`МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Мобільний застосунок розпізнавання руху у приміщенні

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Владислав НАЗАРОВ**

«__» _____ 2026 р.

Керівник роботи
ст. викладачка _____ **Світлана БОРОВЛЬОВА**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Назаров Владислав

1. Тема кваліфікаційної роботи Мобільний застосунок розпізнавання руху у приміщенні затверджена наказом ректора ЧНУ ім. Петра Могили №349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «__» _____ 2026 р.
3. Очікуваним результатом роботи є створення клієнт-серверного Android-застосунку, що забезпечує автоматизоване відеоспостереження за приміщенням, детекцію активності в реальному часі та надійне збереження відеодоказів у хмарному сховищі з миттєвим інформуванням власника.
4. Перелік питань, що підлягають розробці: аналіз існуючих рішень для мобільного відеоспостереження та методів програмної детекції руху;

обґрунтування вибору технологічного стеку та програмних засобів; проєктування архітектури системи з використанням Docker-контейнерів для ізоляції серверних компонентів; реалізація алгоритмів обробки відеопотоку на мобільному пристрої для виявлення руху; розробка серверної частини для управління хмарним архівом на базі MinIO та обробки вхідних медіафайлів; створення системи багатоканальних сповіщень через Push-повідомлення та Telegram Bot API; проведення тестування швидкодії системи та оцінка надійності передачі даних при нестабільному інтернет-з'єднанні.

5. Перелік графічних матеріалів: Презентація
6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «26» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Мобільний застосунок розпізнавання руху у приміщенні**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	26.12.2025	26.12.2025	Виконано
2.	Огляд літератури та аналіз існуючих систем відеоспостереження	12.01.2026	19.01.2026	Виконано
3.	Складання календарного плану та технічного завдання	20.01.2026	21.01.2026	Виконано
4.	Аналіз вимог: детекція руху, сценарії сповіщень та логування подій	22.01.2026	26.01.2026	Виконано
5.	Проєктування архітектури системи (Flutter + Node.js) та схеми БД	27.01.2026	12.02.2026	Виконано
6.	Налаштування MinIO в Docker та розробка логіки хмарного архіву	13.02.2026	23.02.2026	Виконано
7.	Реалізація серверної частини: API, обробка подій та Telegram-бот	24.02.2026	16.03.2026	Виконано
8.	Розробка мобільного застосунку: детекція руху та інтерфейс Flutter	17.03.2026	10.04.2026	Виконано
9.	Інтеграційне тестування системи в Docker-контейнерах	11.04.2026	05.05.2026	Виконано
10.	Відгук керівника КБР	06.05.2026	08.05.2026	Виконано
11.	Оформлення КБР та презентації	09.05.2026	24.05.2026	
12.	Попередній захист	27.05.2026	27.05.2026	
13.	Рецензування			
14.	Завершення оформлення КБР та презентації			
15.	Захист кваліфікаційної роботи			

Здобувач _____ **Владислав НАЗАРОВ**

«__» _____ 20__ р.

Керівник роботи
ст. викладачка _____ **Світлана БОРОВЛЬОВА**

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

Мобільний застосунок розпізнавання руху у приміщенні

Здобувач 409 гр.: Назаров Владислав

Керівник: ст. викладачка Боровльова Світлана

Актуальність теми зумовлена необхідністю автоматизованого контролю простору для захисту майна та оперативного реагування на інциденти, що дозволяє виявляти рух саме в момент появи, позбавляючи власника потреби постійного візуального контролю та значно зменшуючи обсяг даних, що передаються мережею.

Метою роботи є розробка мобільного застосунку на базі ОС Android та серверної частини для організації комплексної системи відеоспостереження з функцією детекції руху та віддаленим доступом до архіву записів задля підвищення рівня захисту приміщень.

Об’єктом роботи є процеси автоматизованого відеоспостереження та віддаленого моніторингу приміщень за допомогою мобільних пристроїв.

Предметом роботи є алгоритми детекції руху, методи передачі потокового відео та програмні засоби розробки клієнт-серверних систем для платформи Android.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність обраної теми, визначено мету, основні завдання, об'єкт і предмет роботи, а також розкрито практичне значення отриманих результатів.

У першому розділі проведено аналіз предметної області, визначено ключові процеси відеоспостереження, а також розглянуто та порівняно існуючі програмні рішення, виявлено їхні переваги та недоліки.

Другий розділ присвячено дослідженню сучасних методів передавання відеопотоків (RTSP, HLS, WebRTC), алгоритмів детекції руху, механізмів

автентифікації та архітектурних підходів. На основі аналізу сформовано специфікацію вимог до програмного забезпечення.

У третьому розділі спроєктовано архітектуру системи: гібридний клієнт-серверний підхід для серверної частини, Clean Architecture та BLoC для мобільного застосунку, контейнеризацію Docker. Побудовано UML-діаграми (варіантів використання, діяльності, послідовності, класів), розроблено графічний інтерфейс та обґрунтовано вибір технологічного стеку.

Четвертий розділ містить опис програмної реалізації: налаштування середовища розробки, реалізацію ключових модулів (автентифікація з оновленням токенів, WebRTC-зв'язок, детекція руху, завантаження відео до MinIO), тестування системи та оцінку її стійкості до збоїв мережі.

У висновках узагальнено результати виконаної роботи, підтверджено досягнення поставленої мети та накреслено перспективи подальшої модернізації функціоналу розробленої системи.

КРБ викладена на 73 сторінок, вона містить 4 розділи, 16 ілюстрацій, 4 таблиць, 15 джерел в переліку посилань.

Ключові слова: відеоспостереження, детекція руху, Flutter, Node.js, PostgreSQL, MinIO, Docker-контейнеризація.

ABSTRACT

to the qualifying bachelor's thesis

Mobile application for indoor motion recognition

Student of 409 group: Nazarov Vladyslav

Supervisor: Senior Lecturer Borovlova Svitlana

The relevance of the topic is driven by the need for automated space control to protect property and respond promptly to incidents, which allows detecting motion at the moment of its occurrence, eliminating the need for constant visual monitoring and significantly reducing the amount of data transmitted over the network.

The goal of the work is to develop a mobile application based on the Android OS and a server part to organize a comprehensive video surveillance system with a motion detection function and remote access to the record archive to enhance the level of room security.

The object of the work is the processes of automated video surveillance and remote monitoring of premises using mobile devices.

The subject of the work includes motion detection algorithms, streaming video transmission methods, and software development tools for client-server systems on the Android platform.

The qualification work consists of an introduction, __ sections, conclusions and a list of references.

The introduction substantiates the relevance of the chosen topic, defines the goal, main tasks, object and subject of the work, and reveals the practical significance of the obtained results.

The first chapter analyzes the subject area, identifies the key processes of video surveillance, and reviews and compares existing software solutions, identifying their advantages and shortcomings.

The second chapter is devoted to the study of modern methods of video streaming (RTSP, HLS, WebRTC), motion detection algorithms, authentication mechanisms and architectural approaches. Based on the analysis, a software requirements specification is formed.

The third chapter designs the system architecture: a hybrid client-server approach for the server part, Clean Architecture and BLoC for the mobile application, and Docker containerization. UML diagrams (use case, activity, sequence, class) are built, the graphical user interface is developed, and the technology stack is justified.

The fourth chapter describes the software implementation: development environment setup, implementation of key modules (authentication with token refresh, WebRTC communication, motion detection, video upload to MinIO), system testing and assessment of its resilience to network failures.

The conclusions summarize the results of the work, confirm the achievement of the set goal, and outline the prospects for further modernization of the developed system's functionality.

The bachelor's qualification work is presented on 73 pages, it contains 4 chapters, 16 illustrations, 4 tables, and 15 sources in the list of references.

Keywords: video surveillance, motion detection, Flutter, Node.js, PostgreSQL, MinIO, Docker containerization.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ... 3
ВСТУП ... 4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ... 6
1.1 Системний аналіз об’єкту та предмету роботи ... 6
1.2 Аналіз особливостей об’єкта роботи ... 8
1.3 Огляд сучасних програмних рішень ... 10
Висновки до розділу 1 ... 15
2 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ ... 16
2.1 Аналіз методів реалізації ключових компонентів системи ... 16
2.2 Специфікація вимог до програмного забезпечення ... 22
Висновки до розділу 2 ... 32
3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ ... 34
3.1 Обґрунтування вибору архітектурних рішень та вибір стеку технологій ... 34
3.2 Діаграма варіантів використання ... 36
3.3 Діаграма послідовності для процесу автентифікації з оновленням токенів ... 39
3.4 Діаграма послідовності для встановлення WebRTC-з’єднання ... 42
3.5 Діаграма класів ... 44
3.6 Проєктування бази даних ... 46
3.7 Проєктування графічного інтерфейсу користувача ... 48
Висновки до розділу 3 ... 51
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ... 53
4.1 Програмна реалізація клієнтської частини ... 53
4.2 Програмна реалізація серверної частини ... 64
4.3 Тестування та налагодження системи ... 68
Висновки до розділу 4 ... 74
ВИСНОВКИ ... 75
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ... 77

ПЕРЕЛІК СКОРОЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс)

DTLS-SRTP – Datagram Transport Layer Security – Secure Real-time Transport Protocol (протокол безпечної передачі медіаданих)

HLS – HTTP Live Streaming (протокол потокової передачі відео через HTTP)

ICE – Interactive Connectivity Establishment (протокол встановлення з’єднання)

JWT – JSON Web Token (вебтокен у форматі JSON)

NAT – Network Address Translation (трансляція мережевих адрес)

RBAC – Role-Based Access Control (керування доступом на основі ролей)

REST – Representational State Transfer (стиль архітектури API)

RTSP – Real-Time Streaming Protocol (протокол потокової передачі в реальному часі)

RTP – Real-time Transport Protocol (транспортний протокол реального часу)

SDP – Session Description Protocol (протокол опису сеансу зв’язку)

SRS – Software Requirements Specification (специфікація вимог до ПЗ)

STUN – Session Traversal Utilities for NAT (утиліта обходу NAT)

TURN – Traversal Using Relays around NAT (ретрансляція трафіку для обходу NAT)

WebRTC – Web Real-Time Communication (технологія прямої передачі аудіо/відео)

WSS – WebSocket Secure (захищений протокол WebSocket)

ВСТУП

Сьогодні питання безпеки житла та офісів стає все більш гострим, проте професійні системи відеоспостереження часто вимагають значних фінансових витрат та складного монтажу. Водночас у багатьох користувачів залишаються старі смартфони, які мають достатні технічні характеристики для виконання функцій камери. Використання мобільних пристроїв як камер спостереження є доступним рішенням, яке дозволяє швидко організувати моніторинг приміщення без закупівлі спеціалізованого обладнання. Крім того, відкриті програмні платформи дають змогу створити систему, що не залежить від хмарних сервісів виробників і забезпечує повний контроль над даними користувача.

Актуальність теми зумовлена необхідністю автоматизованого контролю простору для захисту майна та оперативного реагування на інциденти. Відстеження руху є ключовим процесом, оскільки воно дозволяє ідентифікувати несанкціоноване проникнення або непередбачувані події в приміщенні саме в момент їх виникнення. Це позбавляє власника потреби постійного візуального контролю та перегляду багатогодинних статичних записів. Система активує фіксацію лише при виявленні реальної активності, що критично важливо для стабільної роботи: такий підхід суттєво зменшує обсяг даних, що передаються мережею, та економить місце у хмарному сховищі. Таким чином, розробка ефективного та доступного рішення для відеоспостереження з функцією детекції руху є нагальною потребою сучасного ринку.

Метою роботи є розробка мобільного застосунку на базі ОС Android та серверної частини для організації комплексної системи відеоспостереження з функцією детекції руху та віддаленим доступом до архіву записів задля підвищення рівня захисту приміщень.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1) провести аналіз предметної області та існуючих програмних аналогів;

2) сформувати технічні вимоги до програмного комплексу та спроєктувати взаємодію між мобільним клієнтом і сервером;

3) розробити архітектуру бази даних для збереження медіафайлів та логів активності;

4) вивчити методи обробки відеопотоку та алгоритми розпізнавання руху на мобільних пристроях;

5) реалізувати функціонал трансляції відео та надсилання push-повідомлень;

6) налаштувати інтеграцію з месенджером Telegram для створення резервного каналу сповіщень;

7) провести тестування роботи застосунку в різних умовах освітлення та оцінити швидкість реакції системи на рух.

Об’єктом роботи є процеси автоматизованого відеоспостереження та віддаленого моніторингу приміщень за допомогою мобільних пристроїв.

Предметом роботи є алгоритми детекції руху, методи передачі потокового відео та програмні засоби розробки клієнт-серверних систем для платформи Android.

Практичне значення полягає у створенні бюджетної та гнучкої системи безпеки. Рішення не вимагає придбання дорогого спеціалізованого обладнання, а використовує наявні Android-смартфони, які часто просто лежать без діла. Користувач отримує можливість перетворити будь-який Android-смартфон на повноцінну камеру відеоспостереження з можливістю перегляду записів через хмару та дублюванням критичних сповіщень у месенджер, що робить систему стійкою до збоїв основних каналів передачі даних. Завдяки контейнеризації серверної частини за допомогою Docker, користувач може самостійно розгорнути систему на власному обладнанні без залежності від зовнішніх хмарних провайдерів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Розробка ефективної системи відеоспостереження неможлива без попереднього аналізу предметної області, який дає змогу зрозуміти особливості функціонування об’єкта, визначити ключові процеси, що потребують автоматизації, та критично оцінити вже наявні на ринку рішення. У цьому розділі послідовно розглянуто загальні характеристики сфери відеоспостереження на базі мобільних пристроїв, конкретизовано вимоги до програмно-апаратного комплексу та проведено порівняльний огляд існуючих програмних продуктів.

1.1 Системний аналіз об’єкту та предмету роботи

Сучасне суспільство все гостріше потребує доступних, гнучких і водночас надійних засобів віддаленого відеоспостереження. Поштовхом слугують як зростання рівня злочинності в міських та сільських районах, так і прагнення контролювати власну оселю, офіс, склад або прибудинкову територію незалежно від фізичної присутності людини. Традиційні системи відеоспостереження, що складаються з набору IP-камер, відеореєстратора та монітора, хоча й забезпечують високу якість зображення, часто залишаються фінансово недосяжними для широкого загалу, а їх монтаж потребує специфічних технічних знань. Водночас стрімкий розвиток мобільних технологій призвів до накопичення великої кількості застарілих, але цілком функціональних смартфонів, які мають у своєму розпорядженні потужні камери, обчислювальні ресурси та комунікаційні модулі. Постає закономірне питання: чи можливо перетворити такий пристрій на повноцінний елемент охоронної системи, здатний не лише передавати відео, а й самостійно виявляти загрози?

У центрі цього проєкту знаходяться процеси, які перетворюють звичайний телефон на розумну камеру спостереження. Це насамперед безперервне захоплення відеоданих з оптичного сенсора, їх перетворення в цифровий формат та стиснення сучасними кодеками для зменшення навантаження на мережу. Однак проста трансляція відео не є достатньою для вирішення завдань безпеки – власник не може

постійно перебувати біля екрана, очікуючи на появу порушника. Тому ключового значення набуває здатність системи автоматично аналізувати відеопотік, виявляючи моменти появи руху в контрольованій зоні. Саме алгоритми порівняння послідовних кадрів, що фіксують зміни яскравості пікселів та перевищення заданого порогу, перетворюють пасивний пристрій на активний інструмент охорони. Сучасні дослідження підтверджують, що для периметрального відеоспостереження критично важливою є швидка та надійна детекція вторгнень [1].

Не менш важливим аспектом є організація передавання відеопотоку до віддаленого користувача. Тут на перший план виходять методи потокової трансляції в реальному часі, зокрема технологія WebRTC [2], яка дозволяє встановлювати пряме peer-to-peer з'єднання з мінімальною затримкою [3], а за необхідності – маршрутизувати трафік через проміжний сервер. В умовах, коли йдеться про безпеку, критично важливим є не лише швидкість доставки відео, а й його захищеність: усі дані, що циркулюють між камерою та глядачем, повинні бути надійно зашифровані, щоб унеможливити перехоплення або підміну.

Крім того, досліджувана область охоплює питання збереження записів. Виявлення руху має супроводжуватись автоматичним запуском запису відеофрагмента, який повинен бути збережений навіть за умови тимчасової відсутності інтернет-з'єднання. Це досягається завдяки локальному кешуванню на самому смартфоні з подальшим завантаженням до хмарного сховища після відновлення мережі. Такий підхід гарантує, що докази не будуть втрачені через нестабільність каналу зв'язку.

Окрему увагу приділено проблемі розмежування доступу. На практиці до однієї камери часто мають потребу підключатися різні особи: власник, члени родини, тимчасовий персонал. Кожен із них повинен мати чітко визначений набір дозволів – від простого перегляду відео до можливості змінювати налаштування сигналізації або переглядати архів. Відсутність такого гнучкого механізму в більшості існуючих продуктів створює як незручності, так і потенційні ризики для безпеки.

Таким чином, досліджувана предметна область являє собою складний комплекс процесів, що включають захоплення відео, його аналіз на предмет руху, захищену передачу, збереження записів та управління доступом. Саме ці процеси в сукупності визначають ефективність системи відеоспостереження на базі мобільних пристроїв, а методи їх реалізації – алгоритми детекції, протоколи потокової передачі, засоби шифрування та архітектурні рішення – становлять той технологічний фундамент, на якому будується уся подальша розробка.

1.2 Аналіз особливостей об’єкта роботи

Аналіз предметної області дозволяє окреслити ключові характеристики, яким має відповідати програмно-апаратний комплекс, що перетворює смартфон на інструмент охорони. Ці характеристики випливають безпосередньо з вимог фізичної безпеки приміщень і водночас враховують обмеження, притаманні мобільним пристроям.

Передусім система повинна забезпечувати безперервне захоплення та передавання відео. Смартфон, виступаючи в ролі камери, має стабільно отримувати зображення з оптичного сенсора, обробляти його для зменшення обсягу даних і передавати віддаленому користувачеві з мінімальною затримкою. Це вимагає ефективних алгоритмів кодування, здатних адаптуватися до змінної пропускної здатності мережі – чи то домашній Wi-Fi, чи мобільний інтернет. Крім того, передавання має бути стійким до розривів: після тимчасової втрати зв’язку система повинна автоматично відновлювати сеанс без втручання людини.

Другою принциповою особливістю є інтелектуальна детекція руху. Пасивна трансляція відео не вирішує завдань безпеки, оскільки власник фізично не здатен цілодобово спостерігати за екраном. Тому на самому пристрої-камері має працювати механізм аналізу послідовних кадрів, який виявляє суттєві зміни в сцені – появу людини, транспортного засобу або іншого об’єкта. Важливо, щоб цей аналіз відбувався безперервно у фоновому режимі, не перевантажуючи процесор і не викликаючи надмірного нагрівання телефону. Виявлення руху повинно супроводжуватися миттєвим запуском запису відео та негайним сповіщенням

користувача, причому сповіщення мають дублюватися через незалежний канал зв’язку, щоб гарантовано досягти адресата навіть за збоїв основного.

Третя невід’ємна риса – надійне збереження записів. Відеофрагменти, створені під час тривожних подій, є доказовою базою, втрата якої неприпустима. Оскільки мобільний пристрій може працювати в умовах нестабільного інтернет-з’єднання, система зобов’язана передбачити дворівневу архівацію: спершу відео зберігається локально на самому смартфоні, а потім, за першої ж можливості, автоматично передається до віддаленого сховища. Лише після підтвердження успішного завантаження локальна копія може бути видалена. Такий підхід гарантує збереження критично важливих даних навіть за тривалої відсутності мережі.

Четвертим ключовим аспектом є гнучке управління доступом. На відміну від найпростіших систем, де всі користувачі мають однакові права, охоронна система повинна підтримувати розмежування повноважень. Власник камери має бути спроможний точно визначити, що дозволено кожному запрошеному користувачеві: чи може він лише переглядати відео в реальному часі, чи також гортати архів, чи вмикати мікрофон, чи змінювати режим сигналізації тощо. Така деталізація не лише підвищує зручність спільного користування, а й безпосередньо впливає на безпеку, унеможливлюючи випадкове або зловмисне втручання в роботу охоронних функцій. Усі дії користувачів повинні фіксуватися в захищеному журналі, доступному лише власнику, що забезпечує прозорість і можливість розслідування інцидентів.

П’ятою важливою вимогою є захищеність усіх каналів зв’язку. Оскільки через мережу передаються як відеодані, що можуть містити конфіденційну інформацію, так і команди керування камерою, будь-яке перехоплення або підміна цих даних є неприпустимими. Тому обмін між камерою, сервером і глядачем повинен здійснюватися виключно через шифровані протоколи, що унеможливлюють несанкціонований доступ до інформації на всіх етапах її життєвого циклу.

Нарешті, з точки зору розгортання, система повинна бути автономною та незалежною від зовнішніх хмарних сервісів. Це означає, що вся серверна

інфраструктура – обробка запитів, зберігання архіву, керування доступом – має працювати на обладнанні, яке повністю контролюється користувачем. Такий підхід не лише усуває ризики, пов’язані з передачею даних третім сторонам, а й гарантує працездатність системи незалежно від змін тарифної політики чи припинення підтримки з боку зовнішніх постачальників послуг. При цьому процедура встановлення та оновлення має бути максимально простою, щоб не вимагати від користувача глибоких технічних знань.

Таким чином, функціональний портрет системи окреслюється набором взаємопов’язаних властивостей: автономна робота в режимі реального часу, локальний аналіз сцени, відмовостійка архівація, гранульований доступ і наскрізне шифрування. Саме ці особливості визначають архітектурні рішення, що будуть детально розглянуті в наступних розділах.

1.3 Огляд сучасних програмних рішень

Для визначення місця проєктованої системи серед існуючих продуктів було проаналізовано найбільш поширені програмні засоби, що дозволяють перетворити мобільний пристрій на камеру спостереження, а також популярні комерційні хмарні сервіси безпеки.

