

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ВІЗУАЛІЗАЦІЇ
ДЖЕРЕЛ ВІДЕОТРАНСЛЯЦІЙ ТА МЕТОДІВ ПОКРАЩЕННЯ ЇХ ЧІТКОСТІ
RESEARCH AND SOFTWARE IMPLEMENTATION OF VISUALIZATION
OF VIDEO BROADCAST SOURCES AND METHODS TO IMPROVE THEIR
SHARPNESS

Виконав: студент(ка) 2 курсу, групи КНТ-132м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ЧАБАН І.С.

(ПРИЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О. О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТЕРЕЩЕНКО Е.В.

(ПРИЗВИЩЕ та ініціали)

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

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ЧАБАНА Іллі Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація візуалізації джерел відеотрансляцій та методів покращення їх чіткості Research and Software Implementation of Visualization of Video Broadcast Sources and Methods to Improve Their Sharpness

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

Олексійович,

затверджені наказом закладу вищої освіти від “14” листопада 2023 року №437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) Протоколи потокового відео, методи збільшення чіткості зображення, відеотрансляції, користувачі системи: звичайні користувачі, яким необхідне програмне забезпечення для пошуку та візуалізації джерел відеотрансляцій з можливістю збільшення їх чіткості.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Огляд та аналіз предметної області. 2. Вибір та обґрунтування структури системи, що проєктується. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. 6. Тестування та експериментальне дослідження програми. 7. Висновки.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-6 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтролер	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 04 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	3 тиждень	Розділ 2
4	Розробка архітектури програми.	4 тиждень	Розділ 3
5	Розробка програми.	5–6 тижні	Розділ 4-5
6	Тестування та експериментальне дослідження програмного забезпечення.	7 тиждень	Розділ 6
7	Оформлення пояснювальної записки та документів до неї.	8-10 тижні	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент

_____ Ілля ЧАБАН
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Олександр СТЕПАНЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
116 с., 58 рис., 5 табл., 2 додатки, 12 джерел.

ДОСЛІДЖЕННЯ, ВІЗУАЛІЗАЦІЯ ДЖЕРЕЛ ВІДЕОТРАНСЛЯЦІЙ,
ПРОТОКОЛ, ЧІТКІСТЬ ЗОБРАЖЕННЯ, FFMPEG, RTSP, IP, C++, QML.

Об'єкт дослідження – методи пошуку та візуалізації джерел відеотрансляцій та покращення їх чіткості.

Предмет дослідження – програмне забезпечення для пошуку та візуалізації джерел відеотрансляцій та покращення їх чіткості.

Мета роботи – розробка програмного забезпечення для пошуку та візуалізації джерел відеотрансляцій та покращення їх чіткості.

Матеріали, методи та технічні засоби - мережа інтернет, мова програмування C++, декларативна мова програмування QML, протокол RTSP, персональний комп'ютер, мобільний пристрій.

Результати. Досліджено протоколи потокового відео та методи збільшення чіткості зображення, розроблено програмне забезпечення для пошуку та відображення відкритих трансляцій, що використовують протокол RTSP з можливістю збільшення їх чіткості методом згортки різкості.

Висновки. Розроблено програмне забезпечення, яке виконує пошук та відображення відкритих трансляцій, що використовують протокол RTSP з можливістю збільшення їх чіткості методом згортки різкості.

Галузь використання – пошук та відображення відкритих відеотрансляцій в інтернеті.

ABSTRACT

Explanatory note to the master's thesis project: 116 pages, 5 tables, 58 figures, 2 appendices, 12 sources.

EXPLORATION, VISUALIZATION OF VIDEO SOURCES, PROTOCOL, IMAGE SHARPNESS, FFMPEG, RTSP, IP, C++, QML.

The object of study – methods of finding and visualizing sources of video broadcasts and improving their clarity.

The subject of research – software for finding and visualizing broadcast video sources and improving their clarity.

The purpose of the work is to develop software for searching and visualizing sources of video broadcasts and improving their clarity.

Materials, methods and technical means – Internet network, C++ programming language, QML declarative programming language, RTSP protocol, personal computer, mobile device.

Results. Streaming video protocols and image sharpening methods were studied, software was developed for searching and displaying open broadcasts using the RTSP protocol with the possibility of increasing their clarity using the sharpening method.

Conclusions. Software has been developed that performs search and display of open broadcasts using the RTSP protocol with the possibility of increasing their sharpness using the sharpening method.

Scope - search and display of open video broadcasts on the Internet.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП.....	9
1 ОГЛЯД ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Поняття джерел відео трансляцій	10
1.2 Поняття чіткості зображення.....	10
1.3 Протоколи потокового відео.....	10
1.4 Методи збільшення чіткості зображення	16
1.5 Огляд існуючих засобів пошуку та відображення відео трансляцій.....	18
1.6 Аналіз вимог	24
1.7 Висновок за розділом 1	25
2 ВИБІР ТА ОБГРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЩО ПРОЄКТУЄТЬСЯ.....	27
2.1 Вибір мови програмування	27
2.2 Вибір середовища розробки.....	28
2.3 Вибір засобу для створення інтерфейсу	29
2.4 Вибір бібліотеки для отримання трансляції RTSP протоколу	30
2.5 Вибір бібліотеки для відтворення RTSP трансляції в інтерфейсі.....	31
2.6 Висновок за розділом 2.....	31
3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ ...	32
3.1 Опис файлів та папок програми	32
3.2 Алгоритм пошуку та відображення відеотрансляцій.....	36
3.3 Алгоритм згортки різкості кадрів відео трансляції.....	38
3.4 Розгляд класу DataManager	40
3.5 Розгляд класу FFmpeg.....	42
3.6 Розгляд класу NetworkScanner	42
3.7 Розгляд класу ResultsManager	44
3.8 Зберігання даних	46
4 КЕРІВНИЦТВО ПРОГРАМІСТА.....	50

