація та підтримка	Широка документація, багаторічна історія	Менш поширений, але зручний для досвідчених розробників
Ідеально підходить для	CRUD-додатків, REST API, швидкої побудови backend'у	Високонавантажених застосунків, потребує кастомних рішень

6

Рисунок В.6 – Вибір засобів для розробки бекенд частини програми для паркувальної інспекції

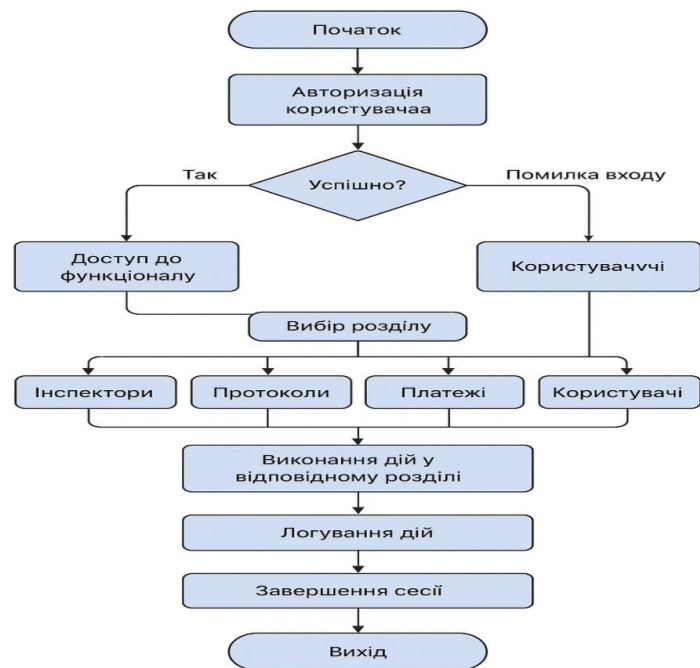
ДІАГРАМА ПРЕЦЕДЕНТІВ ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ



7

Рисунок В.7 – Діаграма прецедентів для програми паркувальної інспекції

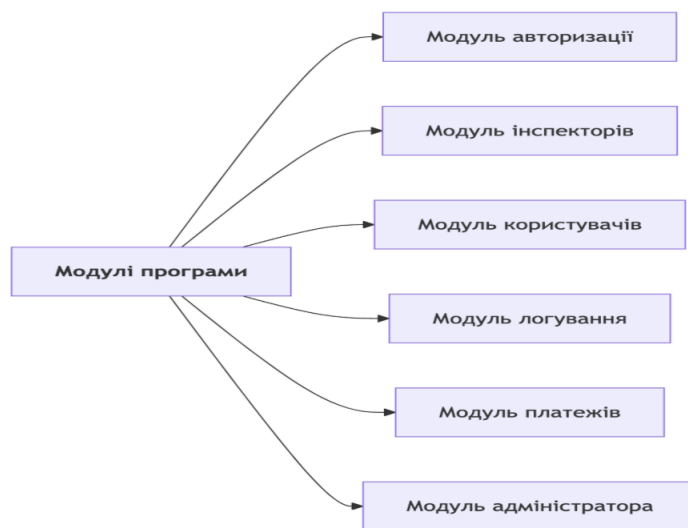
АЛГОРИТМ РОБОТИ ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ



8

Рисунок В.8 – Алгоритм роботи програми паркувальної інспекції

СТРУКТУРНА СХЕМА МОДУЛІВ ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ

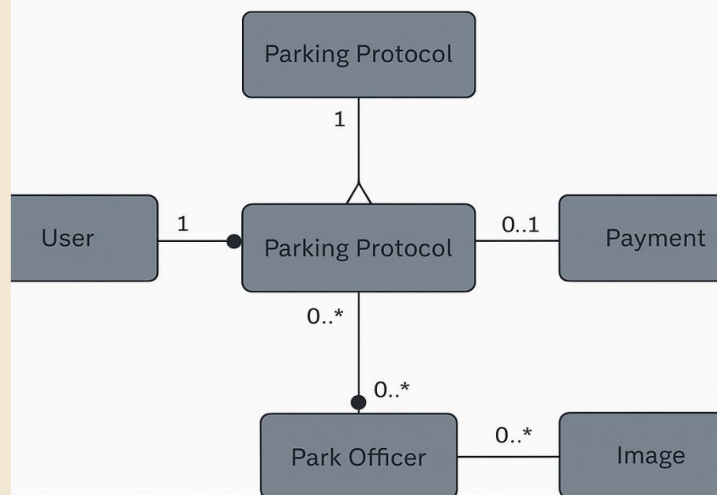


9

Рисунок В.9 – Структурна схема модулів програми для паркувальної інспекції

КОНЦЕПТУАЛЬНА СХЕМА ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ

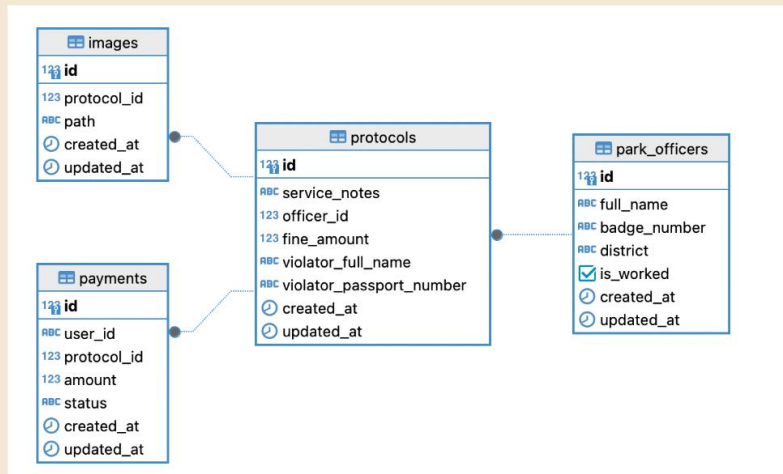
PARKING INSPECTION



10

Рисунок В.10 – Концептуальна схема програми паркувальної інспекції

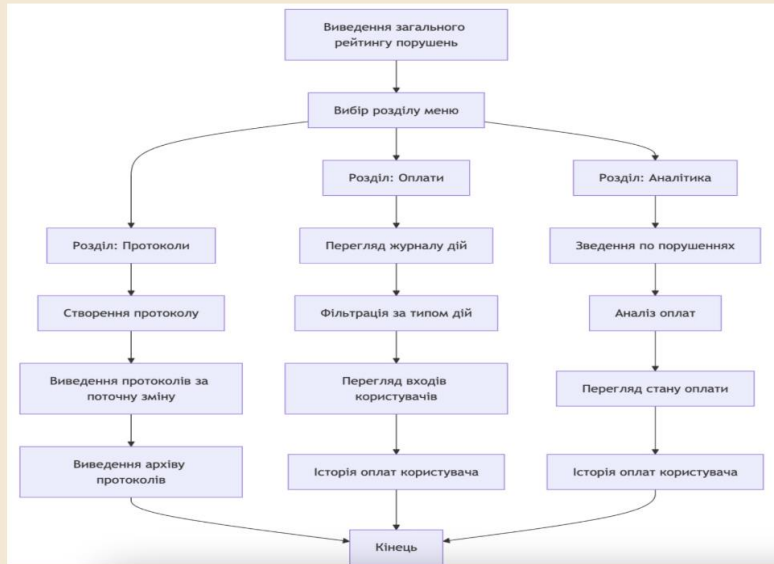
СХЕМА РЕЛЯЦІЙНОЇ БД ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ



11

Рисунок В.11 – Схема реляційної БД програми паркувальної інспекції

ФУНКЦІОНАЛЬНА СХЕМА ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ



12

Рисунок В.12 – Функціональна схема програми паркувальної інспекції

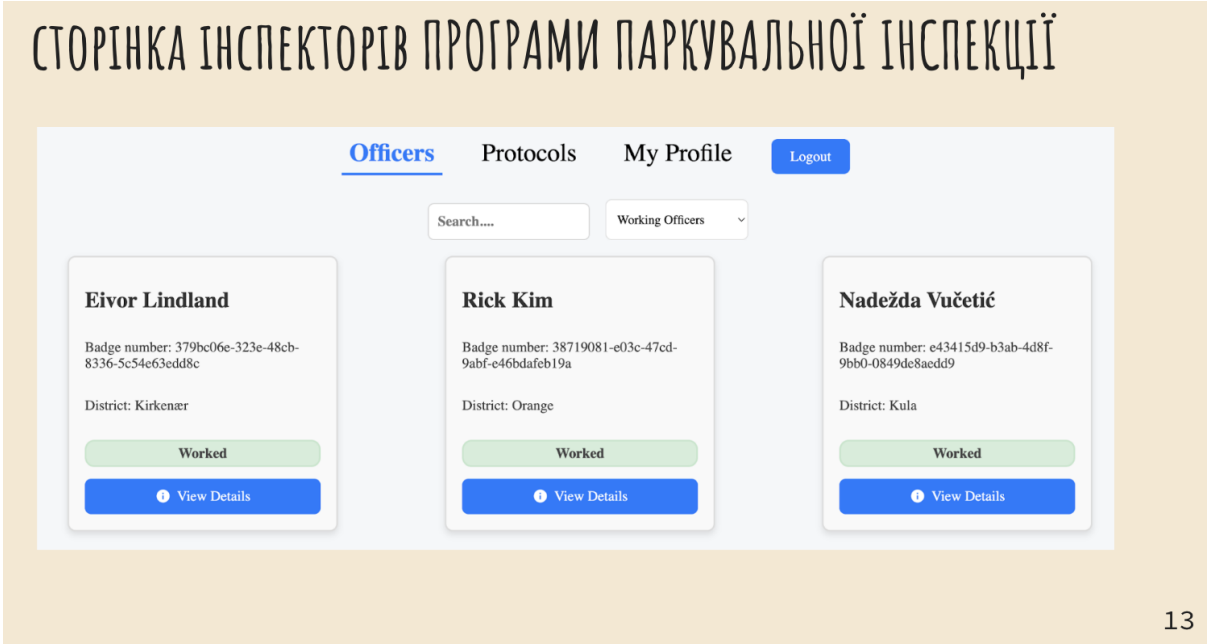


Рисунок В.13 – Сторінка інспекторів програми паркувальної інспекції

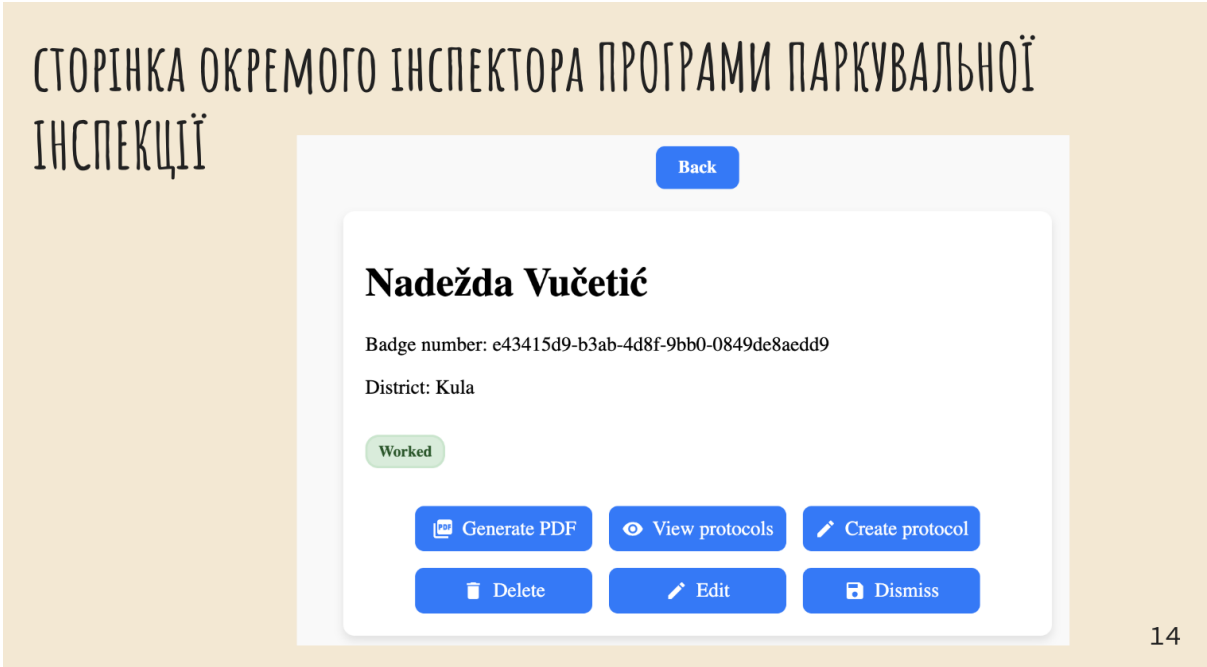


Рисунок В.14 – Сторінка окремого інспектора програми паркувальної інспекції

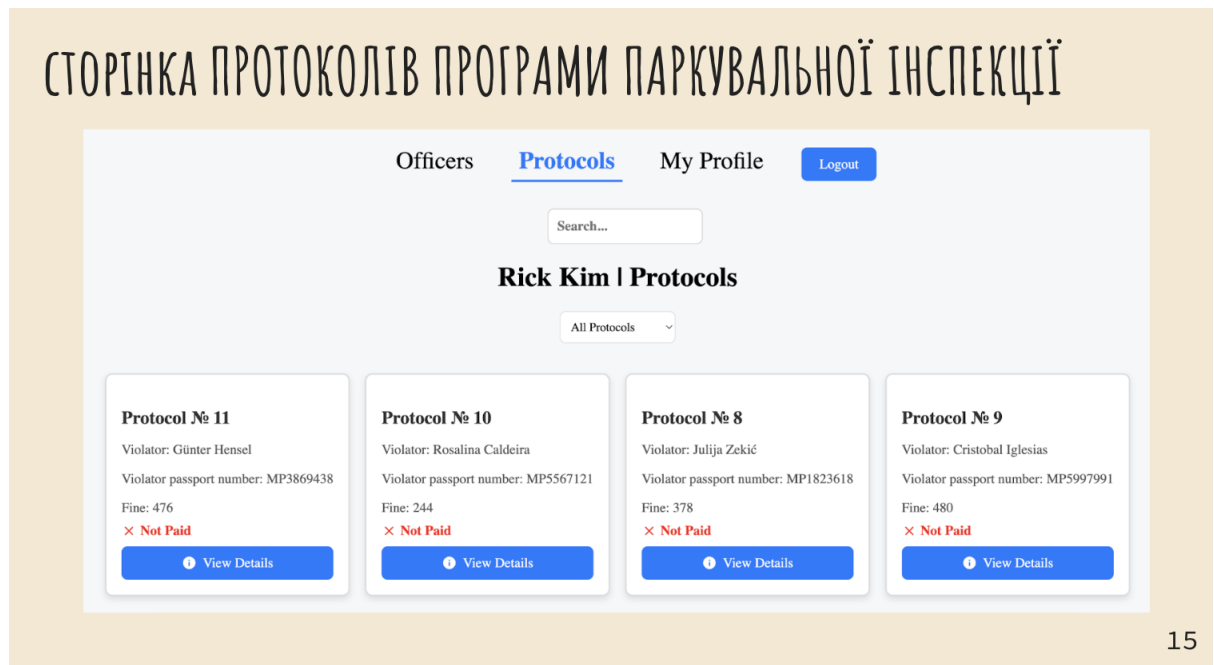


Рисунок В.15 – Сторінка протоколів програми паркувальної інспекції

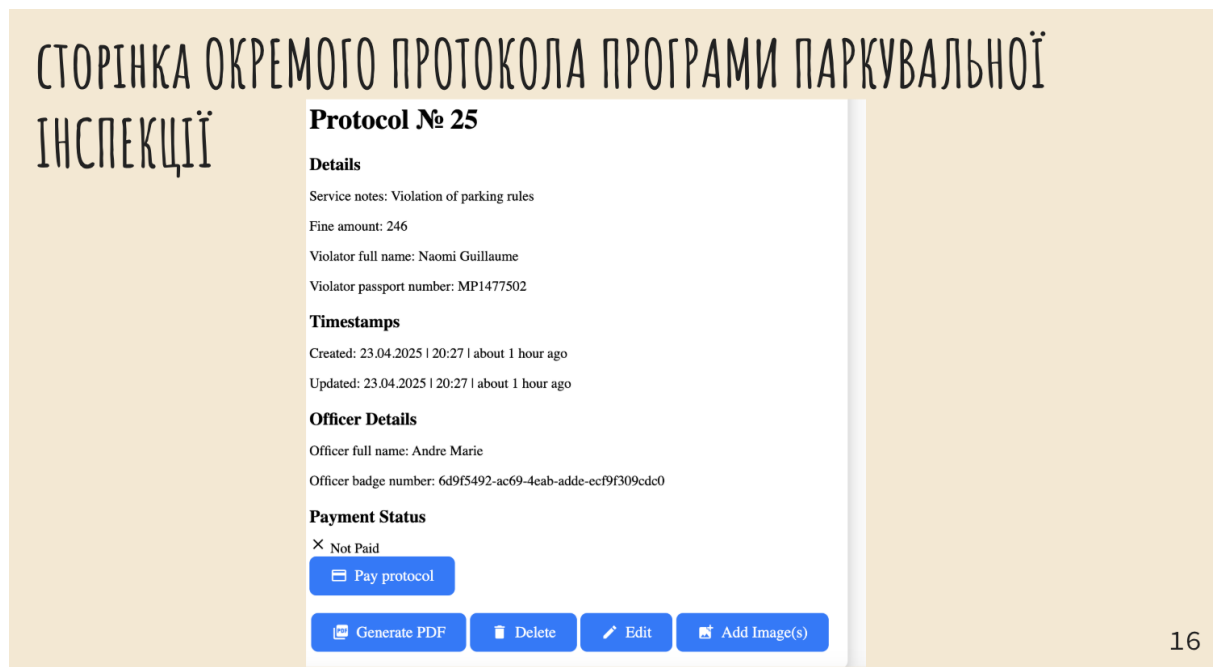
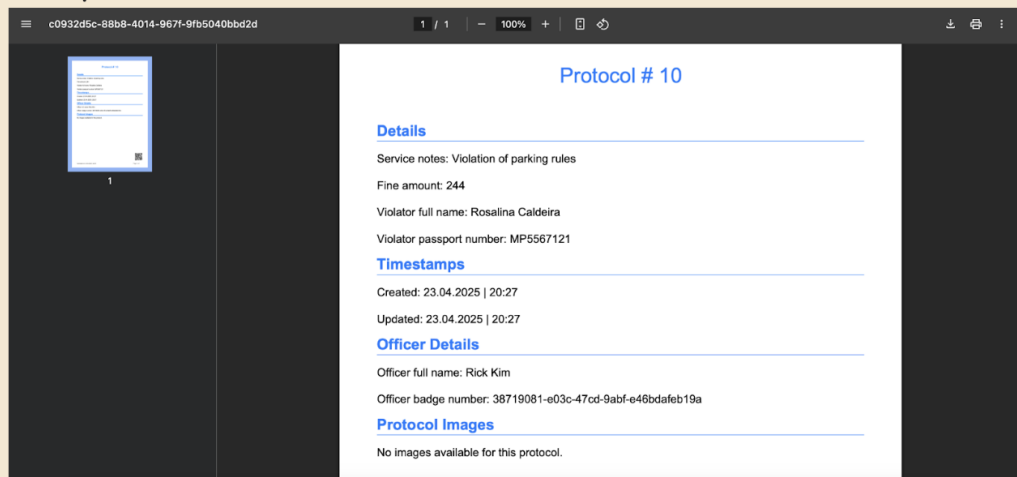


Рисунок В.16 – Сторінка окремого протокола програми паркувальної інспекції