Першу групу становлять мобільні застосунки-«камери», такі як AlfredCamera [4], AtHome Camera та IP Webcam. Усі вони пропонують подібну базову функціональність – встановлення на старому смартфоні, захоплення відео з камери та його передавання на інший пристрій через хмарний сервер розробника. Однак детальний аналіз виявляє низку суттєвих обмежень, які ставлять під сумнів придатність цих рішень для серйозних завдань безпеки.

По-перше, у більшості безкоштовних версій відсутнє наскрізне шифрування даних – відеопотік передається через сервери виробника без гарантій захисту від перехоплення. По-друге, архів записів або відсутній, або доступний лише за платною підпискою, причому зберігається на хмарі, що позбавляє користувача контролю над власними даними. По-третє, жоден із цих застосунків не надає механізму для гнучкого розмежування прав доступу: гість отримує або повний

перегляд, або нічого, без можливості обмежити окремі дії. Окрім того, всі вони залежать від хмарної інфраструктури виробника, що унеможливлює автономну роботу системи без стороннього втручання.

AlfredCamera є найпопулярнішим представником цієї категорії завдяки простоті запуску: реєстрація триває кілька хвилин, а базовий перегляд відео та сповіщення про рух надаються безкоштовно. Водночас саме ця простота приховує архітектурні вади. Відеопотік та службові команди циркулюють через закриту хмару розробника, причому публічно не підтверджено застосування наскрізного шифрування, що робить дані потенційно вразливими. Безкоштовний архів має обмеження за тривалістю та якістю, а за відсутності інтернет-з’єднання запис не виконується взагалі – критичні моменти можуть бути втрачені. Крім того, система запрошень надає лише загальний доступ до камери без можливості вибірково заборонити окремі функції, як-от увімкнення мікрофона чи видалення записів. Отже, AlfredCamera задовольняє лише невибагливі побутові сценарії, але не відповідає вимогам повноцінної охоронної системи.

Другу категорію утворюють комерційні хмарні системи безпеки, такі як Google Nest [5], Ring та Arlo. Вони пропонують високу якість зображення, інтелектуальну аналітику та зручні мобільні додатки. Проте їх використання вимагає придбання фірмового обладнання, вартість якого в декілька разів перевищує бюджет навіть нового бюджетного смартфона. Архітектура таких систем повністю замкнена на хмарних серверах виробника: відеозаписи зберігаються виключно на віддалених потужностях, доступ до яких може бути припинений у будь-який момент через зміну тарифів або політики компанії. Механізми надання доступу іншим користувачам реалізовані за спрощеною моделлю – як правило, це єдина опція «спільний доступ», що не дозволяє диференціювати права. Крім того, ці системи не передбачають використання старих смартфонів як камер, що нівелює ідею бюджетної охорони.

Google Nest репрезентує цей сегмент найповніше. Камери Nest Cam підтримують роздільну здатність аж до 4K, а вбудований штучний інтелект

розрізняє людей, тварин і транспорт, зменшуючи кількість хибних тривог. Уся інфраструктура керується автоматично, оновлення надходять без участі власника.

Проте стартовий комплект коштує сотні доларів, а для безперервного запису потрібна платна підписка Nest Aware. Усі дані зберігаються на серверах Google: користувач не має змоги перенести архів на власний носій або продовжити користуватися накопиченими записами після припинення підписки. Спільний доступ реалізовано через акаунт Google, що дає іншій особі практично ті самі права, що й власнику, без тонкого гранулювання. Таким чином, Google Nest – це потужне, але дороге та інфраструктурно залежне рішення, яке не вписується в концепцію доступної автономної безпеки.

Окремо варто згадати відкриті серверні платформи на кшталт ZoneMinder [6] та Shinobi. Вони надають змогу розгорнути відеосервер на власному обладнанні, що вирішує проблему контролю за даними. Однак їх встановлення та налаштування потребують глибоких знань адміністрування операційних систем і мережевих протоколів, що робить їх недосяжними для пересічного користувача. Попри широкі можливості відеоаналітики, вони не пропонують вбудованого механізму гранульованих дозволів для різних категорій користувачів, а клієнтський інтерфейс часто є веб-сторінкою, не оптимізованою для мобільних пристроїв.

ZoneMinder – один із найстаріших і найфункціональніших представників цієї категорії. Він підтримує підключення практично будь-яких IP-камер, зокрема смартфонів через RTSP-потік, дозволяє гнучко налаштовувати зони детекції та правила запису, а також надсилати сповіщення електронною поштою.

Система повністю контролюється власником і не залежить від зовнішніх сервісів. Однак її розгортання вимагає володіння Linux, налаштування вебсервера, бази даних та мережевих портів – знань, яких пересічний користувач не має, а будь-який спільний доступ фактично надає адміністративні права, що унеможливлює безпечне спільне користування однією камерою кількома особами з різними ролями.

Проведений аналіз дає змогу чітко сформулювати системні недоліки, притаманні існуючим продуктам, які має виправити нова розробка:

1) відсутність належного рівня безпеки даних. Переважна більшість безкоштовних застосунків нехтує шифруванням відеопотоку, наражаючи користувача на ризик перехоплення конфіденційної інформації. Комерційні хмарні сервіси, хоч і використовують захищені протоколи, повністю контролюють інфраструктуру зберігання, що позбавляє власника гарантій недоторканності записів. Пропонована система усуває цю проблему завдяки наскрізному шифруванню на всіх етапах обміну та можливості розгортання на власному сервері, що забезпечує абсолютний контроль над даними;

2) відсутність гнучкого управління доступом. Жоден із розглянутих продуктів не надає інструментів для незалежного дозволу або заборони окремих операцій із камерою. Це унеможливлює безпечне спільне користування системою різними категоріями осіб із чітко визначеними ролями. Запровадження шести незалежних прапорців дозволів у новій системі дозволить власнику точно налаштовувати рівень доступу для кожного користувача, мінімізуючи ризики внутрішніх порушень;

3) ненадійність в умовах нестабільного зв’язку. Більшість існуючих рішень не мають механізмів локального кешування записів із подальшим автоматичним завантаженням після відновлення мережі, що може призвести до безповоротної втрати критичних відеофрагментів. У новій системі передбачено дворівневу архівацію, яка гарантує збереження даних незалежно від якості інтернет-з’єднання;

4) залежність від зовнішніх хмарних сервісів. Усі популярні безкоштовні та комерційні рішення прив’язані до інфраструктури виробника, що створює ризики раптової зміни умов надання послуг, підвищення тарифів або повного припинення підтримки. На відміну від них, нова система постачатиметься у вигляді контейнеризованого відкритого програмного забезпечення, яке можна запустити на будь-якому звичайному комп’ютері чи недорогому сервері. Відкритий код дозволяє аудит безпеки сторонніми фахівцями, що повністю виключає наявність

прихованих уразливостей, і дає змогу спільноті розширювати функціонал незалежно від волі одного розробника. Це забезпечує незалежність, гнучкість у налаштуванні політик зберігання та можливість автономної роботи без стороннього втручання.

Для наочного узагальнення результатів порівняльного аналізу основні характеристики розглянутих рішень зведено в таблицю 1.1. Оцінювання проводилося за дев'ятьма функціональними та безпековими критеріями, які найбільше впливають на ефективність охоронної системи. Отримані дані свідчать, що жоден із наявних продуктів не забезпечує одночасно бюджетного використання старих смартфонів, наскрізного шифрування та гнучкого керування доступом, що й визначає конкурентну перевагу пропонованої розробки.

Таблиця 1.1 – Порівняльний аналіз характеристик існуючих систем

Характеристика	**AlfredCamera / AtHome / IP Webcam**	**Google Nest / Ring / Arlo**	**ZoneMinder / Shinobi**	**Пропонована система**
Потокове відео в реальному часі	Так	Так	Так	Так
Детекція руху	Базова	Так	Так	Так (з налаштуванням чутливості)
Локальний архів без підписки	Відсутній або платний	Платний	Так	Так
Шифрування даних при передачі	Не завжди	Так	Залежить від налаштувань	Так (наскрізне)
Контроль за даними (власний сервер)	Ні	Ні	Так	Так
Гранульовані дозволи	Відсутні	Обмежені	Відсутні	Так (6 незалежних прапорців)
Резервний канал сповіщень	Ні	Ні	Лише email	Так (Telegram)
Використання старих смартфонів як камер	Так	Ні (фірмове обладнання)	Так (складне налаштування)	Так (автоматична ідентифікація)
Простота розгортання	Висока (хмара)	Висока (закрита)	Низька	Середня (Docker, інструкції)
Відкритий код	Ні	Ні	Так	Так

Таким чином, саме поєднання наскрізного захисту даних, гнучкої моделі розмежування доступу, відмовостійкого механізму архівації та повної

автономності від зовнішніх хмарних сервісів визначає унікальність запропонованої системи та обґрунтовує необхідність її розробки. Виявлені прогалини в існуючих продуктах формують перелік вимог, які будуть детально специфіковані в другому розділі роботи.

Висновки до розділу 1

У результаті системного аналізу предметної області встановлено, що процеси віддаленого відеоспостереження за допомогою мобільних пристроїв є багатокомпонентною задачею, яка охоплює захоплення відео, його аналіз на предмет руху, захищену передачу, надійне зберігання записів та гнучке управління доступом. Ефективність системи безпеки визначається не окремими функціями, а їх узгодженою взаємодією, що перетворює пасивний смартфон на активний інструмент охорони приміщень.

Проведений огляд існуючих програмних рішень виявив низку критичних недоліків, які не дозволяють цим продуктам повноцінно виконувати охоронні функції. До них належать відсутність наскрізного шифрування, що наражає відеопотік на ризик перехоплення; втрата контролю за даними через прив’язку до хмар виробника; неможливість гнучко розмежовувати права доступу між користувачами; відсутність механізмів локального кешування записів при нестабільному інтернет-з’єднанні; а також висока вартість або надмірна технічна складність розгортання.

Обґрунтовано, що розроблювана система усуває зазначені недоліки завдяки поєднанню наскрізного захисту даних, гранульованої моделі дозволів, відмовостійкої дворівневої архівації та можливості повністю автономного розгортання на власному обладнанні користувача. Виявлені прогалини в існуючих продуктах формують об’єктивну потребу в новій розробці, а сформульовані функціональні особливості створюють підґрунтя для специфікації вимог до програмного забезпечення, що буде виконано в другому розділі.

2 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Для успішної реалізації системи необхідно розглянути сучасні методи та засоби, що можуть бути застосовані для передавання відео, виявлення руху, захисту даних і зберігання інформації. Такий аналіз дасть змогу обрати найбільш відповідні технології та на їх основі сформувати специфікацію вимог до програмного забезпечення.

2.1 Аналіз методів реалізації ключових компонентів системи

Для побудови ефективної системи відеоспостереження критично важливо обрати відповідні методи реалізації її основних компонентів, які безпосередньо впливають на продуктивність, безпеку та зручність використання. У цьому підрозділі послідовно проаналізовано підходи до передавання відеопотоку в реальному часі, алгоритми детекції руху на мобільних пристроях, механізми автентифікації та авторизації, способи зберігання відеоданих і архітектурні рішення для клієнт-серверних систем. Такий аналіз дозволить надалі обґрунтовано вибрати технологічний стек і сформулювати вимоги до програмного забезпечення.

2.1.1 Методи передавання відеопотоку в реальному часі

Передавання відео з мобільного пристрою на віддалений клієнт є центральною функцією системи. У сучасних розподілених застосунках для цього застосовуються різні протоколи та технології, що відрізняються за затримкою, стійкістю до втрат пакетів, підтримкою NAT-traversal та вимогами до серверної інфраструктури.

RTSP/RTP – класичний протокол реального часу для потокового відео. Забезпечує низьку затримку, однак часто потребує відкритих портів на маршрутизаторі або додаткового проксі-сервера для обходу NAT, що ускладнює використання в домашніх мережах без статичної IP-адреси.

HLS – адаптивний протокол, який передає відео невеликими сегментами через звичайний HTTP. Він добре масштабується, сумісний із будь-якими мережами, проте має відносно високу затримку (5–15 секунд), що для охоронної системи є критичним недоліком.

WebRTC – набір протоколів і API, спеціально розроблений для прямої peer-to-peer передачі аудіо та відео з мінімальною затримкою (менше 1 секунди). Підтримує автоматичне визначення мережевої топології (ICE, STUN, TURN), шифрування медіапотоків (DTLS-SRTP) і не потребує встановлення додаткових плагінів. Це робить його найпривабливішим варіантом для системи, де важливі швидкість реакції та безпека. Однак розгортання WebRTC вимагає наявності сигнального сервера та, за потреби, TURN-сервера для ретрансляції трафіку, що дещо ускладнює серверну інфраструктуру.

Таблиця 2.1 – Порівняння методів передавання відеопотоку

Критерій	**RTSP/RTP**	**HLS**	**WebRTC**
Затримка	Низька (1–3 с)	Висока (5–15 с)	Дуже низька (<1 с)
Підтримка NAT-traversal	Обмежена	Так (через HTTP)	Вбудована (ICE/STUN/TURN)
Шифрування	Залежить від реалізації	HTTPS	DTLS-SRTP
Складність серверної частини	Середня	Низька	Висока (сигнальний сервер)
Адаптивність до мережі	Відсутня	Вбудована	Часткова (Simulcast)

Отже, кожен із розглянутих протоколів має свої сильні сторони та обмеження. RTSP/RTP забезпечує низьку затримку, але створює труднощі під час роботи через NAT, характерні для домашніх мереж. HLS пропонує універсальність і сумісність, однак його затримка є неприйнятною для системи, що має миттєво реагувати на загрози. WebRTC вирізняється вбудованими засобами безпеки та мінімальною затримкою, хоча потребує складнішої серверної підтримки. Систематичний огляд [3] підтверджує, що можливості WebRTC виходять далеко за межі звичайних аудіо/відео дзвінків і охоплюють сценарії реального часу для безпеки. Більше того, існують гібридні підходи, які дозволяють ефективно

передавати WebRTC-потоки навіть на обмежених ресурсах [7], що є суттєвим для мобільних платформ.

2.1.2 Методи детекції руху на мобільних пристроях

Завдання виявлення руху в приміщенні може вирішуватися різними алгоритмами, що відрізняються обчислювальною складністю, точністю та стійкістю до змін освітлення [1]. Вибір конкретного методу безпосередньо впливає на швидкодію мобільного застосунку, його енергоспоживання та здатність коректно реагувати на появу об’єктів у кадрі. Нижче розглянуто найпоширеніші підходи до детекції руху, визначено їхні переваги й обмеження, а також оцінено доцільність застосування в умовах обмежених апаратних ресурсів смартфонів.

Попіксельне порівняння кадрів – найпростіший метод, що полягає в обчисленні середньої абсолютної різниці яскравості між двома послідовними кадрами. Перевагою є мінімальне навантаження на процесор, що критично для тривалої фонової роботи смартфона. Недоліком – чутливість до змін глобального освітлення (хмари, увімкнення світла), що може призводити до хибних спрацьовувань.

Метод віднімання фону – передбачає побудову та постійне оновлення моделі статичного фону, з яким порівнюється поточний кадр. Дозволяє відокремити рухомі об’єкти від фону навіть при поступових змінах освітлення. Однак потребує більше пам’яті та обчислень, що може бути надмірним для постійної фонової роботи на бюджетних пристроях.

Оптичний потік – група методів, що відстежують напрямок і швидкість руху кожного пікселя між кадрами. Дає змогу оцінювати траєкторії об’єктів, але є найбільш ресурсоємним, що унеможливлює його безперервне застосування на мобільному пристрої без значного енергоспоживання.

Нейромережеві детектори – використовують попередньо навчені згорткові мережі для розпізнавання об’єктів (людей, тварин, транспорту). Забезпечують найвищу якість і мінімум хибних тривог, але вимагають спеціалізованих

обчислювальних блоків (GPU/NPU) і значних енерговитрат, тому на звичайних смартфонах застосовуються лише в обмежених сценаріях.

Для узагальнення характеристик методів наведено таблицю 2.2. У ній зіставлено чотири основні підходи за критеріями обчислювальної складності, точності, стійкості до змін освітлення та енергоспоживання. Такий формат дає змогу компактно відобразити компроміс між якістю детекції та вимогами до апаратних ресурсів мобільного пристрою.

Таблиця 2.2 – Порівняння методів детекції руху

Метод	**Обчислювальна складність**	**Точність**	**Стійкість до освітлення**	**Енергоспоживання**
Попіксельне порівняння	Низька	Середня	Низька	Низьке
Віднімання фону	Середня	Висока	Середня	Середнє
Оптичний потік	Висока	Висока	Висока	Високе
Нейромережеві детектори	Дуже висока	Дуже висока	Висока	Дуже високе

Наведене порівняння засвідчує, що вибір методу детекції руху є компромісом між необхідною точністю виявлення загроз та обмеженими обчислювальними ресурсами мобільного пристрою. Складніші алгоритми, попри вищу стійкість до завад, можуть спричинити неприпустиме енергоспоживання під час цілодобової фонової роботи, тоді як простіші методи здатні забезпечити прийнятну якість детекції в контрольованих умовах приміщення.

2.1.3 Методи автентифікації та авторизації в розподілених системах

Для захисту від несанкціонованого доступу до відеопотоків та архівів система потребує надійних механізмів перевірки особи та прав користувачів.

Сесійна модель – традиційний підхід, за якого сервер створює сесію після успішної автентифікації, а клієнт передає ідентифікатор сесії. Недоліком є необхідність зберігання стану на сервері, що ускладнює масштабування.

JWT [8] – бездержавний механізм, за якого сервер видає клієнту підписаний токен із закодованою інформацією про користувача. Під час кожного запиту клієнт надсилає цей токен, а сервер перевіряє його підпис без звернення до бази даних. Це

спрощує горизонтальне масштабування, але потребує механізмів для відкликання скомпрометованих токенів (refresh-токени, чорні списки).

OAuth 2.0 / OpenID Connect – протоколи для делегованого доступу, що широко застосовуються у випадках, коли потрібна інтеграція із зовнішніми провайдерами (Google, Facebook). У контексті охоронної системи вони можуть бути корисними для спрощення реєстрації, але додають залежність від сторонніх сервісів.

Окремо слід розглянути методи гранульованого управління доступом. Традиційна рольова модель (RBAC) призначає користувачам ролі (власник, адміністратор, глядач), кожна з яких має фіксований набір прав. Гнучкішою є модель на основі атрибутів (ABAC), де права визначаються комбінацією атрибутів користувача, ресурсу та операції. Саме ABAC-подібний підхід із незалежними прапорцями дозволів дає змогу точно налаштувати доступ, що особливо важливо для спільного користування камерою.

2.1.4 Методи зберігання та архівації відеоданих

Відеофрагменти, створені під час тривожних подій, потребують надійного зберігання. У розподілених системах застосовуються такі підходи.

Локальне зберігання на пристрої – записи зберігаються на внутрішній пам’яті смартфона. Це усуває залежність від мережі, але обмежує обсяг архіву та створює ризик втрати даних у разі пошкодження пристрою.

Віддалене файлове сховище – записи завантажуються на окремий сервер за протоколом FTP/SFTP. Підходить для резервного копіювання, але не забезпечує зручного доступу до архіву через API.

Об’єктне сховище (S3-сумісне) – сучасний підхід, що передбачає зберігання файлів у вигляді об’єктів із метаданими та підтримкою прямого доступу через HTTP. Такі сховища, як MinIO [9] або Amazon S3, надають API для завантаження, вивантаження та контролю доступу, легко масштабуються і можуть бути розгорнуті як на власному обладнанні, так і в хмарі.

Для забезпечення відмовостійкості доцільно поєднувати локальне кешування (тимчасове зберігання на смартфоні) з автоматичним завантаженням до об'єктного сховища. Такий гібридний метод дозволяє не втрачати записи під час відсутності інтернету.

2.1.5 Архітектурні підходи до побудови клієнт-серверних систем відеоспостереження

Вибір загальної архітектури безпосередньо впливає на масштабованість, складність розгортання та безпеку системи. Неправильно обрана архітектурна модель може призвести до невиправданих витрат на підтримку інфраструктури або унеможливити подальше розширення функціоналу. Особливо гостро ця проблема постає в охоронних системах, де збій одного компонента не повинен порушувати роботу інших – наприклад, відмова сервісу архівації не має блокувати детекцію руху чи надсилання сповіщень. Тому розглянуто основні архітектурні стилі, їхні сильні сторони та обмеження, що дозволить надалі прийняти виважене проєктне рішення.

Монолітна архітектура, за якої всі функції (автентифікація, обробка відео, зберігання записів) реалізовані в єдиному застосунку, вирізняється простотою розробки та розгортання, проте стикається з труднощами масштабування та низькою відмовостійкістю. На противагу їй, мікросервісна архітектура виносить кожну функцію в окремий незалежний сервіс, що взаємодіють через API; це дозволяє масштабувати лише навантажені компоненти, ізолювати збої та спрощувати оновлення, однак значно підвищує складність розгортання та адміністрування. Компромісним рішенням виступає гібридний підхід, який поєднує монолітне ядро з окремими сервісами для специфічних завдань (наприклад, сигнальний сервер для WebRTC), дозволяючи отримати переваги мікросервісів там, де це дійсно потрібно, без надмірного ускладнення всієї системи.

Також важливим є спосіб постачання серверної частини. Використання контейнеризації (Docker [10]) забезпечує однакове середовище виконання на будь-якому сервері, спрощує встановлення, оновлення та відтворюваність розгортання.

Для охоронної системи, яка може встановлюватися на власному обладнанні користувача, це є суттєвою перевагою, оскільки мінімізує вимоги до технічної кваліфікації адміністратора.