4.1 Призначення й умови застосування програми.....	50
4.2 Характеристика програми.....	50
4.3 Звернення до програми.....	51
4.4 Початкові та вихідні дані	52
4.5 Повідомлення	52
5 КЕРІВНИЦТВО ОПЕРАТОРА	53
5.1 Призначення програми.....	53
5.2 Умови виконання програми.....	54
5.3 Виконання програми.....	54
5.4 Повідомлення оператору	74
6 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ	75
6.1 Тестування методу матриці згортки різкості	75
6.2 Тестування методу нерізкої маски	77
6.3 Тестування методу вейвлетного збільшення різкості.....	79
6.4 Висновок за розділом 6	81
ВИСНОВКИ.....	82
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	84
ДОДАТОК А Текст програми.....	85
ДОДАТОК Б Слайди презентації.....	107

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

DASH - Dynamic Adaptive Streaming over HTTP;

HLS - HTTP Live Streaming;

HTTP - Hypertext Transfer Protocol;

IP - Internet Protocol;

QML - Qt Modeling Language;

RTSP - Real Time Streaming Protocol;

RTP - Real-time Transport Protocol;

SDP - Session Description Protocol;

SRT - Secure Reliable Transport;

URL - Uniform Resource Locator;

VoD - Video on Demand;

WebRTC - Web Real-Time Communication.

ВСТУП

З розвитком технологій відеотрансляції стали невід'ємною частиною повсякденного життя, на сьогоднішній день до мережі інтернет підключено мільйони пристроїв транслуючих відео та аудіо контент з різних куточків земної кулі.

Джерелами трансляцій можуть бути як сервери, що транслують заздалегідь збережений контент, так і звичайні веб-камери, що транслують пейзажі, заходи або експерименти, все це приваблює інтерес користувачів, які прагнуть мати можливість їх знаходити та переглядати, щоб знайти те, що їх цікавить. Однак далеко не всі трансляції можна легко знайти в інтернеті, так як вони розташовані на безлічі сайтів, що ускладнює їх пошук.

Також у відображенні відеотрансляцій важливу роль відіграє її якість, адже переважною більшістю пристроїв транслуючих відео контент є камери, різного рівня якості та року випуску, що в сумі зі втратою якості при стисненні або недостатньо чіткого фокусування нерідко призводить до непривабливого для очей результату.

У зв'язку з цим виникла необхідність дослідження способів пошуку та візуалізації відеотрансляцій, що знаходяться у відкритому доступі в мережі інтернет.

Також для можливості покращити візуальну привабливість при недостатній чіткості зображення, виникла необхідність дослідження методів покращення чіткості зображення, які підходять для обробки відеотрансляцій.

На основі проведених досліджень буде розроблено програмне забезпечення для пошуку та візуалізації відеотрансляцій з можливістю збільшення чіткості зображення, що відображається.

1 ОГЛЯД ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття джерел відео трансляцій

Джерело відеотрансляції - це фізичний пристрій, програмне забезпечення або платформа, що дозволяє передавати відеоконтент у режимі реального часу через інтернет. Це можуть бути веб-камери, професійне обладнання для відеовиробництва, мобільні додатки для стрімінгу, комп'ютери, сервери, онлайн-платформи для трансляцій та інші технічні засоби, що використовуються для створення та розповсюдження відеоконтенту в мережі інтернет.

1.2 Поняття чіткості зображення

Чіткість зображення - це ступінь ясності та чіткості деталей у зображенні. Це характеристика, яка визначає, наскільки добре контури об'єктів на зображенні помітні і наскільки деталі зображення виглядають ясними і чіткими. Чіткість зображення залежить від кількох факторів, таких як фокусування, роздільна здатність камери або дисплея, якість оптики та інші технічні параметри.

1.3 Протоколи потокового відео

Протоколи потокового відео - це певні стандартизовані правила та методи, що дозволяють розбивати відеофайли на дрібніші фрагменти та доставляти їх кінцевим користувачам для подальшого збирання та відображення.

Фрагменти медіаконтенту перед транспортуванням стискаються, цей процес досягається за допомогою кодеків, найпоширеніший H.264. Перед передачею стисненого фрагмента він зберігається в контейнерному форматі, такому як mp4 або avi. Джерелом відеофайлу прямої трансляції може бути

камера або статичні файли у разі відео на запит (VoD) [1].

1.3.1 HLS

HLS - це протокол передачі відео з адаптивним бітрейтом. Спочатку розроблений компанією Apple для систем Apple.

Протокол HLS розбиває медіапотік на дрібніші фрагменти та створює спеціальний файл, який називають маніфестом. Плеєр періодично запитує маніфест, і якщо у файлі з'являються нові фрагменти, то вони завантажуються та відтворюються. Завдяки адаптивному бітрейту якість відео підлаштовується залежно від розміру екрану та пропускної спроможності інтернет-з'єднання.

У минулому HLS підтримував такі медіакодеки, як h.264, AAC та MP3. Нещодавно до цього списку був доданий відеокодек h.265; аналогічно працюють протоколи MPEG-DASH, Microsoft Smooth Streaming та Adobe HTTP Dynamic Streaming [2].

1.3.2 MPEG-DASH

MPEG-DASH – це технологія адаптивного потокового мовлення, що забезпечує потокову доставку мультимедійного контенту через Інтернет за протоколом HTTP. Це перше рішення потокової передачі з адаптивним бітрейтом, визнане міжнародним стандартом.

DASH це технологія потокової передачі мультимедіа з адаптивним бітрейтом. Потокове мультимедіа може надсилатися через веб-сервери HTTP.

У 2012 році ISO/IEC опублікував стандарт MPEG-DASH, заснований на технології адаптивного потокового мовлення за протоколом HTTP, і він офіційно став міжнародним стандартом [3].

MPEG-DASH дозволяє використовувати безліч стандартів кодування, щоб бути незалежними від кодеків. Він підтримує такі популярні кодеки, як H.264, H.265, VP9, VP10 і т.д.

1.3.3 WebRTC

WebRTC - це проєкт з відкритим вихідним кодом, призначений для передачі поточкових даних між браузерами та іншими програмами, що їх підтримують, за технологією "точка-точка". Технологія стандартизована на рівні API консорціумом W3C та на рівні протоколу спільнотою IETF [4].

Технологія WebRTC являє собою набір API та мережових протоколів, що дозволяють користувальницьким терміналам обмінюватися даними. API дозволяють користувачам створювати власні рішення на основі цього стандарту та існуючої інфраструктури [5].