PDF з інформацією про протокол програми паркувальної інспекції



17

Рисунок В.17 – PDF з інформацією про протокол програми паркувальної інспекції

ЛОГИ СЕСІЙ КОРИСТУВАЧА, ЯКІ БАЧИТЬ АДМІНІСТРАТОР ПРОГРАМИ ПАРКУВАЛЬНОЇ ІНСПЕКЦІЇ

[Back](#)

User Sessions

Роман Ратушний | Sessions

Session Created At	IP Address	Provider Info	Geolocation	Geolocation Info	Operating System	Browser	Session Status
23.04.2025 21:16 less than a minute ago	146.120.165.159	Ukraine, Zaporizhzhya, Dianet Ltd.	47.88267000658994 35.07153020297373	Ukraine, Zaporizhia Oblast, Zaporizhzhia, Petra Sahaidachnoho street	Mac OS	Chrome 135	End session
23.04.2025 21:00 16 minutes ago	146.120.165.159	Unknown	Unknown	Unknown	Mac OS	Chrome 135	Revoked
23.04.2025 20:23 about 1 hour ago	146.120.165.159	Unknown	Unknown	Unknown	Mac OS	Chrome 135	Revoked

[Load More](#) [Load All](#)

18

Рисунок В.18 – Логи сесій користувача, які бачить адміністратор програми паркувальної інспекції

ВИСНОВКИ

У цій роботі вдалося досягти головної мети – створити програму, яка автоматизує роботу паркувальної інспекції та допомагає ефективніше контролювати дотримання правил паркування.

Під час аналізу вже наявного програмного забезпечення було з'ясовано, що більшість таких рішень або закриті й складні для оновлення, або не охоплюють усі етапи роботи інспектора – від виявлення порушення до сплати штрафу.

Розроблене програмне забезпечення покликане вирішити ці проблеми.

Рисунок В.19 – Висновки