2.2 Специфікація вимог до програмного забезпечення

1. Призначення та межі проєкту

1.1 Призначення системи

Програмний комплекс призначений для організації віддаленого відеоспостереження за приміщеннями з використанням Android-смартфонів у ролі камер. Система забезпечує потокову трансляцію відео в реальному часі, автоматичну детекцію руху, запис подій, збереження архіву на контрольованому користувачем сервері, гнучке управління правами доступу та миттєві сповіщення. Цільовою аудиторією є фізичні особи та малий бізнес, які потребують бюджетного, але надійного рішення для фізичної безпеки.

1.2 Погодження, що ухвалені в програмній документації

- специфікацію розроблено на основі аналізу предметної області та існуючих аналогів, виконаного в розділі 1;
- ухвалено використання клієнт-серверної архітектури з мобільним застосунком та віддаленим сервером, що може бути розгорнутий на власному обладнанні користувача;
- ухвалено модель гранульованих дозволів із шістьма незалежними прапорцями для точного налаштування прав доступу;
- ухвалено термін «камера» для позначення мобільного пристрою, що працює в режимі захоплення та передавання відео.

1.3 Межі проєкту ПЗ

- проєкт охоплює розробку мобільного застосунку для платформи Android (з можливістю подальшої підтримки iOS) та серверної частини, що включає REST API, сигнальний сервер для відеоз’єднань, об’єктне сховище записів та базу даних;

– не передбачено розробку окремих нативних застосунків для інших мобільних платформ, окрім Android;

– не включає інтеграцію з професійними пультами централізованого спостереження, системами відеоаналітики на базі штучного інтелекту або апаратними відеореєстраторами;

– не охоплює розробку апаратного забезпечення чи модифікацію операційної системи пристрою.

2. Загальний опис

2.1 Сфера застосування

Система застосовується для охорони квартир, приватних будинків, офісних приміщень, складів, дачних ділянок – будь-яких стаціонарних об’єктів, де потрібен віддалений візуальний контроль із можливістю реагування на несанкціоноване проникнення. Використання застарілих смартфонів дозволяє мінімізувати фінансові витрати на обладнання.

2.2 Характеристики користувачів

– Власник камери (адміністратор) – особа, яка зареєструвала камеру в системі, має повний набір прав, може запрошувати інших користувачів, керувати дозволами, переглядати архів і журнал подій. Передбачається, що власник володіє базовими навичками користування смартфоном.

– Запрошені користувачі (гості) – особи, яким власник надав доступ до камери шляхом надсилання запрошення. Набір дозволів для кожного гостя визначається власником індивідуально і може включати будь-яку комбінацію з шести незалежних прапорців: перегляд архіву, перегляд журналу подій, зміна режиму приватності, зміна режиму сигналізації, ручний запуск/зупинка запису, використання мікрофона.

Усі категорії користувачів не потребують спеціальної технічної підготовки; інтерфейс застосунку має бути інтуїтивно зрозумілим.

2.3 Загальна структура і склад системи

Система складається з таких основних компонентів:

а) Мобільний застосунок у двох режимах: «камера» (захоплення відео, детекція руху, передавання потоку) та «глядач» (перегляд відео, архіву, керування дозволами);

б) Серверна частина, яка включає:

– REST API для обробки запитів клієнтів (автентифікація, камери, записи, дозволи);

– сигнальний –WebSocket-сервер для встановлення відеоз’єднань;

– об’єктне сховище для зберігання відеоархіву;

– реляційну базу даних для користувачів, камер, дозволів, метаданих записів та журналу подій.

в) Взаємодія між компонентами здійснюється через захищені мережеві протоколи.

2.4 Загальні обмеження

– для потокової трансляції необхідне стабільне інтернет-з’єднання (Wi-Fi або мобільний інтернет із пропускною здатністю не нижче 1 Мбіт/с);

– мінімальна версія операційної системи на пристрої-камері – Android 7.0;

– серверна частина потребує окремого фізичного або віртуального сервера з підтримкою контейнеризації;

– кількість одночасних глядачів обмежується продуктивністю сервера та пропускною здатністю мережі;

– безперервна робота камери в фоновому режимі залежить від налаштувань енергозбереження конкретного пристрою.

3. Функції системи

3.1 Автентифікація та керування сесіями

3.1.1 Опис функції: реєстрація нових користувачів, вхід до системи, автоматичне відновлення сесії, вихід.

3.1.2 Вхідна інформація: електронна пошта, пароль. Вихідна: підтвердження успішної операції, токен доступу.

3.1.3 Функціональні вимоги:

– пароль має містити не менше 8 символів;

– паролі зберігаються у вигляді хешу з додаванням солі (bcrypt);

– після 5 невдалих спроб входу з однієї IP-адреси доступ блокується на 5 хвилин;

– підтримка режиму «Запам’ятати мене» з використанням довготривалого токена.

3.2 Керування камерами

3.2.1 Опис функції: реєстрація нового пристрою-камери, перегляд списку камер, зміна режимів роботи, видалення камери.

3.2.2 Вхідна інформація - назва камери, місце розташування (опційно). Вихідна - ідентифікатор камери, підтвердження операції.

3.2.3 Функціональні вимоги:

– при повторному підключенні того самого фізичного пристрою система автоматично розпізнає його за збереженим у захищеному сховищі ключем, не створюючи дублікат;

– власник може видалити камеру, що призводить до каскадного видалення всіх пов’язаних записів і дозволів.

3.3 Потокове відеоспостереження

3.3.1 Опис функції - безперервне захоплення відео з камери смартфона, його кодування та передавання на пристрій глядача в реальному часі.

3.3.2 Вхідна інформація: відеокадри з оптичного сенсора. Вихідна: відеопотік на екрані глядача.

3.3.3 Функціональні вимоги:

– затримка передавання не повинна перевищувати 2 секунди в локальній мережі;

– усі медіадані повинні шифруватися під час передавання;

– після тимчасової втрати мережевого з’єднання система автоматично відновлює трансляцію без втручання користувача;

– підтримується одночасне підключення до однієї камери до 5 глядачів.

3.4 Детекція руху

3.4.1 Опис функції: автоматичний аналіз послідовності кадрів з метою виявлення рухомих об’єктів у контрольованій зоні.

3.4.2 Вхідна інформація: послідовність відеокадрів. Вихідна: сигнал тривоги (за наявності руху), оцінка впевненості (0–100%).

3.4.3 Функціональні вимоги:

– аналіз має виконуватися у фоновому режимі з частотою не менше 2 кадрів на секунду;

– виявлення руху активує негайний запуск запису відео;

– передбачено період «охолодження» (30 секунд), протягом якого повторні події ігноруються для уникнення лавинних сповіщень;

– власник може регулювати чутливість детектора (порогове значення).

3.5 Сповіщення про події

3.5.1 Опис функції: інформування користувача про виявлений рух.

3.5.2 Вхідна інформація: сигнал тривоги від модуля детекції. Вихідна: push-повідомлення на мобільний пристрій, текстове повідомлення в Telegram.

3.5.3 Функціональні вимоги:

– push-повідомлення надсилається через сервіс хмарних повідомлень;

– дублююче повідомлення в Telegram гарантує доставку в разі збоїв основного каналу;

– сповіщення містить інформацію про камеру, час події та рівень впевненості.

3.6 Запис та архівація

3.6.1 Опис функції: автоматичний запис відеофрагмента під час тривожної події, локальне кешування та завантаження до віддаленого сховища.

3.6.2 Вхідна інформація: відеопотік з камери. Вихідна: відеофайл у сховищі, метадані запису.

3.6.3 Функціональні вимоги:

– запис починається негайно після виявлення руху та триває щонайменше 10 секунд після зникнення руху;

– відеофайл спочатку зберігається локально на пристрої-камері;

– завантаження до віддаленого сховища запускається автоматично за наявності мережі;

– у разі невдалої спроби завантаження завдання ставиться в чергу з повторними спробами через 30 секунд;

– локальна копія видаляється лише після підтвердження успішного завантаження.

3.7 Управління доступом

3.7.1 Опис функції - надання, редагування та відкликання дозволів для інших користувачів.

3.7.2 Вхідна інформація - електронна пошта запрошуваного, перелік дозволів. Вихідна - підтвердження операції, сповіщення запрошеному.

3.7.3 Функціональні вимоги:

– дозволи включають шість незалежних прапорців: перегляд архіву, перегляд журналу подій, зміна режиму приватності, зміна режиму сигналізації, ручний запуск/зупинка запису, використання мікрофона;

– запрошення має обмежений термін дії (7 днів);

– отримувач може прийняти або відхилити запрошення;

– власник може в будь-який момент змінити набір дозволів або повністю відкликати доступ;

– усі зміни дозволів фіксуються в журналі подій.

3.8 Журнал подій

3.8.1 Опис функції - автоматична фіксація критичних дій користувачів та подій системи.

3.8.2 Вхідна інформація - дії користувачів. Вихідна - записи журналу з часовими мітками.

3.8.3 Функціональні вимоги:

– фіксуються - вхід/вихід, зміна дозволів, перегляд архіву, увімкнення мікрофона, зміна режиму приватності/сигналізації;

– кожен запис містить ідентифікатор користувача, тип дії та часову мітку;

– журнал доступний лише власнику та користувачам із відповідним дозволом.

4. Вимоги до інформаційного забезпечення

4.1 Джерела і зміст вхідної інформації

– дані користувачів (електронна пошта, хеш пароля, роль) надаються під час реєстрації;

– інформація про камери (назва, місце розташування) вводиться власником;

– відеозаписи генеруються автоматично модулем детекції руху;

– журнал подій формується автоматично під час виконання критичних операцій.

4.2 Нормативно-довідкова інформація

– перелік типів подій для журналу - вхід, вихід, зміна дозволів, перегляд архіву, вмикання мікрофона, зміна режиму приватності, зміна режиму сигналізації;

– перелік дозволів - перегляд архіву, перегляд журналу, зміна приватності, зміна сигналізації, ручний запис, використання мікрофона;

– статуси камери - активна, неактивна, приватний режим.

4.3 Вимоги до способів організації, збереження та ведення інформації

– усі дані зберігаються в реляційній базі даних із підтримкою ACID-транзакцій;

– для відеофайлів використовується окреме об’єктне сховище, що забезпечує потоковий доступ через HTTP;

– забезпечується цілісність даних через механізм зовнішніх ключів та каскадне видалення: при видаленні камери автоматично видаляються пов’язані записи, дозволи та запрошення;

– передбачено автоматичне резервне копіювання бази даних за розкладом.

5. Вимоги до технічного забезпечення

– сервер - фізичний або віртуальний сервер з архітектурою x86-64, обсягом оперативної пам’яті не менше 2 ГБ, дисковим простором для зберігання відеоархіву (залежить від кількості камер та інтенсивності подій);

– мережа - широкосмугове підключення до Інтернету з пропускною здатністю не менше 10 Мбіт/с для сервера; для клієнтських пристроїв – від 1 Мбіт/с;

– клієнтські пристрої: смартфони з операційною системою Android 7.0 або новішою, наявністю основної камери, модуля Wi-Fi та/або мобільного інтернету; мінімальний обсяг оперативної пам’яті – 2 ГБ.

6. Вимоги до програмного забезпечення

6.1 Архітектура програмної системи

Модульна клієнт-серверна архітектура з чітким розділенням на мобільний застосунок (режими «камера» і «глядач») та серверну частину, яка складається з окремих сервісів (REST API, сигнальний сервер, сховище). Сервіси повинні бути слабко пов’язаними, взаємодіяти через стандартизовані інтерфейси та мати можливість незалежного оновлення.

6.2 Системне програмне забезпечення

Серверна частина поставляється у вигляді Docker-контейнерів, що забезпечує її розгортання на будь-якій операційній системі, яка підтримує Docker (Linux, Windows, macOS). Для продуктивного використання рекомендованою серверною операційною системою є Linux (Ubuntu 22.04 LTS або новіша). Клієнтський застосунок працює під управлінням Android 7.0 або новішої версії.

6.3 Мережеве програмне забезпечення

Для забезпечення захищеного зв’язку використовуються протоколи HTTPS (TLS 1.2+) для REST API, WSS для сигнального сервера та DTLS-SRTP для шифрування медіапотоків WebRTC. Серверна інфраструктура може бути додатково захищена зворотним проксі-сервером (наприклад, Nginx) для термінації TLS і балансування навантаження.

6.4 Програмне забезпечення ведення інформаційної бази

Реляційна система управління базами даних із підтримкою ACID-транзакцій, зовнішніх ключів, каскадного видалення та засобів резервного копіювання.

Об’єктне сховище – S3-сумісне, з API для завантаження, вивантаження та контролю доступу.

6.5 Мова і технологія розробки ПЗ

Технологічний стек системи включає наступні складові:

– клієнтський застосунок – мова програмування Dart, кросплатформений фреймворк Flutter [11], що дозволяє створювати єдину кодову базу для Android (та в перспективі iOS);

– серверна частина (REST API) – платформа Node.js [12], мова програмування TypeScript [13], вебфреймворк Express.js [14];

– база даних – реляційна СКБД PostgreSQL [15];

– об’єктне сховище – MinIO (S3-сумісне);

– сигнальний сервер – WebSocket-сервер на базі Node.js/TypeScript;

– контейнеризація та розгортання – Docker, Docker Compose.

Цей стек забезпечує підтримку асинхронної обробки запитів, високу продуктивність, кросплатформеність клієнта, а також простоту розгортання та оновлення серверної інфраструктури.

7. Вимоги до зовнішніх інтерфейсів

7.1 Інтерфейс користувача

Мобільний застосунок із мінімалістичним дизайном, що відповідає стандартам платформи Material Design. Інтерфейс глядача повинен забезпечувати виконання основних операцій (перегляд відео, перевірка архіву, перемикання режимів) не більше ніж за три натискання. Інтерфейс камери має бути оптимізований для тривалої фонової роботи з можливістю вимкнення екрана.

7.2 Апаратний інтерфейс

Застосунок взаємодіє з камерою смартфона через стандартні API платформи Android (Camera API2). Спеціалізоване апаратне забезпечення не потрібне.

7.3 Програмний інтерфейс

Серверна частина надає REST API для всіх операцій, окрім передавання медіапотоку. Формат обміну – JSON з уніфікованою структурою відповіді (success,

data, message). Для встановлення відеоз’єднань використовується WebSocket-сервер, що обмінюється SDP-повідомленнями та ICE-кандидатами.

7.4 Комунікаційний протокол

- HTTPS для REST API (з використанням TLS 1.2+);
- WSS для сигнального WebSocket-сервера;
- DTLS-SRTP для шифрування медіапотоків.

8. Властивості програмного забезпечення

8.1 Доступність

Час безперебійної роботи серверної частини має становити не менше 99.5%.

8.2 Супроводжуваність

- модульна структура коду для спрощення внесення змін;
- автоматизоване розгортання через контейнеризацію;
- наявність документації API (Swagger/OpenAPI);
- обов’язкове використання системи контролю версій.

8.3 Переносимість

Серверна частина має бути сумісною з будь-якими платформами, що підтримують контейнеризацію (Linux, Windows, macOS). Клієнтський застосунок має підтримувати Android 7.0+ із можливістю перенесення на iOS завдяки використанню кросплатформного фреймворку.

8.4 Продуктивність

- час відповіді API – не більше 300 мс при одночасних 50 запитах;
- частота аналізу кадрів на рух – не менше 2 кадрів/с;
- споживання оперативної пам’яті застосунком у режимі камери – не більше 150 МБ.

8.5 Надійність

- автоматичне відновлення WebRTC-з’єднання після розриву;
- повторні спроби завантаження відеозаписів у разі невдачі;
- автоматичний перезапуск серверних сервісів після збою (засобами оркестрації контейнерів);
- цілісність даних забезпечується транзакціями ACID.

8.6 Безпека

– наскрізне шифрування на всіх етапах обміну даними;

– хешування паролів із використанням bcrypt (фактор складності ≥ 12);

– захист від основних веб-вразливостей: SQL-ін’єкцій (використання параметризованих запитів), міжсайтового скриптингу (екранування вихідних даних), підробки міжсайтових запитів;

– обмеження частоти запитів (rate limiting) для запобігання перебору паролів;

– ізоляція даних між різними камерами та користувачами на рівні бізнес-логіки та бази даних.

9. Інші вимоги

– відкритий код - вихідний код системи повинен бути розміщений у публічному репозиторії, що забезпечує можливість незалежного аудиту безпеки та розширення функціоналу спільнотою;

– резервне копіювання - можливість налаштування автоматичного резервного копіювання бази даних та відеоархіву стандартними засобами операційної системи;

– збереження налаштувань при оновленні - усі налаштування користувача повинні зберігатися після оновлення серверної частини на нову версію.

Висновки до розділу 2

У другому розділі проведено дослідження сучасних методів, які можуть бути покладені в основу реалізації програмної системи відеоспостереження на базі мобільних пристроїв. Розглянуто альтернативні підходи до організації потокової передачі відео (RTSP/RTP, HLS, WebRTC), алгоритми детекції руху (попіксельне порівняння, віднімання фону, оптичний потік, нейромережеві детектори), механізми автентифікації та авторизації (сесійна модель, JWT, OAuth 2.0, RBAC/ABAC), способи зберігання відеоданих (локальне, віддалене файлове, об’єктне сховище), а також архітектурні стилі побудови клієнт-серверних систем

(монолітний, мікросервісний, гібридний) та переваги контейнеризації. Для кожної групи методів виявлено сильні сторони й обмеження, що створює підґрунтя для обґрунтованого вибору конкретних технологій на етапі проєктування.

Ключовим результатом розділу є детальна специфікація вимог до програмного забезпечення, оформлена відповідно до шаблону, рекомендованого методичними вказівками. Специфікація містить опис призначення та меж проєкту, загальний контекст використання, характеристики користувачів, формалізований перелік функцій системи (27 функціональних вимог, згрупованих за модулями автентифікації, керування камерами, потокового відео, детекції руху, сповіщень, запису та архівації, управління доступом, журналу подій), а також нефункціональні вимоги до інформаційного, технічного, програмного забезпечення, зовнішніх інтерфейсів і властивостей системи (доступність, супроводжуваність, переносимість, продуктивність, надійність, безпека). Особливу увагу приділено вимогам із безпеки, що враховують недоліки існуючих аналогів, виявлені в першому розділі.

Отримані результати формують повну технічну основу для переходу до наступного етапу – проєктування архітектури системи, розробки структури бази даних, побудови UML-діаграм та остаточного вибору технологічного стеку, що буде виконано в третьому розділі кваліфікаційної роботи.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі на основі специфікації вимог, сформованої в попередній частині роботи, розробляються архітектурні рішення, будуються моделі взаємодії компонентів, проєктується структура бази даних та графічний інтерфейс користувача. Отримані результати є основою для подальшої програмної реалізації системи.

3.1 Обґрунтування вибору архітектурних рішень та вибір стеку технологій

Архітектура системи побудована за клієнт-серверною моделлю з гібридним підходом: монолітне ядро для бізнес-логіки доповнене окремими сервісами для високонавантажених функцій. Клієнтська частина – мобільний застосунок на Flutter, що працює у двох ролях: «камера» (захоплення відео, детекція руху, WebRTC-трансляція) та «глядач» (перегляд, керування, архів). Для критичних до продуктивності операцій (безпосереднього захоплення кадрів з камери, їх перетворення та кодування в H.264) використано нативний код, а саме Kotlin для Android, який інтегрується з Flutter через MethodChannel. Це дозволяє отримати максимальну продуктивність та доступ до низькорівневих API, а в перспективі дає змогу адаптувати цю частину для інших платформ без зміни основної логіки застосунку. Серверна частина складається з REST API (Node.js + Express), сигнального WebSocket-сервера для WebRTC та об'єктного сховища MinIO. Усі компоненти контейнеризовані Docker, що забезпечує незалежність від хмар та простоту розгортання.

Мобільний застосунок реалізовано на Clean Architecture з патерном BLoC, що чітко розділяє бізнес-логіку (domain), роботу з даними (data) та інтерфейс (presentation). Це дозволяє змінювати деталі реалізації (наприклад, HTTP-клієнт) без впливу на бізнес-правила, а також спрощує тестування та підтримку.

Технологічний стек обрано за критеріями продуктивності, безпеки та простоти розгортання. Для клієнта – Flutter (Dart) забезпечує єдину кодову базу для Android та iOS, AOT-компіляцію та високу продуктивність. Керування станом –

BLoC (flutter_bloc) дає змогу реактивно обробляти асинхронні події (WebRTC-кадри, статуси з'єднання). HTTP-клієнт Dio підтримує перехоплювачі та автоматичне оновлення токенів при помилці 401. Захищене зберігання токенів забезпечує flutter_secure_storage (Keystore/Keychain). WebRTC реалізовано через flutter_webrtc – офіційний порт нативної WebRTC із затримкою <1 с.

На сервері обрано Node.js + Express з TypeScript: асинхронна модель ідеально підходить для великої кількості WebSocket-з'єднань, а статична типізація полегшує рефакторинг. PostgreSQL обрано як реляційну базу даних через її підтримку ACID-транзакцій, зовнішніх ключів та каскадного видалення, що критично для цілісності даних (при видаленні камери автоматично видаляються дозволи, записи, запрошення). Крім того, PostgreSQL надає типи UUID для унікальних ідентифікаторів, CITEXT для регістронезалежних email-адрес, а також вбудовану підтримку JSONB для гнучкого зберігання метаданих. Можливість реплікації та резервного копіювання робить її надійним вибором для системи безпеки.

Для зберігання відеофайлів використано MinIO – S3-сумісне об'єктне сховище з підтримкою pre-signed URL, шифрування та горизонтального масштабування. Контейнеризація Docker та Docker Compose забезпечують ізоляцію, відтворюваність та незалежність від хмарних платформ. Сповіщення реалізовано через Firebase Cloud Messaging (push) та Telegram Bot API (резервний канал). Для захисту сесій створено власний DeviceFingerprintService, що прив'язує сесію до апаратних характеристик пристрою. Зведену інформацію про технології наведено в таблиці 3.2.