1.3.4 SRT

Протокол SRT - це протокол потокової передачі відео з відкритим вихідним кодом, який набув широкого поширення завдяки здатності забезпечувати надійну доставку відео з низькою затримкою. Розроблений Haivision протокол SRT призначений для вирішення проблем доставки відео по непередбачуваних мережах, включаючи Інтернет. Він використовує комбінацію передових методів виправлення помилок, відновлення пакетів, контролю навантаження та шифрування для передачі високоякісних відеопотоків по будь-якій мережі, навіть за наявності втрати пакетів та коливань пропускнуєї спроможності.

Також SRT це протокол транспортного рівня, який працює поверх UDP, який є протоколом без встановлення з'єднання, який не гарантує доставки пакетів. Протокол SRT додає до UDP функції надійності та безпеки, що робить його придатним для програм потокової передачі відео в реальному часі з низькою затримкою. Протокол SRT відрізняється високою гнучкістю та налаштованістю, дозволяючи користувачам налаштовувати параметри протоколу для оптимізації доставки відео відповідно до їх конкретного варіанту використання.

Однією з ключових особливостей протоколу SRT є його здатність адаптуватися до умов мережі, що змінюються, в режимі реального часу. Протокол SRT постійно відстежує продуктивність мережі та коригує її параметри для підтримки стабільного та надійного відеопотоку. Це робить його ідеальним для додатків, де умови мережі непередбачувані, таких як прямі трансляції спортивних подій, трансляції новин або віддалена спільна робота.

Протокол SRT також забезпечує наскрізне шифрування, гарантуючи, що відеопотоки захищені від перехоплення чи несанкціонованого доступу. Це особливо важливо у додатках, де контент є чутливим чи конфіденційним, таких як корпоративні повідомлення, медичні консультації чи засідання уряду [6].

1.3.5 RTSP

Потоковий протокол реального часу RTSP - прикладний протокол, призначений для використання в системах, що працюють з мультимедійними даними і дозволяє віддалено управляти потоком даних із сервера, надаючи можливість виконання команд, таких як запуск, призупинення та зупинка мовлення мультимедійного вмісту, а також доступу до файлів за часом, розташованих на сервері.

RTSP не виконує стиснення, а також не визначає метод інкапсуляції мультимедійних даних та транспортних протоколів. Передача поточкових даних як така перестала бути частиною протоколу RTSP. Більшість серверів RTSP використовує для цього стандартний транспортний протокол реального часу, що здійснює передачу аудіо та відео.

За принципом роботи та операціями, протокол RTSP схожий на HTTP. Проте між цими протоколами є кілька значних відмінностей. При використанні протоколу RTSP, клієнт і сервер можуть здійснювати запити. Як один із варіантів, сервер, який транслює відео може надсилати запити про необхідність встановлення параметрів відображення конкретного відеопотоку. У протоколі RTSP також передбачається, що керування станом або з'єднанням виконує

сервер, на відміну від HTTP, до якого це ніяк не стосується. Також при використанні RTSP дані можуть передаватися іншими протоколами за межами основної лінії, наприклад, за протоколом RTP, що не підтримує HTTP.

У RTSP протоколі сповіщення передаються окремо від потоку мультимедіа. Для надсилання повідомлень використовується стандартно виділений порт з номером 554. Запит на сервер надсилається у форматі: метод «<абсолютна_адреса>[/медіавміст] <версія_протоколу>», у текстовому форматі. На нових рядках разом із запитом можуть передаватися додаткові службові поля [7].

Методи протоколу:

- play – запит для запуску мовлення;
- announce – оновлення інформації яка описує вміст;
- teardown - завершення мовлення та звільнення ресурсів;
- describe - запит інформації про вміст, наприклад, у SDP форматі;
- redirect – переадресація на інший вміст;
- set_parameter – передача параметрів серверу, які необхідно встановити;
- options - запит отримання підтримуваних методів;
- setup – запит для встановлення механізму транспортування вмісту;
- pause – запит для припинення мовлення вмісту;
- get_parameter - запит серверу для отримання зазначених параметрів;
- record – запит серверу для запису вмісту.

Приклад запиту URL: «rtsp://example.com/video/test.mpg/streamid=0 RTSP/1.0».

Присутність стандартного порту дозволяє легко знаходити трансляції в інтернеті. За даними сайту Shodan у глобальній мережі інтернет функціонує близько 4,053,936 джерел відео трансляцій (рис. 1.1).