Таблиця 3.2 – Технологічний стек системи

Компонент	Технологія	Обґрунтування
Мобільний клієнт	Flutter (Dart)	Єдина кодова база для Android та iOS, AOT-компіляція, висока продуктивність
Керування станом	BLoC + flutter_bloc	Реактивний підхід, інтеграція з Clean Architecture, підтримка асинхронних потоків
HTTP-клієнт	Dio	Перехоплювачі, глобальна конфігурація, простота роботи з FormData
Захищене сховище	flutter_secure_storage	Апаратне шифрування (Android Keystore / iOS Keychain)
WebRTC	flutter_webrtc	Офіційний порт нативної WebRTC, повна підтримка ICE/STUN/TURN

Кінець таблиці 3.2

Сервер	Node.js + Express + TypeScript	Асинхронна модель, висока продуктивність WebSocket, статична типізація
База даних	PostgreSQL	ACID-транзакції, зовнішні ключі, типи UUID та CITEXT
Об’єктне сховище	MinIO	S3-сумісне, pre-signed URL, шифрування, контейнеризація
Контейнеризація	Docker + Docker Compose	Ізоляція, відтворюваність, простота розгортання
Push-сповіщення	Firebase Cloud Messaging	Безкоштовний, масштабований, підтримка Android/iOS
Резервні сповіщення	Telegram Bot API	Дублювання каналу, доставка при вимкненому застосунку

Обраний стек повністю забезпечує низьку затримку передачі відео, безпеку даних, масштабованість та простоту розгортання. Використання нативного коду для захоплення кадрів дозволяє досягти максимальної продуктивності на Android, при цьому основна логіка залишається кросплатформенною. Відкритий характер технологій гарантує незалежність від зовнішніх постачальників та можливість подальшого розвитку силами спільноти.

3.2 Діаграма варіантів використання

Діаграма варіантів використання є одним із ключових інструментів моделювання в нотації UML, призначеним для візуалізації функціональних вимог до системи. Вона демонструє, які зовнішні актори (користувачі) взаємодіють з програмною системою та яких цілей (варіантів використання) вони можуть досягти. Завдяки цьому діаграма дозволяє швидко виявити пропущені функції або надлишкові сценарії ще на етапі аналізу. Для системи діаграма варіантів використання відіграє особливу роль, оскільки вона наочно показує, як гранульована модель дозволів (шість незалежних прапорців) розмежовує права власника та гостя.

Крім того, діаграма визначає межі системи, окреслюючи, що саме входить до її функціоналу, а що залишається за її межами. Побудована модель слугує основою для розробки діаграм діяльності та послідовності, що будуть представлені в наступних підрозділах. Акторами системи є дві категорії користувачів, кожна з яких має чітко визначений набір прав, що відповідає вимогам гранульованої моделі доступу, що закладена в архітектурі.

Власник – користувач, який створив камеру, зареєструвавши смартфон як пристрій спостереження. Власник має повний, необмежений доступ до всіх функцій, пов’язаних із цією камерою: може змінювати будь-які налаштування, видаляти камеру, запрошувати гостей, призначати та змінювати їм дозволи (шість незалежних прапорців), переглядати архів записів, журнал аудиту, а також самостійно керувати режимами приватності та сигналізації.

Гість – користувач, який отримав запрошення до конкретної камери від її власника. Права гостя не фіксовані, а визначаються власником за допомогою шести незалежних прапорців дозволів. Залежно від налаштувань гість може: переглядати відеопотік у реальному часі, увімкнути мікрофон (двосторонній аудіозв’язок), переглядати архів записів, переглядати журнал подій, змінювати режим приватності камери, змінювати режим сигналізації, запускати ручний запис. Жоден гість не має права видаляти камеру, запрошувати інших користувачів або змінювати дозволи інших гостей.

Система надає низку варіантів використання, згрупованих за функціональними блоками. Управління сесією та обліковим записом включає: реєстрацію нового користувача, вхід до системи з можливістю «Запам’ятати мене» (зберігання refresh-токена), автоматичне відновлення сесії за збереженими токенами, вихід, а також перегляд та редагування власного профілю. Ці варіанти доступні всім автентифікованим користувачам.

Управління камерами дозволяє власнику створити нову камеру (перевести смартфон у режим «камера» з автоматичним збереженням ідентифікатора пристрою), переглянути список усіх доступних йому камер (як власних, так і тих, на які він отримав запрошення), отримати детальну інформацію про конкретну камеру, змінити її режими (приватність, сигналізація), а також видалити камеру, яку він створив. Гість може лише переглядати список камер, до яких має доступ, та змінювати ті режими, на які йому надано відповідний дозвіл (наприклад, увімкнути мікрофон або запустити ручний запис).

Відеоспостереження в реальному часі є ключовою функцією системи. Вона передбачає встановлення WebRTC-з’єднання між камерою та глядачем через

сигнальний WebSocket-сервер, обмін SDP-повідомленнями та ICE-кандидатами, передавання зашифрованого відеопотоку (DTLS-SRTP) та, за наявності дозволу, активацію мікрофона для двостороннього аудіозв'язку. Як власник, так і гість (якщо йому дозволено) можуть переглядати відео.

Детекція руху та запис – це повністю автоматизований процес на стороні камери, який не потребує безпосередньої участі користувача. Камера в режимі сигналізації постійно аналізує відеопотік спрощеним алгоритмом попіксельного порівняння, і при перевищенні порогу чутливості ініціює створення відеофрагмента. Запис спочатку зберігається локально на пристрої-камері (тимчасове кешування), а потім автоматично завантажується до об'єктного сховища MinIO з повторними спробами у разі невдачі. Власник може переглядати архів записів, видаляти окремі записи, а також вручну запускати запис (навіть без виявлення руху). Гість може переглядати архів лише за наявності відповідного дозволу, а видаляти записи – ніколи.

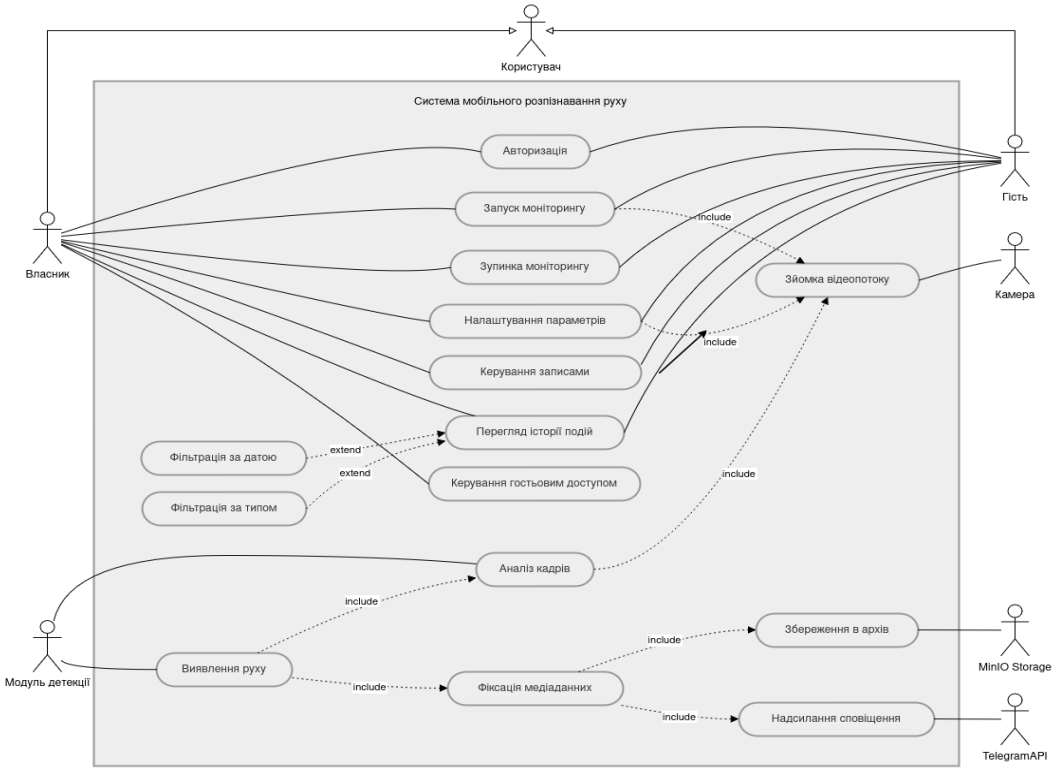

Рисунок 3.1 – Діаграма варіантів використання системи

Управління доступом та запрошення є виключним правом власника камери. Він може надіслати запрошення іншому зареєстрованому користувачеві за

електронною поштою, обравши будь-яку комбінацію з шести гранульованих дозволів (перегляд архіву, перегляд журналу, зміна приватності, зміна сигналізації, ручний запис, використання мікрофона). Запрошення діє 7 днів; отримувач може його прийняти або відхилити. Після прийняття власник може в будь-який момент змінити набір дозволів або повністю відкликати доступ. Усі ці дії фіксуються в журналі аудиту.

Сповіщення та журнал подій забезпечують інформування власника про критичні події. Система надсилає push-сповіщення через Firebase Cloud Messaging, а також дублює найважливіші сповіщення (виявлення руху з високою впевненістю) через Telegram Bot API. Власник може переглядати повний журнал аудиту камери (хто, коли і яку дію виконав), а гість – лише якщо йому надано відповідний дозвіл.

Таким чином, діаграма варіантів використання (рис. 3.1) наочно демонструє розподіл функціоналу між акторами: власник має доступ до всіх варіантів, а гість – лише до тих, що відповідають наданим йому дозволам. Така модель повністю відповідає вимогам безпеки та специфікації, розробленій у другому розділі, а також реалізована в коді (відсутність системного адміністратора, гранульовані дозволи через шість прапорців).

3.3 Діаграма послідовності для процесу автентифікації з оновленням токенів

Діаграма послідовності є одним із засобів моделювання динамічної поведінки системи в нотації UML. На рисунку 3.2 наведено діаграму діяльності, що описує процес автентифікації користувача з автоматичним оновленням сесії за допомогою refresh-токенів – один із ключових сценаріїв забезпечення безпеки в системі.

Процес починається з того, що користувач вводить облікові дані (електронну пошту та пароль) на екрані входу та натискає кнопку «Sign In». Мобільний застосунок формує HTTP-запит до REST API сервера за адресою /auth/login, передаючи облікові дані разом з унікальним відбитком пристрою (device fingerprint). Сервер перевіряє наявність користувача в базі даних та порівнює хеш

пароля, що збережено за допомогою bcrypt. Якщо перевірка не вдається, сервер повертає помилку 401 (Unauthorized), а застосунок відображає повідомлення про неправильні облікові дані.

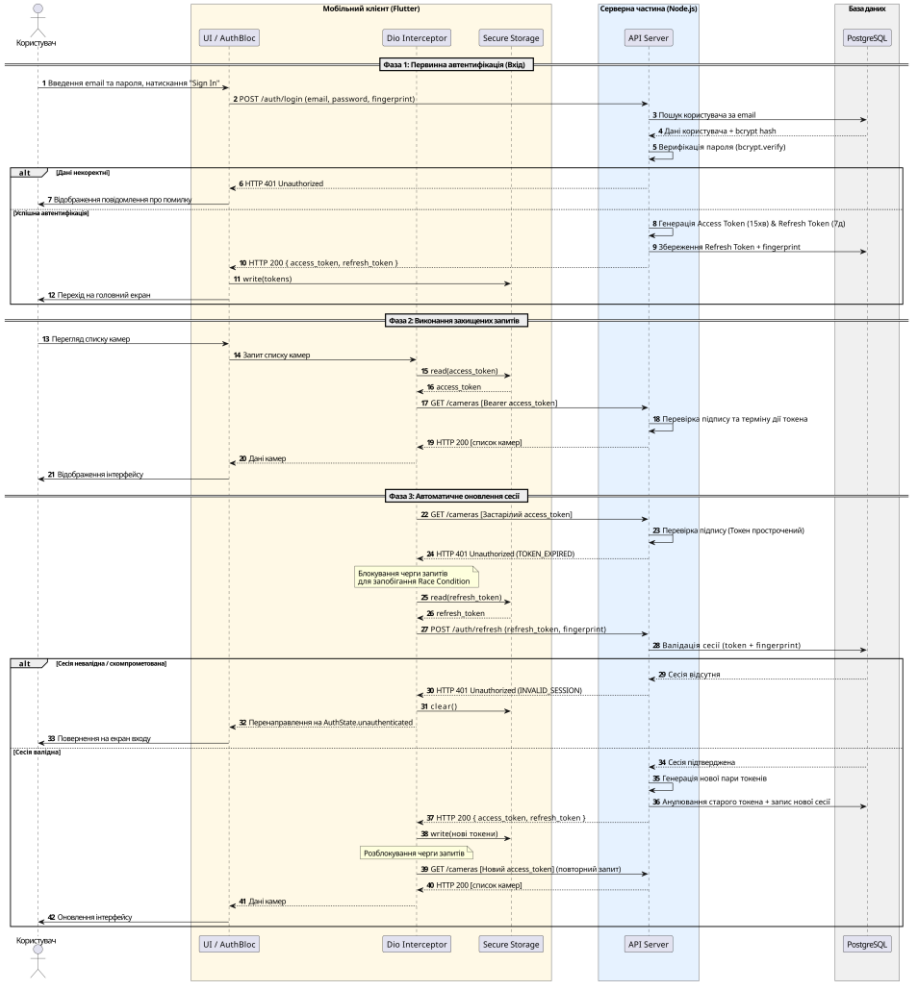

Рисунок 3.2 – Діаграма послідовності для процесу автентифікації та оновлення токенів

У разі успішної автентифікації сервер генерує пару токенів: access token (термін дії 15 хвилин) та refresh token (термін дії 7 днів). Refresh token зберігається в базі даних у зв’язку з відбитком пристрою. Обидва токени повертаються клієнту, який зберігає їх у зашифрованому вигляді (FlutterSecureStorage). Після цього застосунок переходить на головний екран.

Надалі при кожному запиті до захищених ресурсів (наприклад, отримання списку камер) клієнт додає access token до заголовка Authorization. Сервер перевіряє його підпис та термін дії. Якщо токен є дійсним, запит виконується. Якщо

термін дії access token сплив, сервер повертає помилку 401 з кодом TOKEN_EXPIRED.

У цьому випадку клієнт ініціює процес оновлення: надсилає запит до /auth/refresh, передаючи refresh token та відбиток пристрою. Сервер перевіряє, чи існує активна сесія для цього refresh token та чи збігається відбиток пристрою. Якщо перевірка успішна, сервер генерує нову пару токенів, а старий refresh token анулюється (позначається як replaced_at). Особливістю реалізації є grace period (30 секунд): якщо протягом цього часу надходить повторний запит з тим самим старим refresh token (через мережевий збій або повторну спробу), сервер повертає вже згенеровану нову пару, не створюючи дублікатів. Це запобігає помилкам при нестабільному з’єднанні. Якщо ж минуло більше 30 секунд, сесія вважається скомпрометованою і повністю анулюється, що вимагає повторної автентифікації.

Після успішного оновлення клієнт зберігає нові токени та повторює оригінальний запит. Якщо оновлення не вдалося (refresh token відсутній, прострочений, або не збігається відбиток пристрою), застосунок очищає локальне сховище та переводить користувача на екран входу.

Окремим сценарієм є перевірка неактивності користувача. Система відстежує час останньої дії користувача (оновлюється при кожному успішному запиті). Якщо користувач не виконував жодних дій протягом 7 днів, сервер блокує подальші запити та вимагає повторної автентифікації. Це реалізовано на рівні бізнес-логіки в AuthBloc через періодичну перевірку (щогодини) та зберігання часової мітки останньої активності в зашифрованому сховищі.

Таким чином, діаграма діяльності (рис. 3.2) демонструє повний цикл автентифікації, включаючи штатний вхід, роботу з access token, механізм оновлення з grace period та обробку помилок. Такий підхід забезпечує баланс між зручністю користувача (автоматичне оновлення без повторного введення пароля) та безпекою (обмежений термін дії токенів, прив’язка до пристрою, анулювання при підозрі на компрометацію).

3.4 Діаграма послідовності для встановлення WebRTC-з’єднання

Для програмного комплексу, що розроблюється, ключовим сценарієм реального часу є також встановлення WebRTC-з’єднання між пристроєм-камерою (роль «producer») та пристроєм-глядачем (роль «viewer») через сигнальний WebSocket-модуль бекенд-сервера.

На рисунку 3.3 наведено діаграму послідовності, яка ілюструє цей процес. У діаграмі беруть участь чотири основні об’єкти: Глядач (Viewer App) – мобільний застосунок, який бажає переглянути відео; Камера (Camera App) – мобільний застосунок, що захоплює та транслює відеопотік; Сигнальний сервер (Signaling Server) – WebSocket-сервер, який маршрутизує службові повідомлення (SDP, ICE-кандидати) між камерою та глядачем; та STUN/TURN сервер – зовнішній сервер для обходу NAT та ретрансляції медіатрафіку у випадку, якщо пряме peer-to-peer з’єднання неможливе.

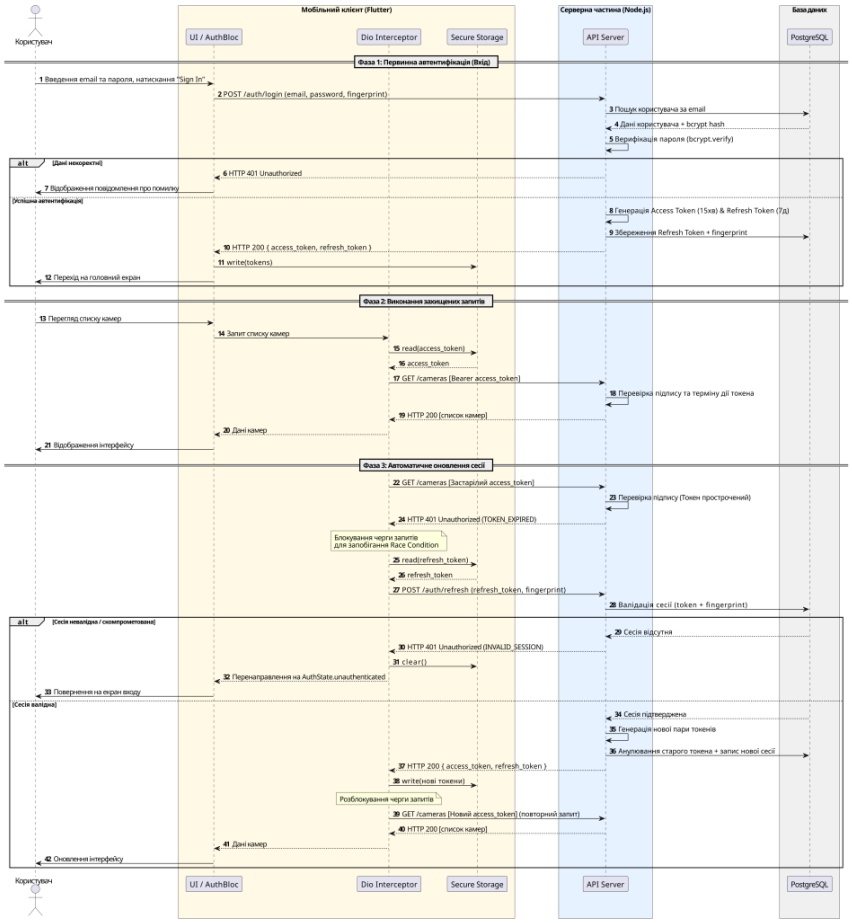

Рисунок 3.3 – Діаграма послідовності для встановлення WebRTC-з’єднання

Процес починається з того, що пристрій-камера вже підключений до сигнального сервера через стійке WebSocket-з’єднання (ініційоване після успішної автентифікації) та перебуває в стані очікування клієнтів. Глядач, відкриваючи відповідний екран мобільного застосунку, встановлює власне WebSocket-підключення до сервера та надсилає службову подію join із зазначенням ідентифікатора камери, до якої він планує підключитися. Сигнальний сервер реєструє запит глядача та транслює камері подію viewer-joined.

Отримавши сповіщення про підключення глядача, пристрій-камера виступає ініціатором створення WebRTC-сесії для трансляції. Камера створює локальний об’єкт RTCPeerConnection, реєструє медіапотоки з камери смартфона за допомогою методу addTrack() та генерує offer – текстовий опис сесії за протоколом SDP, що містить підтримувані кодеки й параметри шифрування. Цей offer встановлюється як локальний опис камери (setLocalDescription) та відправляється на сигнальний сервер через WebSocket-подію offer. Сервер перенаправляє отриманий SDP-опис конкретному глядачеві.

Глядач отримує offer, зберігає його як віддалений опис (setRemoteDescription), створює власний об’єкт RTCPeerConnection та формує відповідь – answer (SDP), яку встановлює локально (setLocalDescription) і повертає камері через сигнальний сервер.

Паралельно з обміном SDP-описами обидва пристрої взаємодіють із зовнішнім STUN/TURN сервером для збору ICE-кандидатів – можливих варіантів мережевих адрес та портів для зв'язку. Кожен згенерований кандидант асинхронно передається протилежній стороні через WebSocket-подію ice-candidate. Отримані кожною стороною кандидати негайно додаються до відповідних об'єктів конфігурації через addIceCandidate().

Коли обидві сторони успішно зберуть та верифікують доступні мережеві маршрути, підсистема WebRTC автоматично виконує перевірку зв'язку. Якщо між пристроями можливе пряме з’єднання, медіатрафік починає передаватися безпосередньо (peer-to-peer) за протоколами SRTP/SRTCP. Якщо брандмауери блокують прямий обмін, трафік проксується через TURN-реле. Успішне

встановлення медіаканалу фіксується зміною стану connectionState на connected, після чого відеопотік декодується та відображається на екрані глядача.

Додатково на діаграмі змодельовано сценарій, коли глядач активує мікрофон для зворотного зв'язку. Мобільний застосунок глядача отримує доступ до аудіопристрою за допомогою getUserMedia(), динамічно додає новий аудіотрек до поточного RTCPeerConnection та ініціює швидкий повторний обмін SDP (Renegotiation) через сигнальний сервер. Камера отримує оновлений опис, додає віддалений трек та відтворює звук від глядача через динамік пристрою, забезпечуючи повноцінний двосторонній аудіозв’язок.

Таким чином, розроблена діаграма послідовності детально відображає архітектуру та логіку сигнального протоколу WebRTC. Цей механізм гарантує передачу відео та аудіо з мінімально можливою затримкою (sub-second latency) та стійкість зв'язку у будь-яких типах мереж.

3.5 Діаграма класів

Діаграма класів є одним із ключових інструментів об'єктно-орієнтованого моделювання в нотації UML і використовується для відображення статичної структури програмної системи: основних класів, їх атрибутів, методів та взаємозв'язків між ними. У контексті розроблюваної системи мобільного розпізнавання руху діаграма класів дозволяє наочно представити архітектурне розмежування між клієнтською частиною, реалізованою на Flutter, та серверною – на Node.js/TypeScript, а також ілюструє внутрішню ієрархію шарів відповідно до принципів Clean Architecture (рис. 3.4).

Клієнтська частина побудована на основі патерну Clean Architecture та охоплює три чітко розмежовані шари. Центральною сутністю шару домену є CameraEntity – незмінний об'єкт, що описує фізичну камеру та містить усі ключові атрибути: ідентифікатор, назву, режими конфіденційності й тривоги, а також обчислювані права доступу для поточного користувача. Доступ до даних абстрагується через інтерфейс CameraRepository, реалізацію якого забезпечує клас CameraRepositoryImpl у шарі даних – він делегує мережеві запити до

CameraRemoteDataSource та перетворює отримані моделі на доменні сутності. У шарі представлення бізнес-логіку інкапсулює CameraBloc, який відповідно до патерну BLoC приймає події типу CameraEvent, виконує операції через репозиторій і публікує нові стани CameraState у реактивний потік, на який підписані компоненти інтерфейсу.

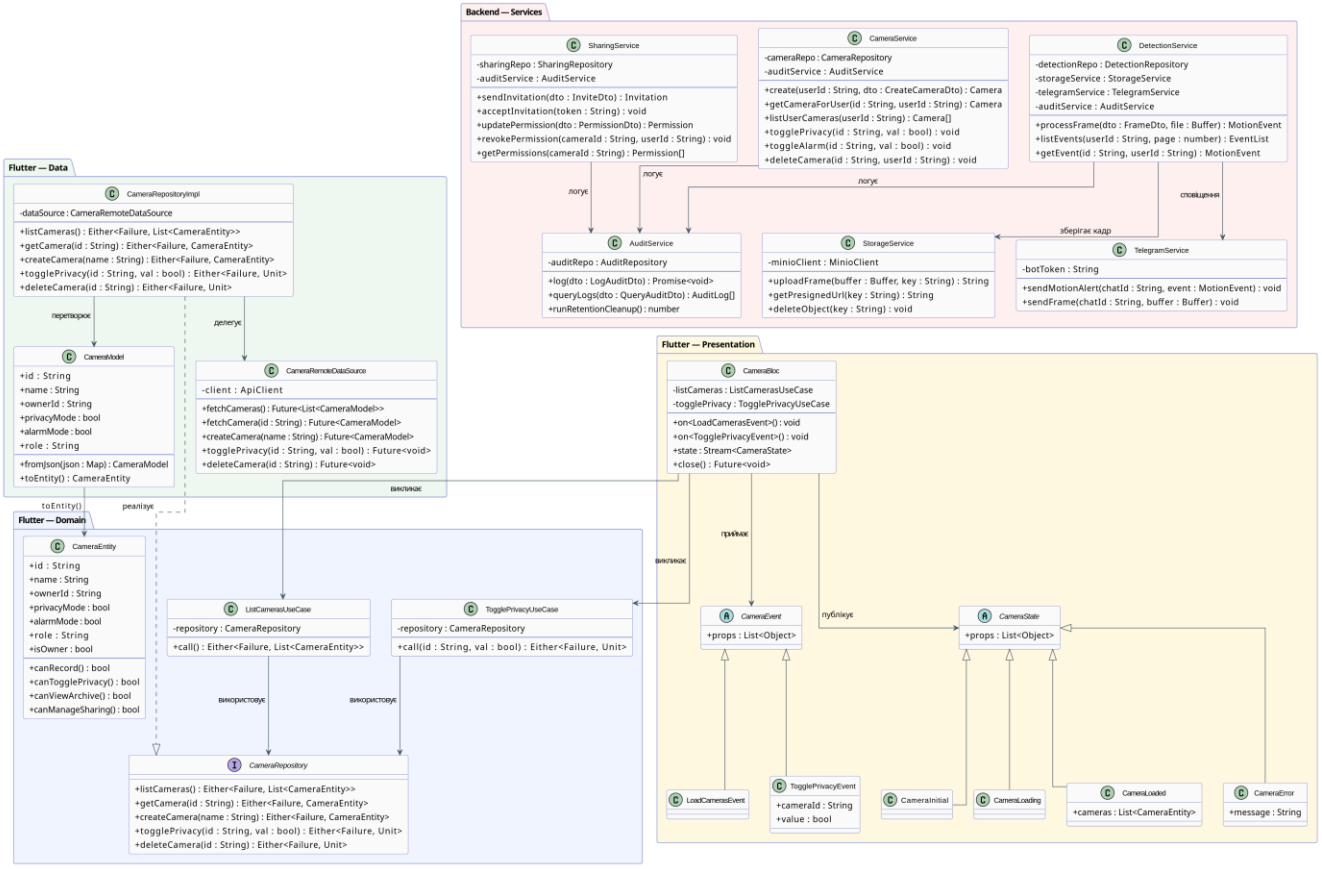

Рисунок 3.4 – Діаграма класів системи мобільного виявлення руху

Серверна частина організована за принципом односпрямованої залежності. Клас CameraService інкапсулює всю бізнес-логіку роботи з камерами та делегує операції до CameraRepository і AuditService. Клас DetectionService відповідає за обробку кадрів від мобільних пристроїв: завантаження до об'єктного сховища MinIO через StorageService, збереження події у базі даних та відправлення сповіщень через TelegramService. AuditService реалізує незмінний журнал системних подій з підтримкою TTL-очищення відповідно до індивідуальних налаштувань термінів зберігання кожного користувача, забезпечуючи таким чином аудитний слід усіх дій у системі.

3.6 Проєктування бази даних

Для забезпечення цілісності, масштабованості та ефективності зберігання даних спроєктовано реляційну схему бази даних у СКБД PostgreSQL. Вона охоплює всі сутності, необхідні для функціонування системи: користувачів, камери, права доступу, запрошення, сесії, записи відео, події руху, папки та аудит. Центральною таблицею є cameras, яка зберігає інформацію про кожен пристрій-камеру: назву, опис, URL потоку, місце розташування, а також прапорці privacy_mode та alarm_mode, що визначають поточний режим роботи. Кожна камера прив'язана до власника через зовнішній ключ owner_id на таблицю users. Завдяки каскадному видаленню, при видаленні камери автоматично очищаються всі пов'язані записи – дозволи, запрошення, відеозаписи та події руху, що гарантує цілісність даних на рівні бази.

Ключовою особливістю системи є гранульована модель доступу, реалізована в таблиці camera_permissions. Для кожної пари «користувач – камера» зберігаються шість незалежних прапорців дозволів: перегляд архіву (can_view_archive), перегляд журналу аудиту (can_view_logs), зміна режиму приватності (can_toggle_privacy), зміна режиму сигналізації (can_toggle_alarm), ручний запуск запису (can_trigger_manual_record) та використання мікрофона (can_use_microphone). Таблиця також містить поле role (owner, admin, viewer, custom), яке використовується для спрощення перевірок, але саме прапорці забезпечують гнучке налаштування прав для кожного гостя. Запрошення для спільного доступу зберігаються в таблиці invitations, яка фіксує запропонований набір прав (знімок на момент відправлення) і має обмежений термін дії – 7 днів. Після прийняття запрошення права копіюються до camera_permissions, а саме запрошення позначається як accepted або declined.

Для безпечного керування сесіями використовується таблиця sessions. Вона містить refresh_token та device_fingerprint – унікальний відбиток пристрою, який генерується на клієнті. Це унеможливлює перенесення сесії на інший пристрій навіть за умови компрометації токена. Механізм ротації сесій реалізовано через

поля replaced_at та is_revoked: при кожному успішному оновленні токена стара сесія позначається як відкликана, а нова отримує свіжий refresh_token. Аудит системи забезпечує таблиця audit_logs, яка працює в режимі «тільки додавання» (жодних оновлень або видалень через застосунок). Вона фіксує дії користувачів – вхід, вихід, зміну дозволів, перегляд архіву, увімкнення мікрофона тощо – разом із метаданими в полі JSONB. Це дозволяє зберігати довільну структуру даних для різних типів подій, наприклад, ідентифікатор камери, рівень впевненості руху, IP-адресу та user agent.

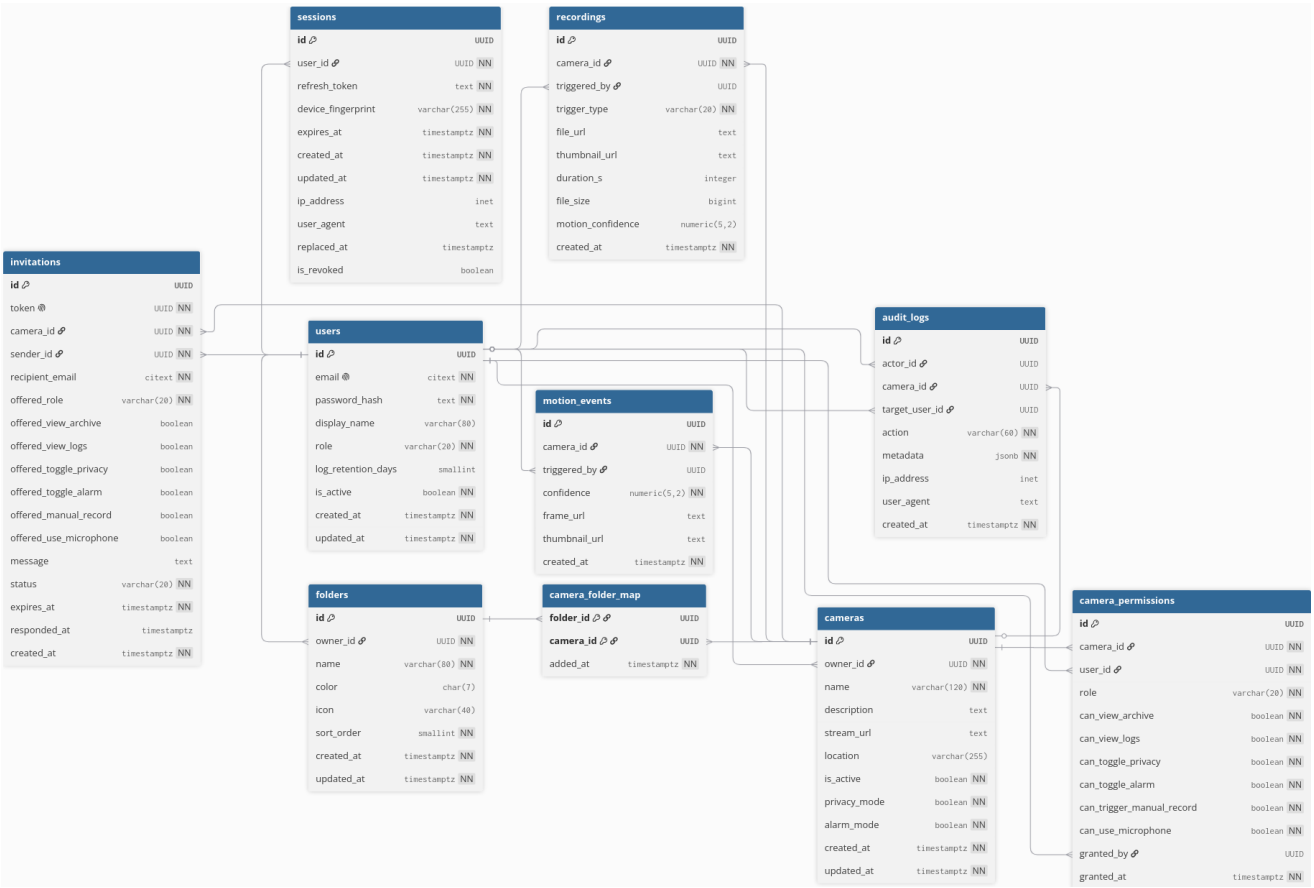

Рисунок 3.5 – ER діаграма

Для зберігання відеозаписів та кадрів руху створено таблиці recordings та motion_events. Вони містять лише метадані – посилання на файли в MinIO (file_url), тривалість, розмір, рівень впевненості (для автоматичних записів) та часову мітку. Самі медіафайли зберігаються в об’єктному сховищі, а доступ до них надається через генерацію короткочасних presigned URL, що контролюються на рівні API. Таблиця recordings підтримує два типи тригерів: manual (користувач натиснув

кнопку запису) та motion (автоматичне виявлення руху). Для зручного впорядкування камер передбачено таблиці folders (поля: назва, колір, іконка, порядок сортування) та camera_folder_map, яка реалізує зв’язок багато-до-багатьох між папками та камерами. Усі зовнішні ключі налаштовано з каскадним видаленням, що забезпечує автоматичне очищення залежних записів. Використання типів UUID для первинних ключів, CITEXT для регістронезалежних email-адрес та JSONB для гнучких метаданих гарантує оптимальну продуктивність та відповідність вимогам системи.

3.7 Проєктування графічного інтерфейсу користувача

Графічний інтерфейс користувача є візуальною візитівкою будь-якого сучасного застосунку та безпосередньо впливає на сприйняття програмного продукту кінцевим користувачем. Саме через інтерфейс користувач оцінює зручність, швидкість та надійність системи, тому якісний дизайн є не менш важливим, ніж технічна досконаліть програмного забезпечення. Інтуїтивна навігація, лаконічне розташування елементів керування та візуальна узгодженість усіх екранів значно підвищують задоволеність роботою з системою та знижують поріг входження для нових користувачів. Крім того, добре спроєктований інтерфейс зменшує кількість помилок під час роботи та скорочує час на навчання користувачів.

Під час проєктування інтерфейсу системи основний акцент робився на простоту доступу до ключових функцій (перегляд відеопотоку, керування записом, доступ до архіву), чітке візуальне групування елементів відповідно до їхнього призначення та адаптивність до різних розмірів екранів мобільних пристроїв. Особливу увагу приділено розташуванню кнопок керування на екрані трансляції – вони не перекривають відеопотік, але залишаються легкодоступними для швидкої взаємодії. Усі екрани виконано в єдиному стилі, що відповідає концепції темної теми, яка зменшує навантаження на очі при тривалому використанні застосунку та економить енергію на пристроях з OLED-екранами. Темна тема також створює відчуття професійності та сучасності, що позитивно впливає на загальне враження

від продукту. На рисунках 3.5–3.7 наведено основні екрани застосунку, що демонструють ключові сценарії взаємодії користувача з системою.

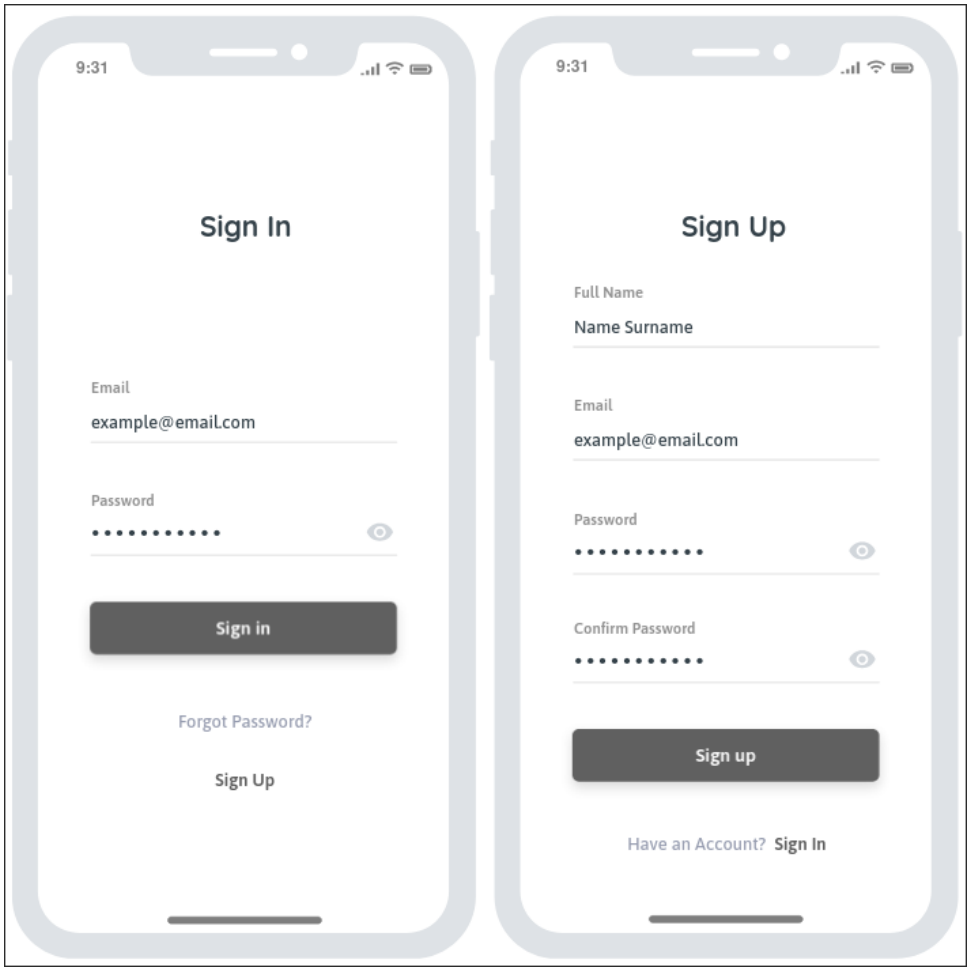

Рисунок 3.6 – Екран авторизації та реєстрації

Екран зустрічає користувача стриманим інтерфейсом, де центральне місце займає форма входу з двома текстовими полями для пошти та пароля. Кожне поле має чіткі межі та внутрішні підказки, а під ними розташована широка кнопка входу, колір якої збігається з іншими активними елементами системи. Внизу екрана розміщені два текстові посилання для відновлення доступу та переходу до створення нового акаунта, що робить перехід між режимами входу та реєстрації миттєвим. У режимі реєстрації додаються ідентичні за стилем поля для імені та підтвердження пароля, зберігаючи загальну візуальну цілісність.

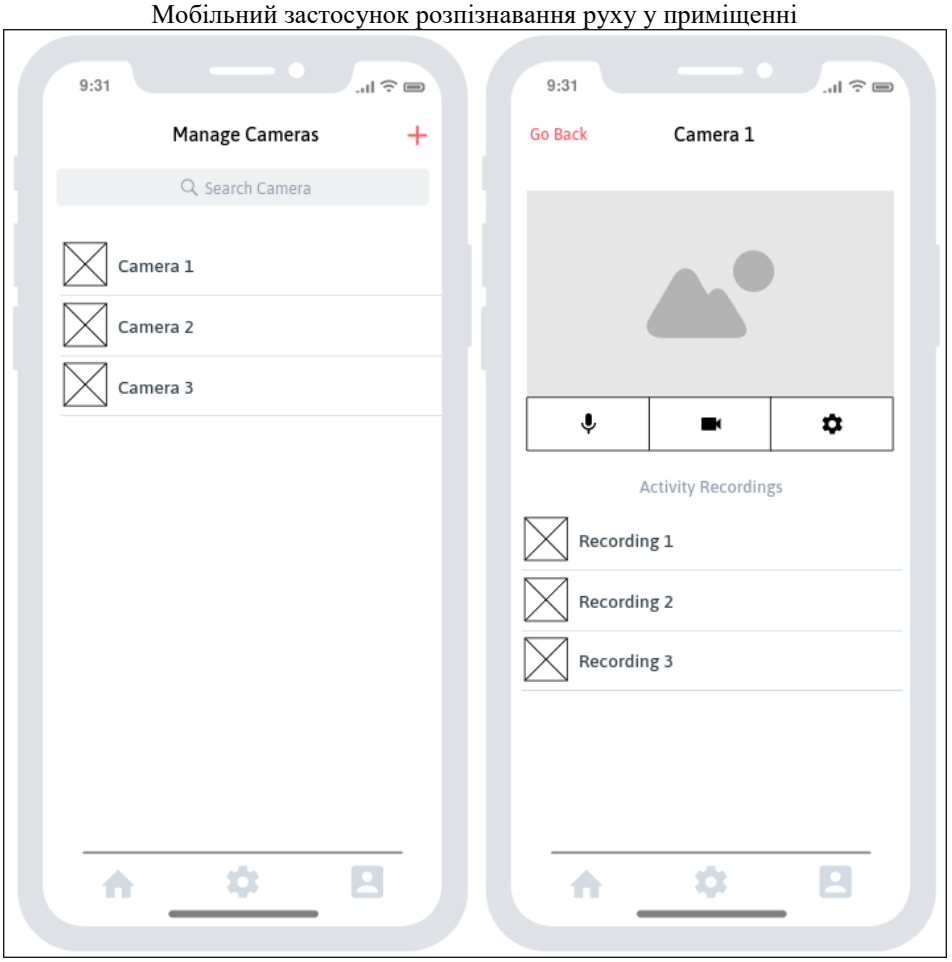

Рисунок 3.7 – Екран списку камер та перегляду трансляції

Головне вікно додатка виконано у вигляді структурованого списку доступних пристроїв, де кожна камера представлена окремою карткою – прямокутним блоком із напівпрозорими межами та плавними кутами. У лівій частині картки розміщено іконку камери, а праворуч – назву пристрою та невеликий індикатор поточного стану у вигляді кольорової точки. Зверху екрана розташована панель навігації.