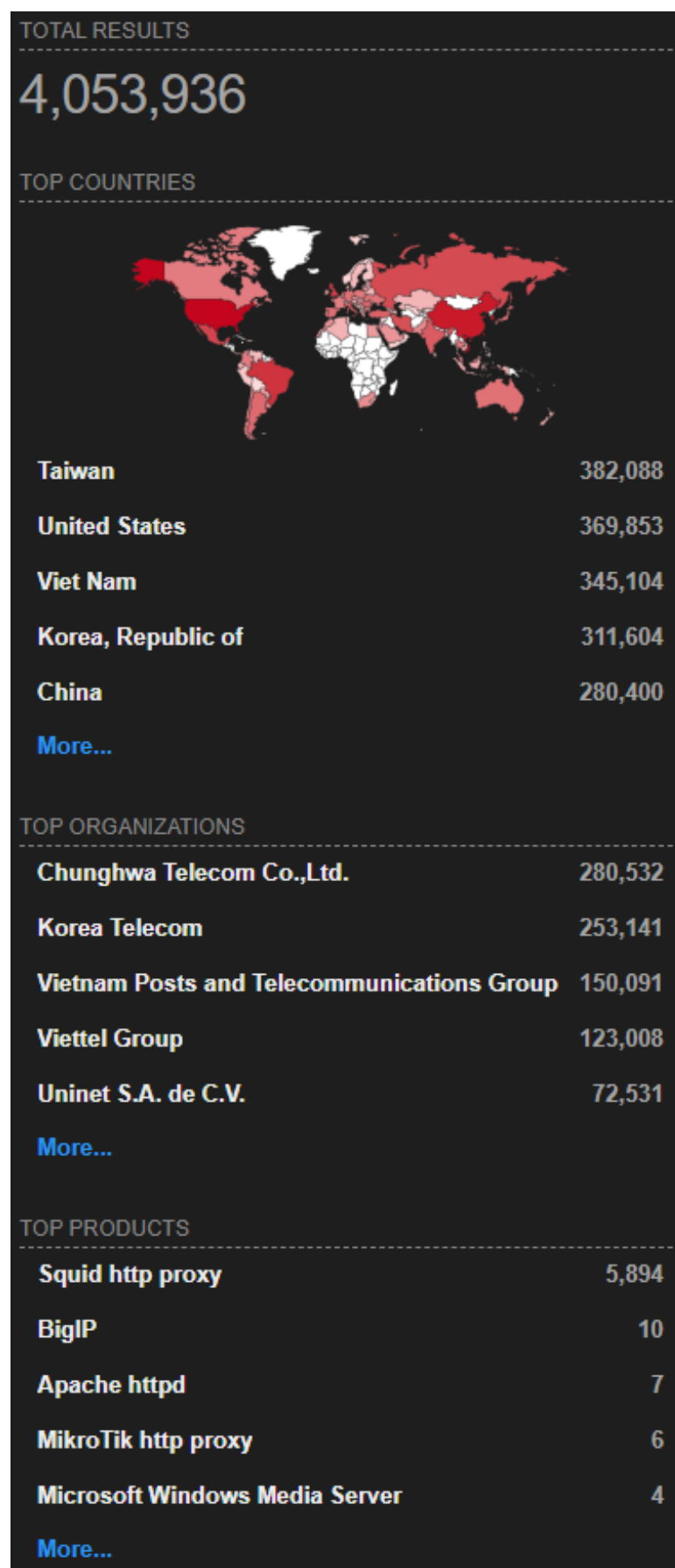


Рисунок 1.1 – Результат пошукового запиту у Shodan по відкритому 554 порту та протоколу RTSP

1.4 Методи збільшення чіткості зображення

1.4.1 Матриця згортки чіткості

Матриця згортки, що використовується для збільшення різкості зображення, називається матрицею згортки різкості чи матрицею згортки для операції збільшення різкості (sharpening kernel). Ця матриця є шаблоном, який застосовується до кожного пікселя зображення за допомогою операції згортки. Застосування цієї матриці дозволяє посилити високочастотні компоненти зображення, що робить його різкішим.

Зазвичай матриця згортки різкості має розмір 3 на 3, значення яких зображено на рисунку 1.2.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Рисунок 1.2 – Матриця згортки 3x3 для збільшення різкості

У матриці центральний елемент (значення 5) зазвичай має позитивне значення оточене негативними значеннями. Застосування такої матриці до пікселя зображення призводить до збільшення контрасту та різкості на піксельному рівні.

Згортка ядра та зображення виражається формулою:

$$g(x, y) = \omega * f(x, y) = \sum_{\Delta x=-a}^a \sum_{\Delta y=-b}^b \omega(\Delta x, \Delta y) f(x - \Delta x, y - \Delta y) \quad (1.1)$$

де $g(x, y)$ – зображення після застосування згортки;

ω – ядро згортки;

$f(x, y)$ – початкове зображення;

a, b – довільні межі згортки;

Операція буде застосована в діапазоні від $-a$ до a і от $-b$ до b для координат x та y відповідн.

1.4.2 Нерізка маска

Метод збільшення різкості зображення з використанням розмиття Гауса (Gaussian Blur) часто називається методом збільшення різкості з контрастним посиленням (Unsharp Masking), і він може бути описаний наступним чином:

- на копії зображення застосовується операція розмиття Гауса за допомогою фільтра Гауса. Розмиття Гауса згладжує зображення та видаляє високочастотні деталі, такі як шум та контрастні краї. Розмиття виконується з метою отримати більш гладку і менш різку версію зображення. Розмір та стандартне відхилення фільтра Гауса можуть бути налаштовані залежно від бажаного ступеня розмиття;
- створення маски різниці шляхом віднімання розмитої версії зображення з вихідної копії. Ця маска містить інформацію про високочастотні деталі, видалені розмиттям;
- масштабування маски різниці та накладення її до вихідного зображення, що посилює контраст і збільшує різкість зображення.

1.4.3 Вейвлетне збільшення різкості

Вейвлетне збільшення різкості (Wavelet Sharpening) – це метод обробки зображень, який використовує вейвлет-перетворення для покращення різкості зображення. У контексті збільшення різкості метод вейвлетного збільшення різкості застосовується для посилення високочастотних компонентів зображення, що робить його більш чітким і деталізованим.

Процес починається із застосування вейвлет-перетворення до вихідного зображення. Це перетворення поділяє зображення на кілька масштабів та орієнтацій. Воно відокремлює високочастотні компоненти, такі як краї та деталі

від низькочастотних, гладких областей зображення.

Після вейвлет-перетворення високочастотні компоненти посилюються або розширюються. Це призводить до збільшення величини високочастотних коефіцієнтів за відносно незмінних низькочастотних коефіцієнтів. Таке посилення підкреслює краї та деталі зображення.

Покращені високочастотні компоненти об'єднуються з вихідними низькочастотними компонентами за допомогою зворотного перетворення вейвлету. Цей крок відновлює підвищену різкість зображення на основі змінених коефіцієнтів.

1.5 Огляд існуючих засобів пошуку та відображення відео трансляцій

1.5.1 Shodan

Shodan — пошукова система, яка дозволяє користувачам шукати різні типи серверів (веб-камери, маршрутизатори, сервери тощо), підключених до Інтернету, з використанням різних фільтрів. Деякі також описують її як пошукову систему сервісних банерів, що є метаданими, які сервер відправляє назад клієнту. Цими метаданими можуть бути інформація про серверне програмне забезпечення, які опції підтримує сервіс, вітальне повідомлення або ще щось, що клієнт повинен з'ясувати перед взаємодією з сервером.

Сайт за допомогою веб-краулерів сканує Інтернет у пошуках загальнодоступних пристроїв. В даний час Shodan відображає 10 результатів пошуку користувачам без облікового запису та 50 результатів тим, у кого він є.

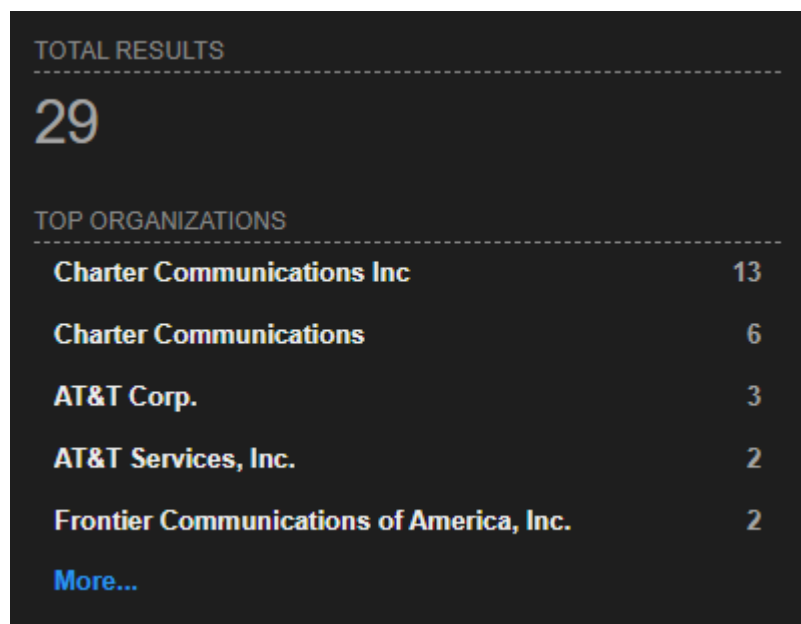
Shodan збирає дані головним чином про веб-сервери HTTP/HTTPS (порти 80, 8080, 443, 8443), а також FTP (порт 21), SSH (порт 22), Telnet (порт 23), SNMP (порт 161), IMAP (порти 143 або зашифрований 993), SMTP (порт 25), SIP (порт 5060) та RTSP (порт 554). Останній протокол може використовуватися для доступу до веб-камер та їх потоку відео [8].

Для пошуку відкритих трансляцій RTSP можна використовувати

пошуковий запит у форматі: “port:554 has_screenshot:true country:us city:Los-Angeles ”, де:

- port:554 - директива пошуку по відкритому порту;
- has_screenshot:true - директива пошуку для видачі результатів, що містять знімок трансляції;
- country:us - директива пошуку за кодом країни за стандартом ISO 3166-1;
- city: Los-Angeles - директива пошуку по місту латинськими літерами.

При введенні запиту “port:554 has_screenshot:true country:us city:Los-Angeles ” було знайдено 29 трансляцій RTSP (рис. 1.6-1.7).



The image shows a screenshot of a Shodan search results page. At the top, it says 'TOTAL RESULTS' followed by a large number '29'. Below this, there is a section titled 'TOP ORGANIZATIONS' which lists several organizations and the number of results found for each. The organizations listed are Charter Communications Inc (13), Charter Communications (6), AT&T Corp. (3), AT&T Services, Inc. (2), and Frontier Communications of America, Inc. (2). There is a 'More...' link at the bottom of the list.

TOTAL RESULTS	
29	
TOP ORGANIZATIONS	
Charter Communications Inc	13
Charter Communications	6
AT&T Corp.	3
AT&T Services, Inc.	2
Frontier Communications of America, Inc.	2
More...	

Рисунок 1.6 – Результат пошукового запиту у Shodan за запитом “port:554 has_screenshot:true country:us city:Los-Angeles”