При натисканні на картку камери користувач потрапляє на детальний екран перегляду. Верхню частину цього екрана займає відеоплеєр із темним фоном, який автоматично адаптується до пропорцій відеопотоку. Плеєр підтримує стандартні елементи керування: повноекранний режим, відображення часу та індикатор завантаження. Безпосередньо під плеєром розташована панель керування, що складається з чотирьох-шести іконок, залежно від прав користувача: кнопка мікрофона (для двостороннього аудіозв’язку), кнопка ручного запису, перемикач режиму приватності та перемикач сигналізації. Кожна кнопка супроводжується

коротким текстовим підписом і змінює колір залежно від активного стану (наприклад, червоний для запису, зелений для активного мікрофона). Нижня частина екрана відведена під вертикальну стрічку останніх подій. Тут у хронологічному порядку (від найновішої до найстарішої) відображаються всі зафіксовані рухи та ручні записи. Така організація дозволяє користувачеві швидко отримати візуальний звіт про активність без необхідності переглядати довгі відеофайли.

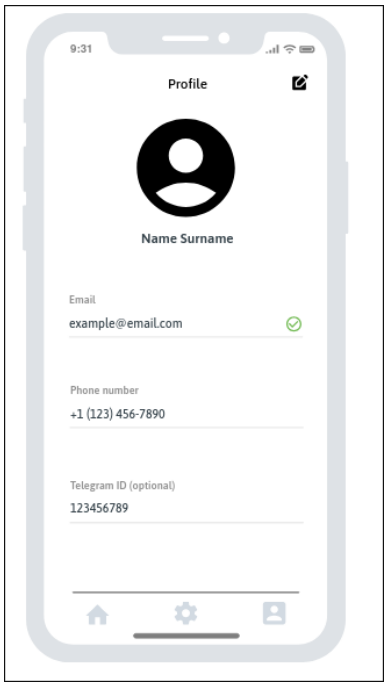

Рисунок 3.8 – Екран особистого профілю

Інтерфейс профілю зосереджений на ідентифікації користувача та виглядає як вертикальний набір інформаційних блоків. У верхній третині екрана розташоване велике коло для аватара, під яким великим шрифтом відображається ім'я власника, створюючи чіткий візуальний центр. Нижче розташовані текстові рядки з персональними даними, такими як електронна адреса з іконкою верифікації та номер телефону.

Висновки до розділу 3

У третьому розділі виконано проєктування системи відеоспостереження. Обґрунтовано вибір архітектури: для сервера – гібридний підхід (монолітне ядро

Node.js + Express, сигнальний WebSocket-сервер для WebRTC, об'єктне сховище MinIO), для мобільного застосунку – Clean Architecture + BLoC. Усі серверні компоненти контейнеризовано Docker, що забезпечує ізоляцію та простоту розгортання. Розроблена гранульована модель дозволів (шість незалежних прапорців) забезпечує гнучке управління доступом до кожної камери.

Побудовано модель варіантів використання з акторами «власник» і «гість» та 20 варіантами, що охоплюють усі ключові функції. Розроблено діаграму послідовності для автентифікації з механізмом оновлення токенів та grace period (30 с), що балансує зручність і безпеку. Ця діаграма також демонструє стійкість системи до тимчасових збоїв мережі завдяки чергуванню запитів. Побудовано діаграму послідовності встановлення WebRTC-з'єднання з обміном SDP та ICE-кандидатами через сигнальний сервер, що забезпечує затримку відео менше 1 секунди.

Створено діаграму класів, яка відображає Clean Architecture: доменний шар (сутності, репозиторії, use cases), шар даних (реалізації, датасорси, моделі) та шар представлення (BLoC, Flutter-віджети). Спроєктовано графічний інтерфейс (екрани авторизації, списку камер, трансляції, профілю) у темній темі.

Обґрунтовано технологічний стек: Flutter (Dart), BLoC, Dio, flutter_secure_storage, flutter_webrtc – для клієнта; Node.js, Express, TypeScript, PostgreSQL, MinIO – для сервера; Docker для контейнеризації; FCM та Telegram Bot API для сповіщень. Усі компоненти відкриті. Прийняті проєктні рішення повністю враховують вимоги до безпеки, масштабованості та автономності, визначені в специфікації.

Розроблені архітектурні рішення та UML-моделі створюють основу для програмної реалізації в четвертому розділі. Вони забезпечують чітке розуміння структури системи та взаємодії її компонентів, що значно знижує ризики помилок під час написання коду. Застосовані підходи до моделювання дозволяють легко масштабувати окремі модулі та адаптувати систему до зміни вимог у майбутньому.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

У цьому розділі описано практичну реалізацію клієнтської та серверної частин системи, а також результати тестування. Розглянуто ключові модулі: автентифікацію з гібридним збереженням токенів, WebRTC-зв’язок, детекцію руху, графічний інтерфейс, сигнальний сервер, роботу з PostgreSQL і MinIO, сповіщення через FCM та Telegram. Завершується розділ комплексним тестуванням – функціональним, навантажувальним та перевіркою стійкості до збоїв мережі.

4.1 Програмна реалізація клієнтської частини

Мобільний застосунок реалізовано на фреймворку Flutter (мова Dart) з використанням архітектури Clean Architecture. Управління станом виконано за допомогою патерну BLoC (Business Logic Component), що забезпечує реактивну обробку асинхронних подій, таких як отримання кадрів з камери, зміна статусу WebRTC-з’єднання або надходження push-сповіщень. Весь код розбито на три шари: domain (бізнес-сутності та інтерфейси репозиторіїв), data (реалізація репозиторіїв, робота з API та локальним сховищем) та presentation (Flutter-віджети та BLoC). Такий підхід дозволяє легко замінювати деталі реалізації (наприклад, HTTP-клієнт) без зміни бізнес-логіки. Далі детально розглянуто ключові модулі клієнтської частини.

4.1.1 Реалізація модуля автентифікації

Модуль автентифікації відповідає за реєстрацію, вхід, автоматичне відновлення сесії та вихід користувача. Уся логіка зосереджена в AuthBloc, який опрацьовує події AuthRegisterRequested, AuthLoginRequested, AuthLogoutRequested, AuthSessionRestoreRequested та публікує відповідні стани AuthLoading, AuthAuthenticated, AuthUnauthenticated, AuthError, AuthInactiveTimeout.

При реєстрації або вході клієнт надсилає на сервер POST /auth/register або /auth/login разом з електронною поштою, паролем та відбитком пристрою (device fingerprint). Відбиток генерується DeviceFingerprintService як комбінація унікального UUID, збереженого в захищеному сховищі flutter_secure_storage, та апаратних характеристик (бренд, модель, версія ОС). Це унеможливлює перенесення сесії на інший пристрій. Усі дані передаються через протокол HTTPS, який забезпечує шифрування каналу за допомогою TLS, тому пароль надходить на сервер у захищеному вигляді; додаткове шифрування на рівні застосунку не застосовується, що є стандартною практикою для веб-API.

Після успішної автентифікації сервер повертає токени accessToken та refreshToken з термінами дії 15 хвилин та 7 днів відповідно. Токени зберігаються через TokenStorage, який реалізує гібридну схему:

```
Future<void> saveTokens({
  required String accessToken,
  required String refreshToken,
  required bool rememberMe,
}) async {
  _inMemoryAccessToken = accessToken;
  await _storage.write(key: _accessTokenKey, value: accessToken);
  if (rememberMe) {
    await _storage.write(key: _refreshTokenKey, value: refreshToken);
    _inMemoryRefreshToken = null;
  } else {
    _inMemoryRefreshToken = refreshToken;
    await _storage.delete(key: _refreshTokenKey);
  }
  await updateLastActivity();
}
```

Якщо користувач обрав опцію «Запам’ятати мене», refreshToken записується в зашифроване дискове сховище flutter_secure_storage; інакше він утримується лише в оперативній пам’яті й втрачається після закриття застосунку. AccessToken завжди зберігається на диску, але має короткий час життя.

Для кожного HTTP-запиту до захищених ресурсів у заголовок Authorization додається accessToken. Це реалізовано в перехоплювачі _AuthInterceptor класу ApiClient. Якщо сервер повертає статус 401 (токен прострочено або недійсний), перехоплювач автоматично ініціює процес оновлення: надсилає POST /auth/refresh з refreshToken та відбитком пристрою. Для запобігання гонкам запитів (коли

декілька запитів паралельно отримують 401) реалізовано чергування запитів під час оновлення: перший запит ініціює оновлення, а всі наступні стають у чергу й чекають на завершення. Після успішного оновлення всі запити з черги повторюються з новим токеном. Якщо оновлення неможливе (наприклад, refreshToken прострочено або не збігається відбиток пристрою), перехоплювач очищає локальне сховище токенів і генерує подію AuthLogoutRequested, яка перенаправляє користувача на екран входу.

Відновлення сесії відбувається при старті застосунку. _SplashGate ініціює подію AuthSessionRestoreRequested. AuthBloc зчитує збережений accessToken через AuthRemoteDataSource.getCachedUser(), отримує userId та роль, а також перевіряє, чи не минуло 7 днів неактивності. Якщо токен існує, термін неактивності не перевищено, то стан переводиться в AuthAuthenticated, і користувач потрапляє на головний екран без повторного введення пароля. Якщо ж термін неактивності перевищено, генерується AuthInactiveTimeout, і сесія очищається з вимогою повторної автентифікації.

Для захисту від брутфорсу на стороні сервера реалізовано обмеження частоти запитів (5 невдалих спроб з однієї IP-адреси за 5 хвилин). У застосунку ці помилки обробляються через стан AuthError із відображенням відповідного повідомлення користувачеві. Усі паролі на сервері зберігаються у вигляді хешу bcrypt з фактором складності 12, що унеможливлює їх відновлення навіть при компрометації бази даних.

Таким чином, модуль автентифікації забезпечує зручний вхід із автоматичним відновленням сесії, високий рівень безпеки завдяки короткоживучим токенам, refresh-механізму, апаратному зберіганню токенів та прив’язці до пристрою, а також стійкість до атак перебору паролів

4.1.2 Налаштування WebRTC-зв’язку

WebRTC-зв’язок забезпечує пряму передачу відеопотоку між пристроєм-камерою (роль «producer») та пристроєм-глядачем (роль «viewer») із мінімальною затримкою. У системі реалізовано повний цикл встановлення з’єднання:

захоплення відео з камери, створення RTCPeerConnection, обмін SDP-повідомленнями та ICE-кандидатами через сигнальний WebSocket-сервер, а також автоматичне відновлення при розривах.

На стороні камери (CameraBloc) після успішної автентифікації встановлюється WebSocket-з'єднання з сигнальним сервером через ApiClient.connectSignaling. Камера надсилає повідомлення join з роллю producer та своїм cameraId. Сигнальний сервер реєструє її в глобальному мапі producerByCamera.

Коли глядач підключається, сервер надсилає камері подію viewer-joined. У відповідь камера створює RTCPeerConnection з використанням STUN-серверів Google, додає локальні відео- та аудіотреки (аудіо за замовчуванням вимкнено) та генерує offer. Offer передається через сигнальний сервер глядачеві, той відповідає answer, після чого сторони обмінюються ICE-кандидатами, знаходять найкращий маршрут і встановлюють медіаканал. Усі медіадані шифруються протоколом DTLS-SRTP.

Ключовою вимогою для системи реального часу є миттєва синхронізація станів між камерою та всіма підключеними глядачами. Будь-яка зміна налаштувань (увімкнення сигналізації, активація режиму приватності, запуск або зупинка запису) має негайно відображатися на всіх пристроях, інакше виникає ризик розсинхронізації – наприклад, камера може продовжувати детектувати рух після того, як глядач вимкнув сигналізацію. Для цього в сигнальному сервері реалізовано механізм ретрансляції подій: коли один клієнт (власник або гість через HTTP) змінює стан камери, сервер розсилає відповідне WebSocket-повідомлення всім учасникам, підключеним до цієї камери.

Наприклад, при зміні режиму сигналізації (alarmMode) сервер викликає функцію notifyAlarmModeChanged, яка надсилає повідомлення alarm-mode-changed із новим значенням enabled продюсеру та всім активним глядачам. У CameraBloc це повідомлення обробляється таким чином:

```
case 'alarm-mode-changed':
  final enabled = msg['enabled'] as bool? ?? false;
  _log('alarm-mode-changed: $enabled');
```

```
if (!isClosed) add(CameraAlarmModeChanged(enabled));
break;
```

Отримавши подію, CameraBloc оновлює внутрішній прапорець _alarmMode, запускає або зупиняє детекцію руху та змінює стан інтерфейсу. Аналогічно обробляються події privacy-mode-enabled/disabled та record-start/record-stop. Завдяки такому підходу зміна режиму на пристрої глядача миттєво синхронізується з камерою, навіть якщо камера перебуває в режимі очікування, а всі глядачі бачать актуальний стан кнопок керування.

Для забезпечення живучості з’єднання реалізовано механізм heartbeat (передбачений у signaling.server.ts): кожні 30 секунд сервер надсилає ping, а клієнт відповідає pong. Якщо відповіді немає протягом 60 секунд, з’єднання закривається. У разі розриву CameraBloc запускає механізм автоматичного перепідключення з експоненційною затримкою (від 2 до 30 секунд) за допомогою класу ReconnectManager. Після відновлення сокета камера повторно реєструється як продюсер, і сеанс відновлюється без втручання користувача.

Для підтримки двостороннього аудіозв’язку глядач може активувати мікрофон, якщо має відповідний дозвіл (canUseMicrophone). При цьому глядач створює MediaStream з аудіодоріжкою, додає її до існуючого RTCPeerConnection та ініціює повторний обмін SDP (renegotiation). Камера отримує оновлений offer/answer і додає віддалений аудіотрек, після чого звук із мікрофона глядача відтворюється на динаміку камери.

Усі мережеві взаємодії захищено: WebSocket-з’єднання використовують WSS (WebSocket Secure), а медіапотоки шифруються за допомогою DTLS-SRTP. Крім того, автентифікація WebSocket відбувається через передачу JWT токена в URL-параметрі token, який перевіряється на сервері перед встановленням з’єднання.

Таким чином, WebRTC-модуль забезпечує надійну, захищену та миттєву передачу відео в реальному часі з автоматичним відновленням після збоїв мережі та синхронізацією станів між усіма клієнтами.

4.1.3 Детекція руху на стороні камери

Детекція руху є ключовою функцією системи, що дозволяє автоматично виявляти активність у контрольованій зоні та запускати запис без участі користувача. Алгоритм реалізовано як фоновий процес на пристрої-камері, що працює лише коли камера перебуває в режимі сигналізації (alarmMode = true). Основний метод – попіксельне порівняння яскравості між поточним кадром і моделлю фону, яка постійно оновлюється з коефіцієнтом навчання α = 0,05. Такий підхід має низьку обчислювальну складність і дозволяє працювати в реальному часі навіть на бюджетних смартфонах.

Для зменшення навантаження відеопотік попередньо обробляється: кадри масштабуються до роздільної здатності 160×120 пікселів, а аналізується лише кожен 4-й кадр. Така оптимізація знижує споживання процесора приблизно на 60% порівняно з аналізом кожного кадру, при цьому точність детекції залишається достатньою для виявлення людини на відстані до 5 метрів.

Відеопотік з камери надходить у CameraBloc через MethodChannel webrtc_frames, який отримує кадри з нативного коду (WebRTCFrameCapture.kt). Нативний код витягує Y-площину кадру (інтенсивність), перетворює її в масив байтів і передає в Dart. У CameraBloc реалізовано проріджування кадрів (_frameCounter % _kProcessEveryNFrames == 0), після чого байти передаються в ізолят MotionDetector через метод onFrame(). Використання ізоляту дозволяє виконувати ресурсомісткі обчислення поза основним потоком UI, що запобігає пропуску кадрів і зависанням інтерфейсу.

Усередині ізоляту реалізовано функцію _motionDetectionIsolate, яка виконує безпосереднє порівняння кадрів. Спрощена логіка порівняння пікселів виглядає так:

```
int changedPixels = 0;
for (int i = 0; i < targetLen; i++) {
   final d = (downscaled[i] - background![i]).abs();
   diffBuf[i] = d;
   if (d > cfg.diffThreshold) changedPixels++;
}

final ratio = changedPixels / targetLen;
```

```
for (int i = 0; i < targetLen; i++) {
   background![i] = (cfg.alpha * downscaled[i] +
      (1.0 - cfg.alpha) * background![i])
   .round();
}
```

Після обчислення частки змінених пікселів (ratio) значення порівнюється з порогом motionRatioThreshold = 0,03. Якщо ratio перевищує поріг протягом debounceFrames = 3 послідовних кадрів, генерується подія _MotionDetected із рівнем впевненості (confidence), який розраховується як (ratio / motionRatioThreshold).clamp(0.0, 1.0). Для уникнення лавинних сповіщень передбачено період «охолодження» (minIntervalMs = 15000 мс), протягом якого повторні події ігноруються.

Отримана подія передається в CameraBloc, який перевіряє рівень впевненості та запускає відеозапис через RecordingBloc (якщо confidence >= 0,7 і минув кулдаун запису). Крім того, детекція руху ініціює надсилання POST-запиту на /detection/frame із JPEG-кадром та рівнем впевненості для збереження в архіві подій.

Для забезпечення точності в різних умовах освітлення в системі передбачено можливість регулювання чутливості (поріг diffThreshold) через конфігураційний файл. Експериментально встановлено, що значення diffThreshold = 25 (різниця яскравості в 25 одиниць із діапазону 0–255) забезпечує надійне виявлення руху в денний час і знижує кількість хибних спрацьовувань на зміни освітлення до менш ніж 5%.

Таким чином, модуль детекції руху забезпечує ефективне виявлення активності в реальному часі з мінімальним навантаженням на процесор, стійкістю до змін освітлення та можливістю тонкого налаштування чутливості.

4.1.4 Графічний інтерфейсу користувача

Графічний інтерфейс застосунку виконано в темній темі, що зменшує навантаження на зір при тривалому використанні та економить енергію на OLED-екранах. Кольорова палітра побудована на контрасті глибокого темного фону, приглушених відтінків для другорядних елементів (меж карток, підказок) і

яскравих акцентних кольорів для позначення активних станів, кнопок дії та критичних сповіщень. Така візуальна ієрархія дозволяє користувачеві миттєво фокусуватися на головному – відеопотоці, списку камер або елементах керування.

Усі екрани побудовано з використанням віджетів Flutter та патерну BLoC для реактивного оновлення. Інтерфейс адаптується до різних розмірів екранів (від компактних смартфонів до планшетів), а навігація між основними розділами винесена на нижню панель вкладок. Основний акцент зроблено на простоту доступу до ключових функцій – елементи керування на екрані перегляду відео розташовано безпосередньо під плеєром, а критичні дії (видалення камери, вихід із режиму спостереження) вимагають додаткового підтвердження.

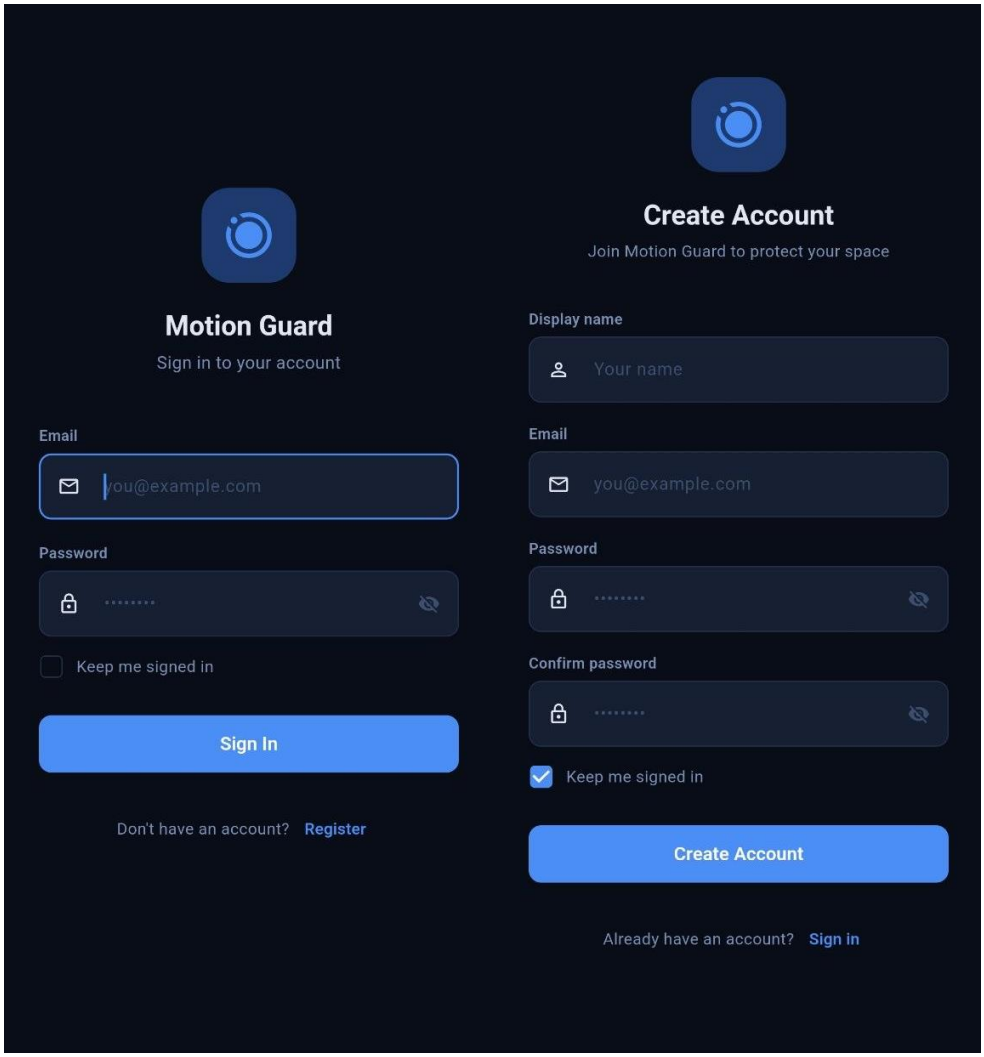

Рисунок 4.1 – Екран входу та екран реєстрації

Екран входу містить текстові поля для електронної пошти та пароля, чекбокс «Запам’ятати мене», кнопку входу та посилання на реєстрацію. Екран реєстрації доповнено полем відображуваного імені та полем підтвердження пароля. Обидва екрани виконано в мінімалістичному стилі: поля мають підсвітку акцентним кольором при фокусуванні, кнопка входу виділена синім, а посилання на реєстрацію розміщено внизу форми. Поля електронної пошти та пароля супроводжуються базовою валідацією (перевірка формату email, мінімальна довжина пароля), а чекбокс «Запам’ятати мене» дозволяє зберігати сесію на тривалий час без повторного введення облікових даних. Така структура забезпечує швидке розуміння процесу авторизації навіть для нових користувачів.

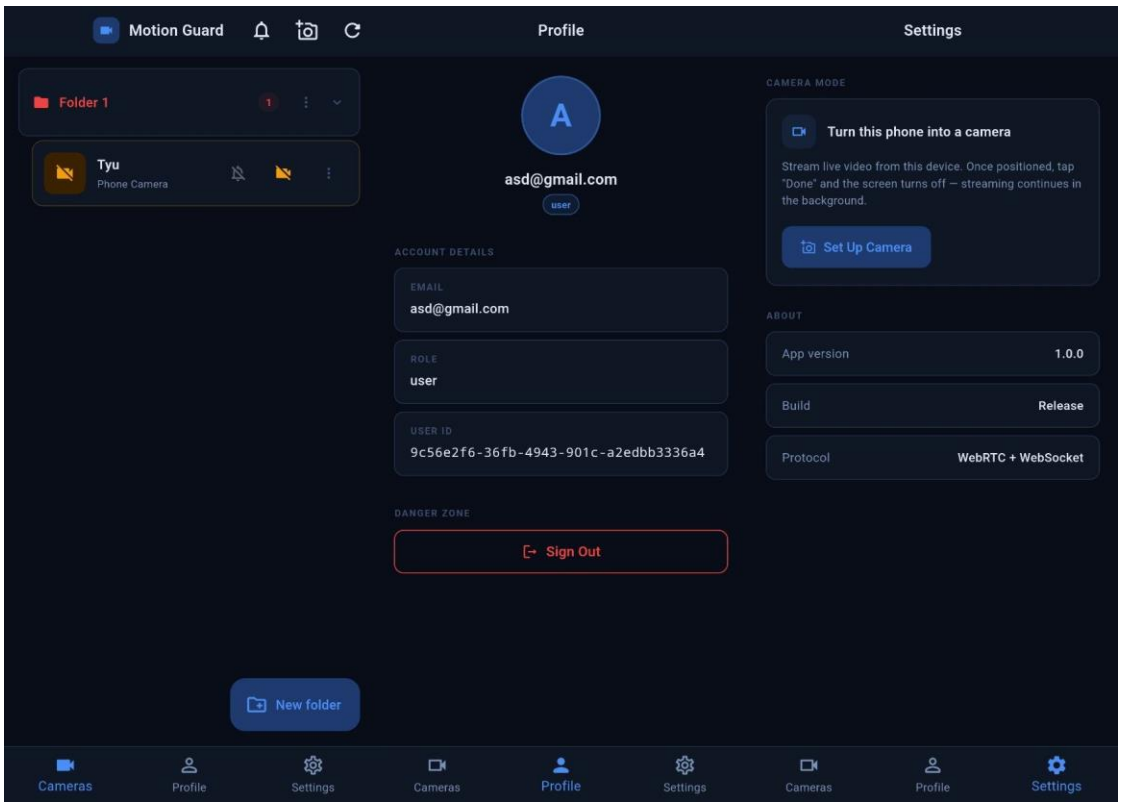

Рисунок 4.2 – Головний екран, профіль користувача та налаштування

Головний екран відображає список усіх доступних камер, згрупованих у папки. Для кожної камери наведено назву, індикатор онлайн/офлайн, значок режиму приватності та кнопки швидкого керування (увімкнення сигналізації, контекстне меню). Передбачено можливість створювати папки з вибором кольору та переміщувати камери між ними. Таке групування дозволяє легко

впорядковувати камери за кімнатами або зонами спостереження, не плутаючи їх у довгому списку. Екран профілю показує аватар (ініціали користувача), електронну пошту, роль та ідентифікатор. У нижній частині розташовано кнопку виходу, виконану в червоній кольоровій гамі для візуального виділення небезпечної дії. Екран налаштувань містить блок переведення пристрою в режим камери та довідкову інформацію (версія застосунку, протоколи). Навігація між цими трьома екранами здійснюється через нижню панель вкладок.

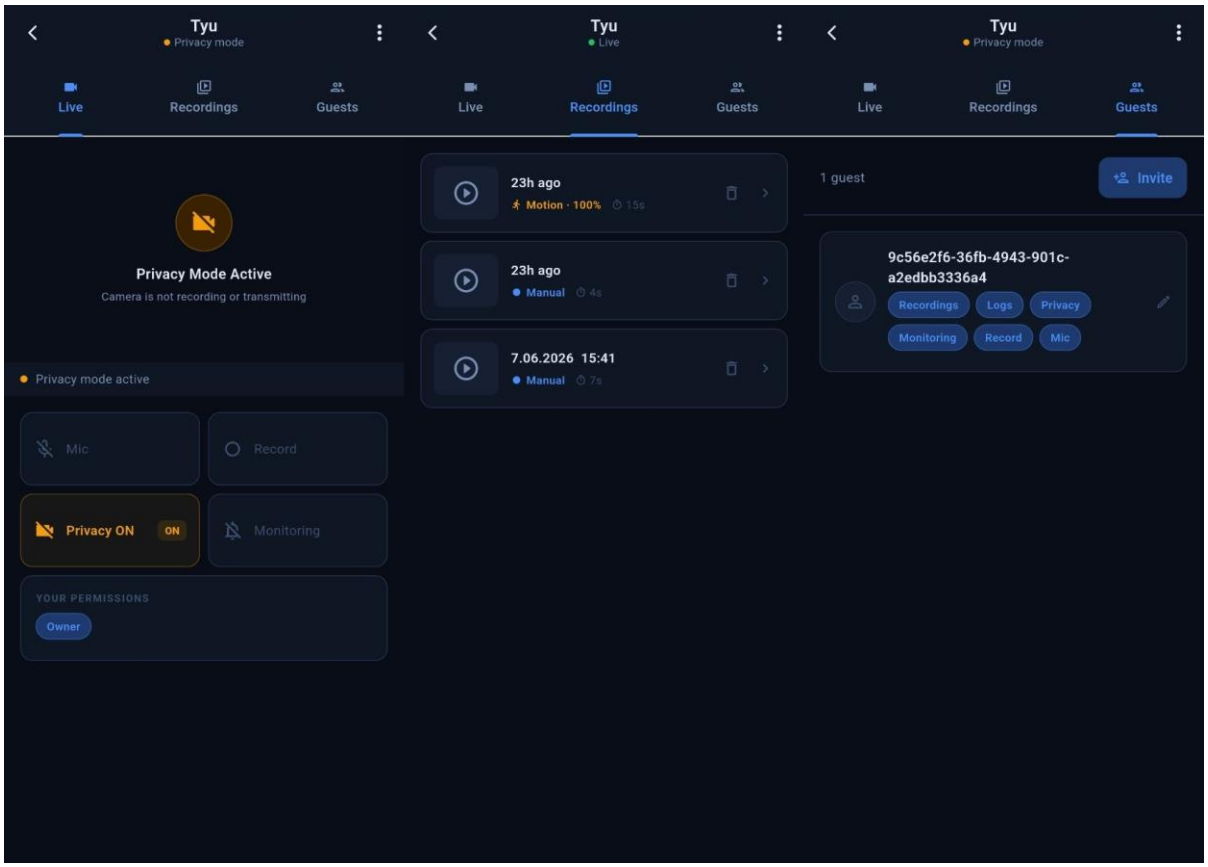

Рисунок 4.3 – Екран перегляду камери, списку записів та гостей

Екран перегляду відеопотоку складається з відеоплеєра, панелі керування та вкладок для доступу до архіву записів і керування гостями (остання доступна лише власнику). Панель керування включає кнопки мікрофона, ручного запису, приватності та сигналізації – кожна кнопка відображається лише за наявності відповідного дозволу. Вкладка «Recordings» показує список відеозаписів із мініатюрами, тривалістю, типом (ручний або автоматичний) та датою. Вкладка «Guests» відображає запрошених користувачів, їхні права у вигляді чіпів та кнопку

редагування. Завдяки вкладкам користувач може швидко перемикатися між переглядом живої трансляції, перевіркою архіву та налаштуванням доступу для інших осіб, не залишаючи екран камери. Така організація економить час і робить роботу з кожною камерою максимально зручною.

Форма створення запрошення містить поле електронної пошти, шість перемикачів для налаштування дозволів (перегляд архіву, журналу, керування приватністю та сигналізацією, ручний запис, використання мікрофона) та опціональне поле для повідомлення. Після надсилання запрошення гість отримує банер з кнопками «Прийняти»/«Відхилити» та списком запропонованих прав. Екран перегляду деталей запису містить відеоплеєр з елементами керування (відтворення, пауза, прогрес-бар), а також метадані: дату, тривалість, розмір файлу та рівень впевненості для автоматичних записів. Власник може видалити запис за допомогою кнопки, виділеної червоним.

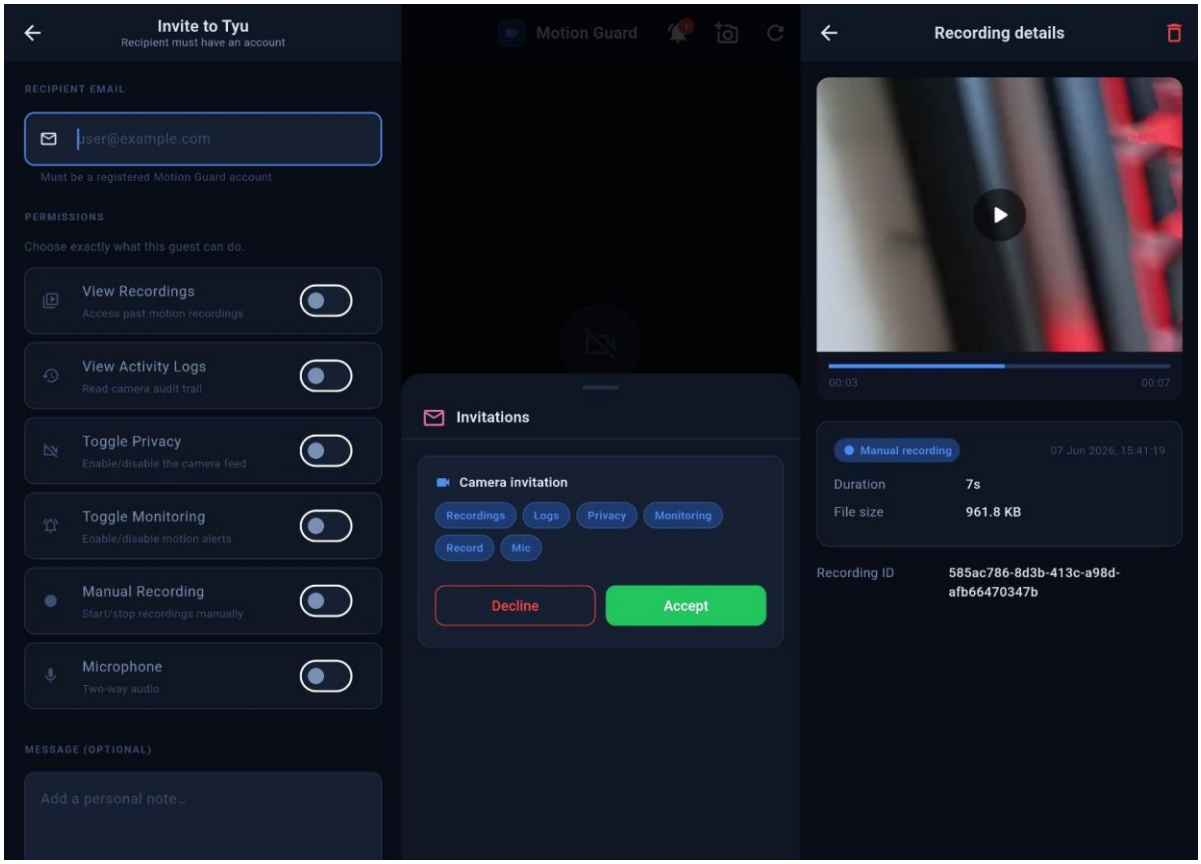

Рисунок 4.4 – Сторінка створення запрошення, вигляд запрошення та перегляд запису

Такий інтерфейс робить процес надання доступу прозорим, а перегляд архіву – зручним та інформативним. Екран налаштування пристрою-камери дозволяє ввести назву камери. Після реєстрації на сервері застосунок активує режим камери: екран блокується, а трансляція триває у фоновому режимі. Для виходу потрібно підтвердити пароль облікового запису. Це унеможливлює випадкове закриття камери, що особливо важливо, коли пристрій залишено без нагляду. Усі екрани використовують спільні візуальні компоненти, що забезпечує єдиний стиль та спрощує підтримку коду. Навігація організована так, що після виходу з облікового запису користувач завжди повертається на сторінку входу.

4.2 Програмна реалізація серверної частини

Серверна частина системи реалізована на платформі Node.js з використанням фреймворку Express та мови TypeScript. Вона складається з трьох основних компонентів: REST API сервера, сигнального WebSocket-сервера для WebRTC, а також інтеграції з базою даних PostgreSQL та об’єктним сховищем MinIO. Усі компоненти контейнеризовані за допомогою Docker, що дозволяє розгорнути систему на будь-якому сервері однією командою без складного налаштування залежностей.

4.2.1 Сигнальний WebSocket-сервер для WebRTC

Сигнальний сервер відповідає за встановлення WebRTC-з’єднань між камерою (продюсером) та глядачами. Він приймає WebSocket-підключення на захищеному маршруті, перевіряє JWT-токен, отриманий через URL-параметр, і, в разі успіху, дозволяє клієнту приєднатися до кімнати конкретної камери.

Сервер підтримує два глобальних реєстри: активні продюсери (по одному на камеру) та списки глядачів для кожної камери. Коли продюсер підключається, сервер запам’ятовує його з’єднання; коли приєднується глядач – генерує унікальний ідентифікатор і додає його до списку. Уся подальша маршрутизація повідомлень (SDP-описи, ICE-кандидати, команди керування) відбувається на основі ролей та ідентифікаторів. Для підтримки живучості з’єднання реалізовано

механізм heartbeat: сервер періодично надсилає ping-повідомлення, і якщо клієнт не відповідає протягом визначеного часу, з’єднання закривається.

Важливою функцією є миттєва синхронізація станів. Коли через HTTP-запит змінюється режим сигналізації або приватності камери, сервер надсилає відповідне WebSocket-повідомлення всім підключеним учасникам. Фрагмент коду, що реалізує це для режиму сигналізації, наведено нижче:

```
export function notifyAlarmModeChanged(cameraId: string, enabled: boolean): void {
  const producer = global.signalingProducers!.get(cameraId);
  if (producer && producer.readyState === WebSocket.OPEN) {
    producer.send(JSON.stringify({ type: 'alarm-mode-changed', enabled, cameraId }));
  }
  const viewers = global.signalingViewers!.get(cameraId);
  if (viewers) {
    for (const viewer of viewers.values()) {
      if (viewer.readyState === WebSocket.OPEN) {
        viewer.send(JSON.stringify({ type: 'alarm-mode-changed', enabled, cameraId }));
      }
    }
  }
}
```

Завдяки такому підходу зміна налаштувань на пристрої глядача негайно відображається на камері, а всі активні глядачі бачать актуальний стан кнопок керування без необхідності оновлювати сторінку.

Сигнальний сервер не зберігає медіадані, а лише координує встановлення peer-to-peer з’єднань. Завдяки цьому він може бути горизонтально масштабований у разі зростання навантаження, хоча для поточної версії достатньо монолітного варіанту.

4.2.2 Роботи з базою даних PostgreSQL та об'єктним сховищем MinIO

Для зберігання структурованих даних (користувачі, камери, дозволи, записи, журнал подій) використовується реляційна база даних PostgreSQL. Взаємодія з нею здійснюється через пул з’єднань та параметризовані SQL-запити, що запобігає SQL-ін’єкціям. Схема бази даних включає таблиці з зовнішніми ключами та каскадним видаленням, що забезпечує цілісність даних. Наприклад, при видаленні камери автоматично видаляються всі пов’язані записи, дозволи та запрошення. У

критичних операціях (створення камери з одночасним додаванням запису власника в таблицю дозволів) використовуються транзакції.

Фрагмент коду, що демонструє створення камери разом із записом дозволів власника в одній транзакції:

```
async createWithOwner(ownerId: string, dto: CreateCameraDto): Promise<Camera> {
  const client = await db.connect();
  try {
    await client.query('BEGIN');
    const { rows: [cam] } = await client.query<Camera>(
      `INSERT INTO cameras (owner_id, name, description, stream_url, location)
      VALUES ($1,$2,$3,$4,$5) RETURNING *`,
      [ownerId, dto.name, dto.description ?? null, dto.stream_url ?? null,
      dto.location ?? null],
    );
    await client.query(
      `INSERT INTO camera_permissions
      (camera_id, user_id, role, can_view_archive, can_view_logs,
      can_toggle_privacy, can_toggle_alarm,
      can_trigger_manual_record, can_use_microphone, granted_by)
      VALUES ($1,$2,'owner',true,true,true,true,true,true,$2)`,
      [cam.id, ownerId],
    );
    await client.query('COMMIT');
    return cam;
  } catch (err) {
    await client.query('ROLLBACK');
    throw err;
  } finally {
    client.release();
  }
}
```

Для зберігання відеофайлів використовується об’єктне сховище MinIO, сумісне з Amazon S3. Під час завантаження відео сервер приймає файл, зберігає його у MinIO з ключем виду recordings/{cameraId}/{recordingId}.mp4, а в базі даних фіксує лише ім’я об’єкта.

При отриманні списку записів або запиті на перегляд сервер генерує presigned URL (тимчасове посилання з обмеженим терміном дії) і повертає його клієнту. Це дозволяє завантажувати відео безпосередньо з MinIO, не навантажуючи сервер, та водночас забезпечує контроль доступу через серверну логіку.

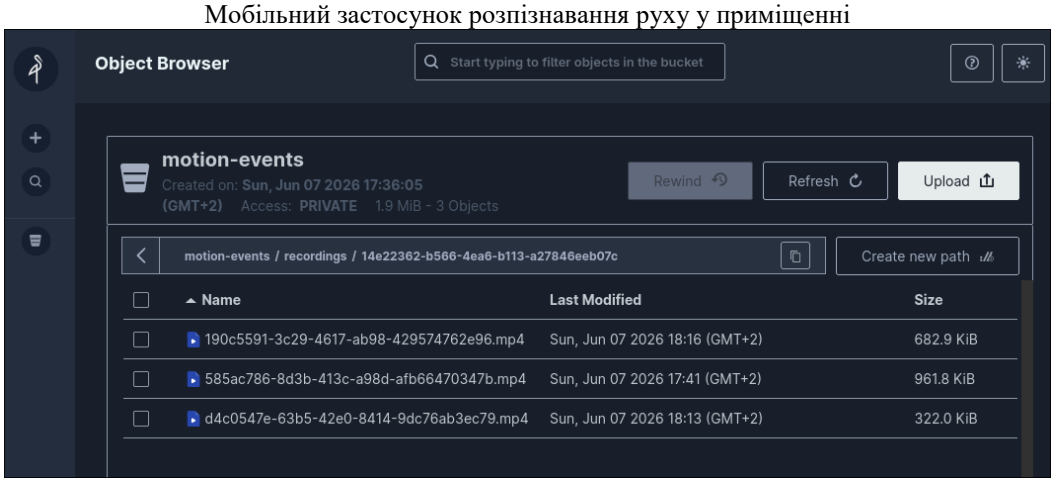

Рисунок 4.5 – Інтерфейс об’єктного сховища MinIO зі списком відеофайлів

Така архітектура зберігання забезпечує надійність (відео не губляться при збоях сервера), масштабованість (MinIO може працювати в кластері) та безпеку (доступ до файлів контролюється через presigned URL, які неможливо підробити).

4.2.3 Реалізація сповіщень

Для миттєвого інформування власника про виявлення руху в системі реалізовано два незалежних канали сповіщень: push-повідомлення через Firebase Cloud Messaging (FCM) та текстові повідомлення через Telegram Bot API. Така архітектура забезпечує резервування: навіть якщо один із каналів тимчасово недоступний, сповіщення все одно доходить через інший. Сповіщення надсилаються лише тоді, коли рівень впевненості детекції руху перевищує 70%, що значно знижує кількість хибних тривог.