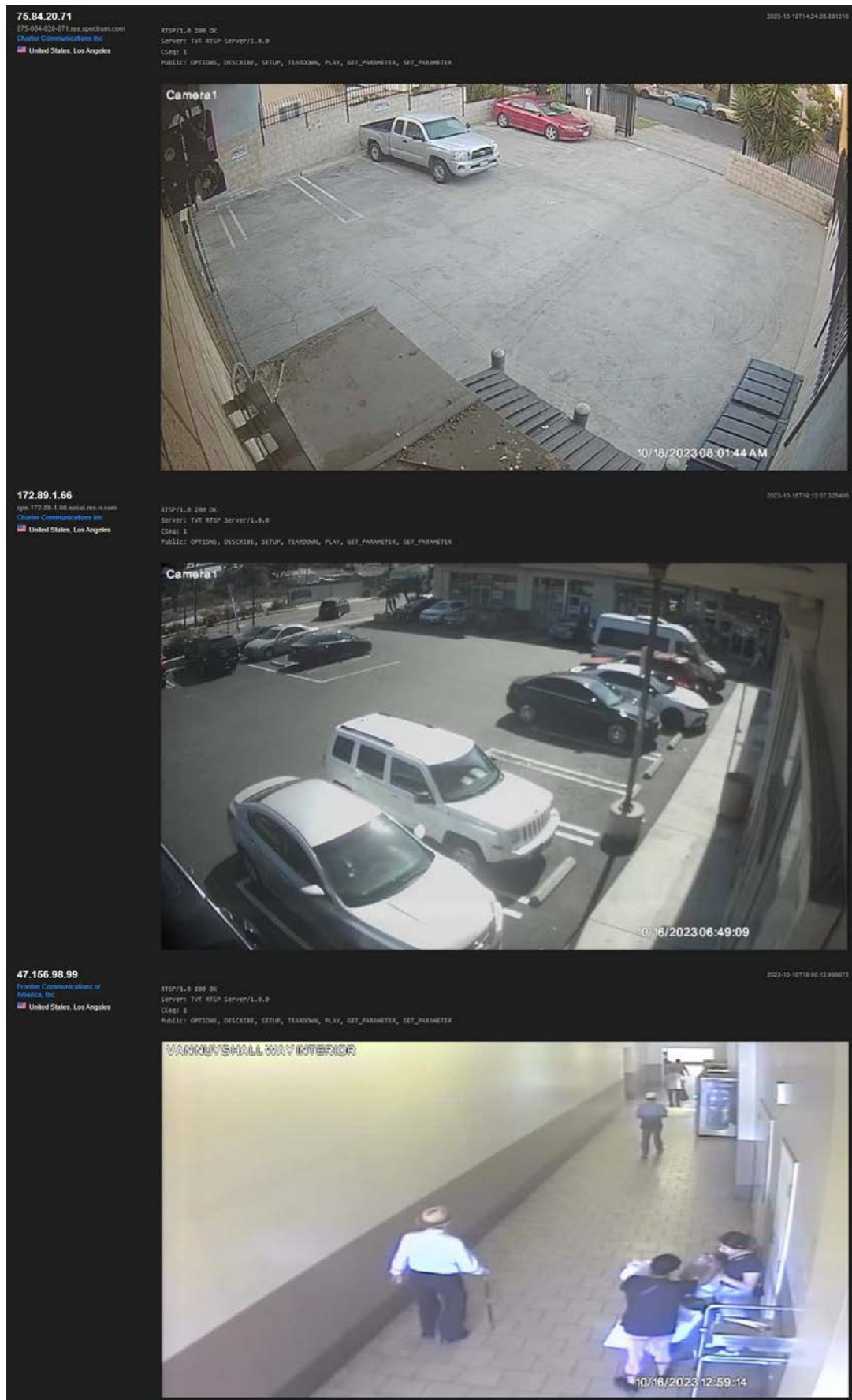


Рисунок 1.7 – Знімки RTSP трансляцій знайдених за запитом “port:554 has_screenshot:true country:us city:Los-Angeles”

1.5.2 Google

Пошукова система Google дозволяє шукати трансляції за допомогою певних запитів, таких як: "inurl:/mjpg/video.mjpg" (рис. 1.8-1.9).

Під час використання запиту "inurl:/mjpg/video.mjpg" у Google виконується пошук веб-камер, які використовують технологію Motion JPEG (MJPEG) для трансляції відео. MJPEG є форматом відео, який зберігає кожен кадр як окреме зображення JPEG. Це дозволяє отримати високу якість відео, але потребує великої пропускної спроможності.

Щоб використати цей запит, потрібно відкрити пошукову систему Google і ввести в пошуковий рядок "inurl:/mjpg/video.mjpg". Натиснути клавішу Enter або кнопку "Пошук" і Google почне шукати веб-камери, які транслюють відео через файл video.mjpg.

Запит "inurl" є директивою пошуку, яка вказує Google шукати URL-адреси, які містять певні ключові слова. У цьому випадку ключове слово "/mjpg/video.mjpg" вказує на файл video.mjpg, який використовується для трансляції відео на веб-камері.



Рисунок 1.8 – Знімок трансляції знайденої пошуковою системою Google

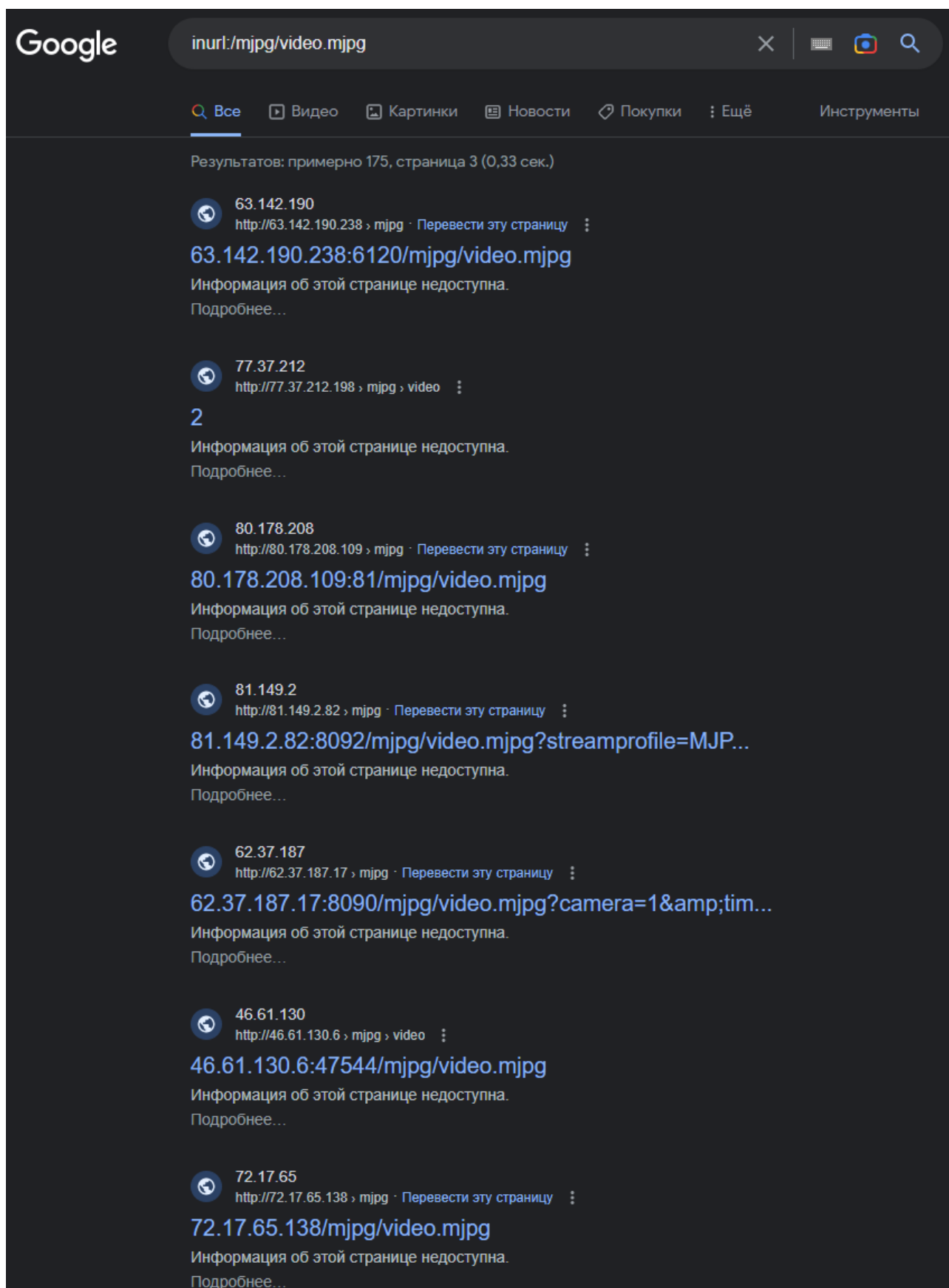


Рисунок 1.9 – Результат поискового запиту у Google за запитом “
inurl:/mjpg/video.mjpg”

1.5.3 Insecam

Сайт Insecam.org був запущений як проєкт, який збирає відео-потoki з камер спостереження встановлених у різних частинах світу та надання доступу до цих потоків в інтернеті (рис. 1.10-1.11).

Відображення відео потоків здійснюється у форматі Motion JPEG (MJPEG).

Сайт надає можливість пошуку трансляцій по локації, часовому поясу та виробникам камер.

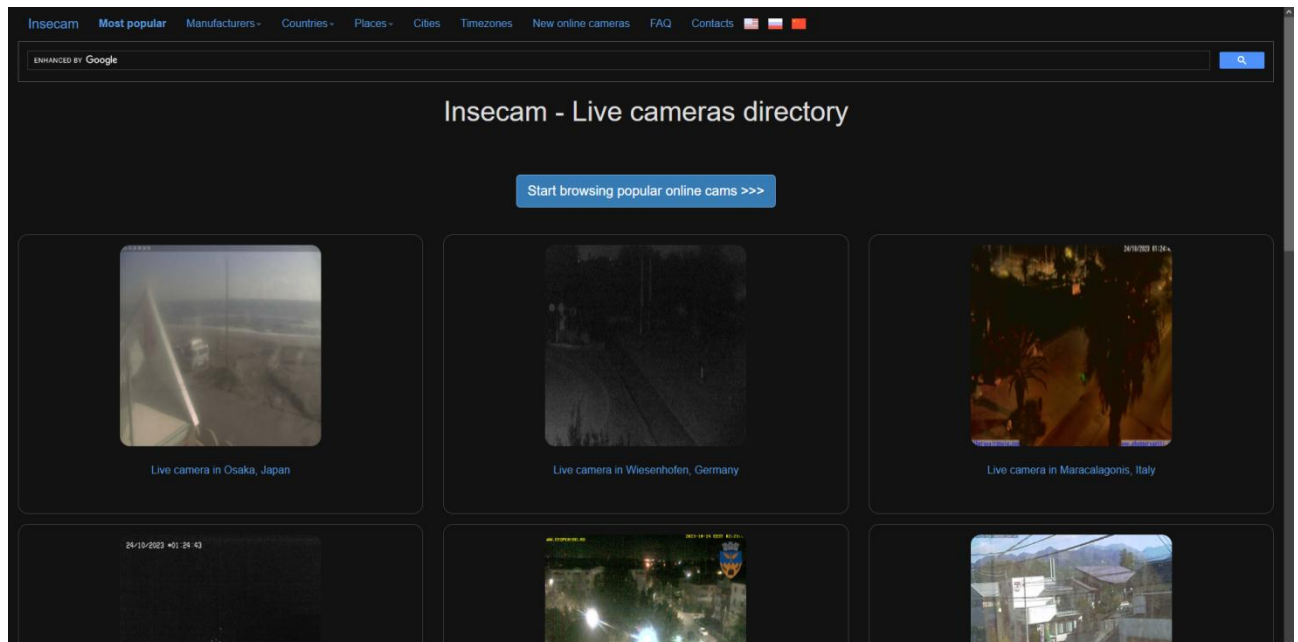


Рисунок 1.10 – Головна сторінка сайту Insecam

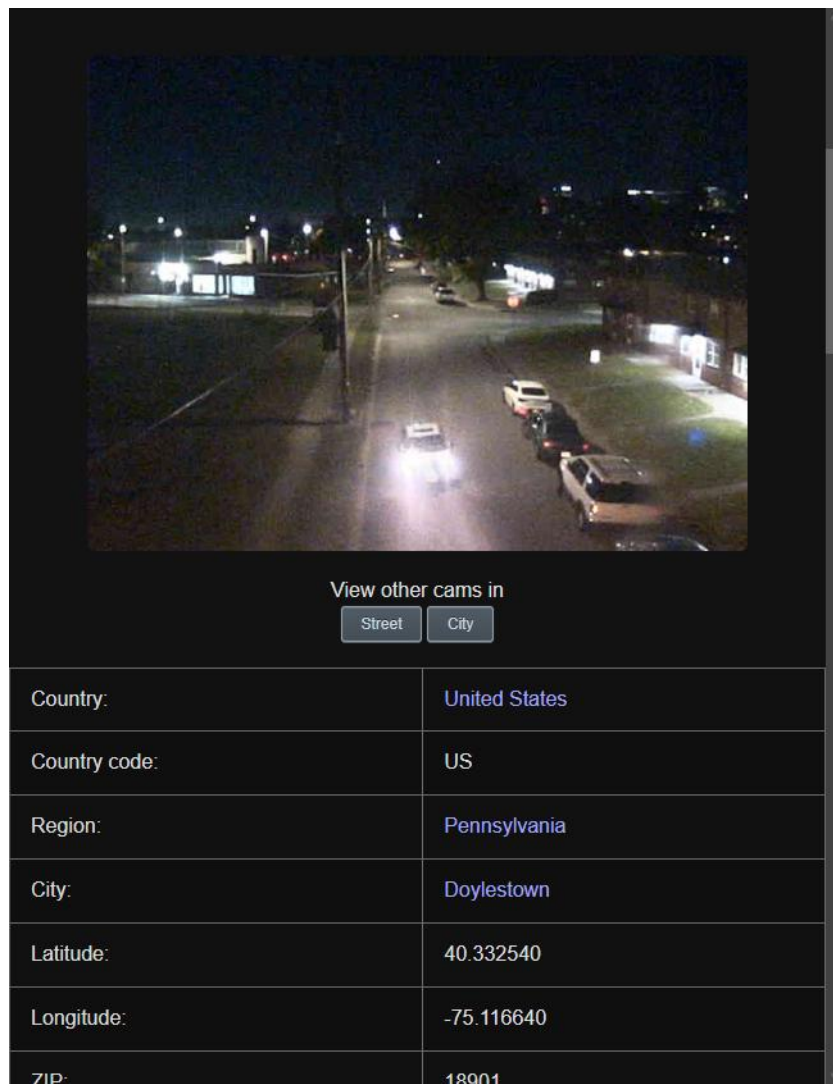


Рисунок 1.11 – Сторінка сайту Insecam з відображенням вибраної трансляції

1.6 Аналіз вимог

У результаті дослідження предметної області було сформовано оптимальні функціональні та нефункціональні вимоги до розробляемого програмного забезпечення. Ці вимоги призначені забезпечити тривалу актуальність застосунку та підтримувати його конкурентоспроможність у майбутньому серед потенційних аналогів.