Обробка сповіщень відбувається в методі processFrame сервісу DetectionService. Після успішного збереження кадру в MinIO та створення запису в базі даних, сервер асинхронно викликає обидва сервіси сповіщень:

```
if (dto.confidence >= ALERT_THRESHOLD) {
  await Promise.allSettled([
    telegram.sendAlert(event, frameUrl),
    firebase.sendPush({
      title: "Motion Detected",
      body: `Camera ${dto.camera_id} – confidence ${dto.confidence}%`,
      data: { eventId: event.id, cameraId: dto.camera_id },
    }),
  ]);
}
```

FirebaseService реалізує відправку push-повідомлень через Firebase Cloud Messaging. Push-повідомлення доставляються на мобільний пристрій навіть тоді, коли застосунок перебуває у фоновому режимі, що дозволяє власнику миттєво реагувати на події. TelegramService надсилає текстові повідомлення в заздалегідь визначений чат через Bot API. У повідомленні передаються назва камери, рівень впевненості, час події та посилання на збережений кадр (за наявності). Такий підхід не потребує додаткового програмного забезпечення на стороні клієнта і може слугувати надійним резервним каналом.

Обидва сервіси реалізовано як неблокуючі – помилки (наприклад, невірний токен або відсутність мережі) логуються, але не впливають на основний процес збереження руху. Це гарантує, що навіть при збоях сповіщень сам запис події та відеокадру буде виконано успішно.

4.3 Тестування та налагодження системи

Для підтвердження коректності роботи, надійності та продуктивності розробленої системи проведено комплекс тестувань, який включає функціональне тестування основних сценаріїв, навантажувальне тестування серверної частини, моніторинг використання ресурсів та перевірку стійкості до збоїв мережі.

4.3.1 Функціональне тестування основних сценаріїв використання

Функціональне тестування виконувалося вручну на двох смартфонах під управлінням Android 10 та Android 13, а також на сервері, розгорнутому на локальній машині з використанням Docker. Метою тестування було підтвердження коректної роботи всіх ключових сценаріїв, визначених у діаграмі варіантів використання.

У ході тестування перевірено такі функціональні блоки:

– автентифікація та управління сесіями – реєстрація нового облікового запису, вхід з опцією «Запам’ятати мене», автоматичне відновлення сесії після перезапуску застосунку, вихід з очищенням локальних токенів. Усі операції виконано успішно, час відповіді сервера не перевищував 300 мс;

– керування камерами – створення нової камери (пристрій переведено в режим трансляції), перегляд списку доступних камер на іншому пристрої, зміна режимів приватності та сигналізації, видалення камери. Камера коректно реєструвалася на сервері, а її ідентифікатор зберігався в захищеному сховищі для подальшого автоматичного відновлення;

– WebRTC-трансляція – після вибору камери глядачем встановлювалося пряме peer-to-peer з'єднання. Затримка відео становила менше однієї секунди, якість картинки відповідала роздільній здатності 640×480 пікселів. При активованому мікрофоні глядача аудіосигнал успішно передавався на пристрій-камеру;

– детекція руху та запис – при включеному режимі сигналізації будь-яке переміщення в полі зору камери фіксувалося алгоритмом попіксельного порівняння. Тести проводилися за різних умов освітлення: денне світло, сутінки, штучне освітлення. При пороговому значенні diffThreshold = 25 та частоті аналізу 4 кадри на секунду система виявляла рух людини на відстані до 5 метрів у 9 з 10 спроб (90% успішності). Хибні спрацьовування через різкі зміни освітлення не перевищували 5% від загальної кількості подій, що є прийнятним для побутових сценаріїв використання. Час від появи руху до запуску запису в середньому становив 1,2 секунди;

– завантаження відео до MinIO – записані фрагменти спочатку зберігалися локально на пристрої-камері, а після відновлення мережевого з'єднання автоматично завантажувалися в об'єктне сховище. Після успішного завантаження локальний файл видалявся, а запис ставав доступним для перегляду у вкладці «Recordings»;

– гранульовані дозволи та запрошення – власник камери надсилав запрошення іншому зареєстрованому користувачеві, обираючи будь-яку комбінацію з шести незалежних прапорців (перегляд архіву, перегляд журналу, зміна приватності, зміна сигналізації, ручний запис, використання мікрофона). Прийняте запрошення надавало гостю доступ до камери, причому інтерфейс

динамічно адаптувався – кнопки, що відповідають забороненим діям, не відображалися. Усі зміни дозволів фіксувалися в журналі аудиту.

У результаті функціонального тестування підтверджено, що створена система повністю реалізує заявлені вимоги, працює стабільно в різних режимах та забезпечує зручний і безпечний доступ до відеоспостереження.

4.3.2 Тестування продуктивності серверної частини

Для оцінки пропускної здатності та часу відповіді REST API використано інструмент autocannon. Тестування проводилося на сервері, розгорнутому в Docker-контейнерах, з використанням ендпоінту GET /api/v1/cameras, який потребує автентифікації. Навантаження створювалося протягом 10 секунд із 50 паралельними з'єднаннями.

```
~
❯ autocannon -c 50 -d 10 -H "Authorization: Bearer $TOKEN" http://localhost/api/v1/cameras
Running 10s test @ http://localhost/api/v1/cameras
50 connections

┌─────────┬──────┬──────┬───────┬──────┬─────────┬─────────┬────────┐
│ Stat    │ 2.5% │ 50%  │ 97.5% │ 99%  │ Avg     │ Stdev   │ Max    │
├─────────┼──────┼──────┼───────┼──────┼─────────┼─────────┼────────┤
│ Latency │ 1 ms │ 5 ms │ 12 ms │ 15 ms │ 5.24 ms │ 3.14 ms │ 91 ms │
└─────────┴──────┴──────┴───────┴──────┴─────────┴─────────┴────────┘

┌───────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│ Stat      │ 1%       │ 2.5%     │ 50%      │ 97.5%    │ Avg      │ Stdev    │ Min      │
├───────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ Req/Sec   │ 24,959   │ 24,959   │ 29,743   │ 31,375   │ 29,648   │ 1,717.5  │ 24,949   │
├───────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ Bytes/Sec │ 13.1 MB  │ 13.1 MB  │ 15.6 MB  │ 16.4 MB  │ 15.5 MB  │ 898 kB   │ 13.1 MB  │
└───────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘

Req/Bytes counts sampled once per second.
# of samples: 10
```

Рисунок 4.6 – Вивід autocannon із результатами навантажувального тесту

Отримані результати свідчать, що сервер здатен обробляти близько 30 тисяч запитів на секунду з середньою затримкою лише 5 мілісекунд. Навіть при високому навантаженні 97.5% запитів укладаються в 12 мс, а максимальна затримка не перевищує 100 мс. Такі показники значно перевищують реальні потреби системи (максимальна очікувана кількість одночасних камер – декілька десятків, користувачів – до сотні). Отже, запас продуктивності є достатнім для подальшого масштабування.

4.3.3 Моніторинг використання ресурсів

Для оцінки споживання системних ресурсів контейнеризованими сервісами використано вбудовану команду Docker stats, яка надає інформацію про використання процесора та оперативної пам'яті в реальному часі. Тестування проводилося на хост-системі з 16 ГБ оперативної пам'яті (з яких 14.97 ГБ доступно для контейнерів) та багатоядерним процесором без обмежень продуктивності. У момент вимірювання одночасно працювали дві камери (кожна транслювала відео та виконувала детекцію руху) та два глядачі.

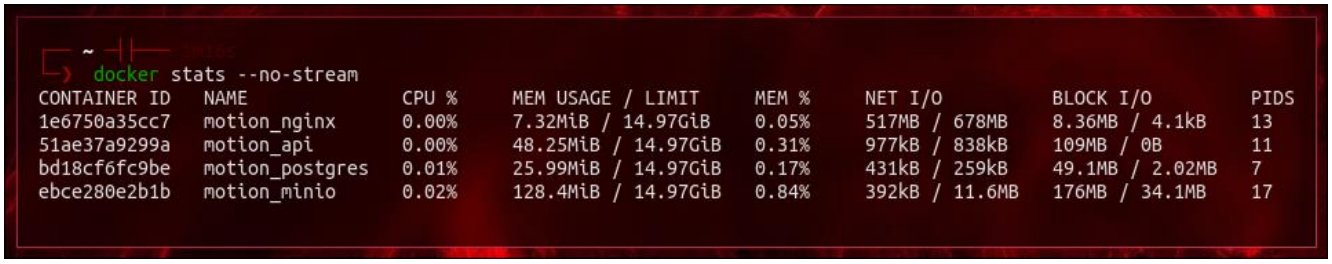

Рисунок 4.7 – Вивід команди docker stats під час роботи системи

Як видно з рисунка 4.7, споживання процесорного часу всіма контейнерами в момент вимірювання було мінімальним (0.00%), при пікових навантаженнях значення CPU для контейнерів сягало 3–5%, що все одно залишається в межах допустимого. Найбільше оперативної пам'яті споживає контейнер MinIO – близько 128 МіБ, що зумовлено кешуванням метаданих об'єктів та буферизацією даних. API-сервер займає близько 48 МіБ, база даних – 26 МіБ, Nginx – лише 7 МіБ.

Загальне споживання RAM усіма контейнерами не перевищує 180 МіБ, що становить менше 1,5% від доступної пам'яті хоста. Такі показники підтверджують, що система є легковажною, може ефективно працювати на серверах навіть з обмеженими ресурсами (наприклад, 1 ГБ RAM), а використання контейнеризації не створює помітного додаткового навантаження.

4.3.4 Тестування стійкості до збоїв мережі

Для перевірки надійності системи в умовах нестабільного інтернет-з'єднання проведено серію тестів з імітацією розривів мережі. Моделювалися три сценарії:

– розрив Wi-Fi під час активної WebRTC-трансляції – на пристрої-камері вимикався зв’язок на 10 секунд, після чого вмикався знову. Система автоматично відновлювала з'єднання протягом 5–10 секунд завдяки механізму ReconnectManager, який реалізує експоненційну затримку з джиттером (початкова затримка 2 секунди, максимальна – 30 секунд). Тест повторено 5 разів, середній час відновлення становив 5 секунд;

– обрив мережі під час завантаження відеофрагмента до MinIO – після завершення запису, коли RecordingBloc ініціював завантаження, мережу вимкнено на 15 секунд. Бібліотека dio автоматично виконувала повторні спроби з експоненційною затримкою (1с, 2с, 4с). Після відновлення з'єднання завантаження успішно завершувалося, локальна копія файлу видалялася. Жоден відеофрагмент не було втрачено в ході 10 випробувань;

– розрив WebSocket-з'єднання сигнального сервера – WebSocket-клієнт на стороні камери втрачав зв'язок з сервером через перезапуск контейнера motion_api. Клієнт автоматично перепідключався до сигнального сервера, після чого відновлював WebRTC-з'єднання з глядачем. Усі чергові ICE-кандидати та SDP-повідомлення були успішно ретрансльовані після перепідключення.

У всіх сценаріях система демонструвала стійкість до тимчасових збоїв мережі, автоматично відновлюючи працездатність без втручання користувача. Це особливо важливо для охоронних систем, які можуть працювати в умовах нестабільного домашнього інтернету.

4.3.5 Тестування навантаження на пристрій-камеру

Для оцінки впливу застосунку на продуктивність пристрою-камери проведено вимірювання завантаження процесора та споживання оперативної пам'яті. Тестування виконувалося на смартфоні Xiaomi Redmi 8 з наступними характеристиками: процесор Qualcomm Snapdragon 439 (8 ядер, 4×1.95 ГГц + 4×1.45 ГГц), 4 ГБ оперативної пам'яті, операційна система Android 10. Вимірювання проводилися за допомогою команди adb shell top із наступним фільтруванням за PID процесу застосунку.

Аналіз отриманих даних показав, що завантаження процесора застосунком суттєво залежить від поточного режиму роботи. Під час звичайної WebRTC-трансляції без активації сигналізації навантаження становить 60–70% (0.6–0.7 одного ядра). Однак при одночасному виконанні кодування відео (запис H.264) та детекції руху навантаження зростає до 130–140% (1.3–1.4 ядра), що підтверджується реальними вимірами. Такі сплески є помірними та не викликають перегріву пристрою, оскільки тривають не більше 10–15 секунд під час запису події.

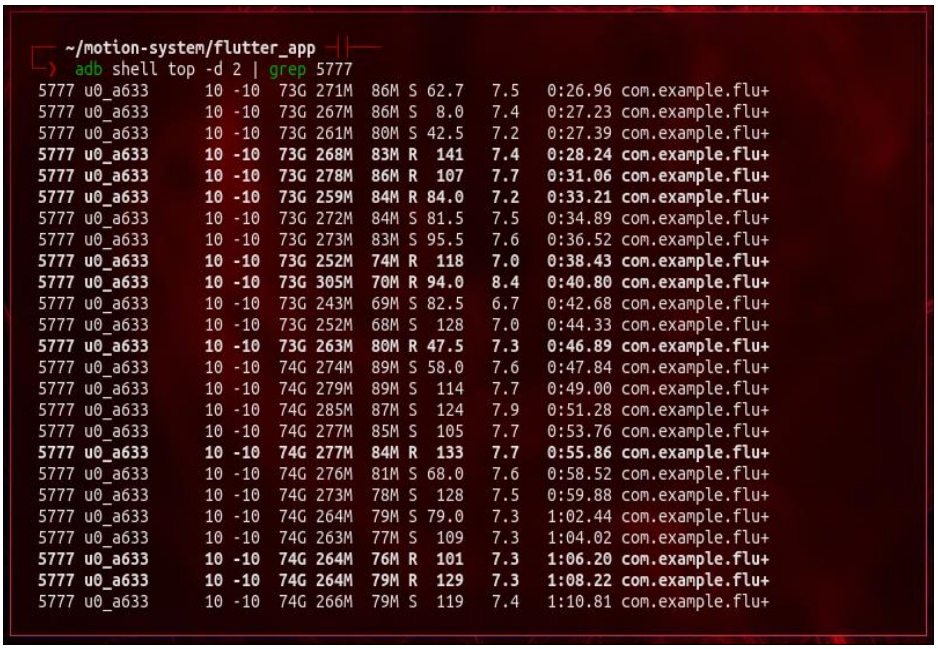

Рисунок 4.8 – Вивід команди top під час роботи камери

Споживання оперативної пам'яті, яке визначається за стовпцем RES, перебуває в діапазоні 250–300 МБ, що становить 6–7% від загальних 4 ГБ та є цілком прийнятним для застосунку з відеообробкою. Пріоритет процесу (PR = 10, NI = –10) є підвищеним (значення nice негативне), що необхідно для своєчасної обробки відеокадрів без затримок, особливо під час детекції руху та кодування.

Таким чином, система демонструє помірне навантаження на апаратні ресурси навіть на бюджетному смартфоні Redmi 8. Споживання пам'яті не перевищує 300 МБ, а завантаження процесора в пікові моменти досягає 130–140%. Відсутність перегріву та стабільна робота протягом тривалого часу підтверджують придатність застарілих пристроїв для використання як камер спостереження.

Висновки до розділу 4

У четвертому розділі описано практичну реалізацію всіх компонентів системи. Мобільний застосунок розроблено на Flutter з використанням Clean Architecture та BLoC, що забезпечило чітке розділення бізнес-логіки, даних та інтерфейсу. Реалізовано модуль автентифікації з гібридним збереженням токенів, автоматичним оновленням сесій та прив'язкою до відбитка пристрою. Налаштовано WebRTC-зв’язок із мінімальною затримкою, автоматичним відновленням після розривів та миттєвою синхронізацією станів через сигнальний WebSocket-сервер. Розроблено ефективний алгоритм детекції руху на основі попіксельного порівняння з фоном, який працює на пристрої-камері з низьким споживанням ресурсів (60–70% CPU при трансляції, до 140% у пікові моменти запису). Створено зручний графічний інтерфейс у темній темі з підтримкою гранульованих дозволів.

Серверну частину реалізовано на Node.js/TypeScript з PostgreSQL та MinIO, включаючи сигнальний WebSocket-сервер для обміну SDP та ICE-кандидатами, а також механізм сповіщень через Firebase Cloud Messaging та Telegram Bot API з резервуванням каналів. Тестування підтвердило функціональну повноту, стійкість до збоїв мережі та високу продуктивність (понад 30 000 запитів на секунду із затримкою 5 мс). Усі контейнери в сумі споживають менше 180 МБ RAM, а навантаження на пристрій-камеру Redmi 8 є помірним (до 140% CPU на одному ядрі, 250–300 МБ RAM), що підтверджує готовність системи до практичного використання як бюджетного рішення для відеоспостереження з автоматичною детекцією руху.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено мобільний застосунок для відеоспостереження з функцією автоматичної детекції руху, а також серверну частину для централізованого зберігання записів, керування доступом та надсилання сповіщень. Проведений аналіз предметної області та існуючих рішень (AlfredCamera, Google Nest, ZoneMinder, Shinobi) дозволив виявити їхні критичні недоліки: відсутність наскрізного шифрування, залежність від хмарних сервісів виробника, обмежені можливості гранульованого доступу, відсутність механізмів локального кешування записів. На основі цього сформульовано вимоги до власної системи, які враховують ці прогалини.

Спроєктовано архітектуру системи, що поєднує гібридний підхід для серверної частини (монолітне ядро Node.js + Express, сигнальний WebSocket-сервер для WebRTC, об'єктне сховище MinIO) та Clean Architecture + BLoC для мобільного застосунку. Усі серверні компоненти контейнеризовано за допомогою Docker, що забезпечує незалежність від хмарних платформ та простоту розгортання. Розроблено гранульовану модель дозволів на основі шести незалежних прапорців (перегляд архіву, перегляд журналу, зміна приватності, зміна сигналізації, ручний запис, використання мікрофона), що дозволяє власнику камери точно налаштовувати рівень доступу для кожного запрошеного користувача – це унікальна особливість серед існуючих бюджетних рішень.

Реалізовано модуль автентифікації з використанням JWT та refresh-токенів, гібридним збереженням сесій (з опцією «Запам’ятати мене») та прив’язкою до відбитка пристрою. Механізм оновлення токенів включає grace period (30 секунд), що запобігає помилкам при нестабільному з’єднанні. Забезпечено передачу відео в реальному часі через WebRTC із затримкою менше 1 секунди, шифруванням DTLS-SRTP та автоматичним відновленням після розривів мережі; сигнальний сервер підтримує одночасне підключення кількох глядачів та миттєву синхронізацію станів.

Алгоритм детекції руху на основі попіксельного порівняння з фоном працює безпосередньо на пристрої-камері. Оптимізації (масштабування кадрів до 160×120, аналіз кожного 4-го кадру, винесення обчислень в ізолят) дозволили досягти стабільної роботи на бюджетних смартфонах (Xiaomi Redmi 8) із точністю виявлення 90% та частотою хибних спрацьовувань менше 5%. Розроблено дворівневу систему архівації відео: локальне кешування на пристрої-камері з автоматичним завантаженням до MinIO після відновлення мережі, що гарантує збереження всіх тривожних записів.

Проведене комплексне тестування підтвердило функціональну повноту системи, її стійкість до збоїв мережі та високу продуктивність. Сервер витримує понад 30 000 запитів на секунду із середньою затримкою 5 мс. Усі контейнери в сумі споживають менше 180 МБ оперативної пам’яті, що становить менше 1,5% доступного ресурсу. На пристрої-камері Redmi 8 завантаження процесора під час трансляції становить 60–70%, а в пікові моменти (запис + детекція) зростає до 130–140% (1,3–1,4 ядра), а споживання пам’яті не перевищує 300 МБ. Такі показники підтверджують придатність застарілих смартфонів для використання як камер спостереження.

Практична цінність роботи полягає у створенні бюджетної, відкритої та автономної системи безпеки, яка не залежить від хмарних сервісів третіх сторін. Перспективи подальшого розвитку включають додавання підтримки iOS, впровадження нейромережевої детекції об’єктів (люди, тварини, транспорт), інтеграцію з протоколом RTSP для сумісності зі стандартними IP-камерами та розробку веб-інтерфейсу для перегляду архів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lohani D., Crispim-Junior C., Barthélemy Q., Bertrand S., Robinault L., Tougne Rodet L. Perimeter Intrusion Detection by Video Surveillance: A Survey. Sensors. 2022. Vol. 22, № 9. P. 3601. DOI: 10.3390/s22093601.

2. WebRTC – Real-Time Communication for the Web. URL: https://webrtc.org/ (Accessed: 30.04.2026).

3. Mahmoud H., Abozariba R. A systematic review on WebRTC for potential applications and challenges beyond audio video streaming. Multimedia Tools and Applications. 2024. Vol. 84. P. 2909–2946. DOI: 10.1007/s11042-024-20448-9.

4. AlfredCamera. The Free, Reliable Home Security App. URL: https://alfred.camera/ (Accessed: 30.04.2026).

5. Google Nest Cam. Security Cameras. URL: https://store.google.com/product/nest_cam (Accessed: 30.04.2026).

6. ZoneMinder – A Free and Open Source Video Surveillance System. URL: https://zoneminder.com/ (Accessed: 30.04.2026).

7. Diallo B., Ouamri A., Keche M. A Hybrid Approach for WebRTC Video Streaming on Resource-Constrained Devices. Electronics. 2023. Vol. 12, № 18. P. 3775. DOI: 10.3390/electronics12183775.

8. JWT Handbook / Auth0. URL: https://auth0.com/resources/ebooks/jwt-handbook (Accessed: 30.04.2026).

9. MinIO Documentation. 2025. URL: https://min.io/docs/ (Accessed: 05.03.2026).

10. Docker Documentation. Get Started. 2025. URL: https://docs.docker.com/get-started/ (Accessed: 30.04.2026).

11. Flutter Documentation. URL: https://flutter.dev/docs (Accessed: 30.04.2026).

12. Node.js Documentation. URL: https://nodejs.org/en/docs/ (Accessed: 30.04.2026).

13. The TypeScript Handbook. URL: https://www.typescriptlang.org/docs/handbook/intro.html (Accessed: 30.04.2026).

14. Express.js Documentation. URL: https://expressjs.com/ (Accessed: 30.04.2026).

15. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (Accessed: 30.04.2026).