Нефункціональні вимоги:

- програмний продукт повинен підтримувати хоча б одну з популярних операційних систем, таких як Android або Windows;
- виконувати пошук відкритих трансляцій RTSP;

- надавати можливість збільшення чіткості трансляції, що відображається;

- використовувати мінімальну кількість ресурсів пристрою;
- мати простий та зрозумілий графічний інтерфейс.

Функціональні вимоги:

- приймати діапазон ір-адрес;
- вибір кількості потоків для виконання пошуку;
- пошук відкритих RTSP-трансляцій;
- можливість призупинення процесу пошуку;
- можливість завершення процесу пошуку із збереженням результатів;

- відображення інформації про виконання процесу пошуку;
- можливість виведення зображення з RTSP-трансляцій;
- можливість підвищення чіткості трансляції, що відображається;
- перегляд результатів пошуку;
- оновлювати знімок трансляції;
- оновлювати знімки трансляцій за заданими параметрами;
- отримувати URL-адресу трансляції;
- переглядати історію знімків;
- видалення результатів пошуку;
- очищати історію знімків;
- додавати трансляцію до обраних.

1.7 Висновок за розділом 1

У розділі були розглянуті результати дослідження протоколів потокового відео, методів збільшення чіткості зображення та існуючих засіб для пошуку та відображення відеотрансляцій.

Найбільш підходящим протоколом для вирішення завдання пошуку та відображення джерел відеотрансляцій є протокол RTSP, завдяки стандартно

виділеному 554 порту і великій кількості пристроїв які підтримують трансляцію по даному протоколу.

Також були складені вимоги до програмного забезпечення.

2 ВИБІР ТА ОБГРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЩО ПРОЄКТУЄТЬСЯ

2.1 Вибір мови програмування

Програма буде реалізована на мові C++ з використанням фреймворку QT.

C++ є однією з найпопулярніших мов програмування у світі. Поєднання продуктивності, потужності, кросплатформенності та розширюваності мови C++ робить її ефективним інструментом для розробки застосунків, особливо у поєднанні з Qt Framework. Переваги вибору мови програмування C++ та Qt Framework для розробки застосунків:

- висока продуктивність, C++ - це компілювана мова програмування, що означає, що код компілюється в машинний код, який виконується безпосередньо на цільовому пристрої. Це дозволяє створювати програми з високою продуктивністю, які можуть виконуватись на пристроях з обмеженими ресурсами, такими як мобільні пристрої;
- кросплатформеність, C++ та Qt Framework дозволяють створювати кросплатформні мобільні програми, які можуть працювати на різних операційних системах, включаючи Android, Windows та інші. Це означає, що розробники можуть створювати програми, які можуть використовуватись на різних пристроях без необхідності переписування коду для кожної платформи;
- ефективне використання ресурсів пристрою, поєднання C++ та Qt Framework дозволяє створювати мобільні програми, які ефективно використовують ресурси пристрою, такі як процесор, пам'ять та батарея. Крім того, Qt Framework надає безліч інструментів для оптимізації та керування ресурсами пристрою;
- велика спільнота розробників та наявність бібліотек та фреймворків, C++ має велику спільноту розробників, що означає, що є багато ресурсів та інструментів, які можуть бути використані для розробки застосунків. Qt Framework також має широке співтовариство користувачів та

розробників, що дозволяє швидко знаходити рішення для складних завдань та швидко реагувати на зміни у розробці застосунків;

- зручність розробки інтерфейсу користувача, Qt Framework надає безліч інструментів для створення красивого і функціонального інтерфейсу користувача. Бібліотека компонентів Qt дозволяє швидко створювати кросплатформні програми з привабливим та інтуїтивно зрозумілим інтерфейсом, що спрощує розробку застосунків;

- ефективність у багатопоточності, C++ дозволяє ефективно працювати з багатопоточністю, що дозволяє розподіляти навантаження на кілька ядер процесора та підвищує продуктивність програми. Qt Framework також надає потужні інструменти для роботи з багатопоточністю, що робить розробку програм більш ефективною;

- широкі можливості розробки серверних та клієнтських застосунків, C++ та Qt Framework також можуть використовуватися для розробки серверних та клієнтських програм, таких як веб-сервери або програми для ефективної роботи з мережею та обробки даних на стороні сервера. Це може бути корисним для створення програм, що працюють у режимі клієнт-сервер, а також для програм, які потребують інтенсивної обробки даних [9].

2.2 Вибір середовища розробки

Для створення програми буде використано інтегроване середовище розробки Qt Creator.

Одними з основних переваг вибору середовища розробки Qt Creator для створення застосунків на базі Qt є:

- інтеграція з Qt framework, Qt Creator є офіційним IDE для розробки на базі Qt framework. Це означає, що розробники можуть використовувати всі переваги Qt, включаючи безліч готових модулів та бібліотек для створення кросплатформових програм;

– підтримка різних мов, Qt Creator підтримує різні мови програмування, у тому числі C++, QML, JavaScript та ін. Це дає можливість вибрати найбільш підходящу мову для вирішення конкретного завдання та створити високопродуктивну програму [10].

Порівняння середовищ розробки наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння середовищ розробки

	MS Visual Studio	Qt Creator IDE	Eclipse IDE
Вбудований відладчик	Так	Так	Так
Переносимість середовища на різні платформи	Так	Так	Так
Встановлене ПЗ на підприємстві та наявність ліцензії	Так	Так	Ні
Досвід роботи	Так	Так	Ні
Безкоштовний доступ	Так	Так	Так

2.3 Вибір засобу для створення інтерфейсу

Для реалізації графічного інтерфейсу програми використовуватиметься декларативна мова програмування QML.

QML є мовою для опису інтерфейсу користувача і його структури, який дозволяє дизайнерам і розробникам створювати програми з високою продуктивністю, плавними анімаціями і привабливим дизайном.

Синтаксис QML заснований на JSON і має гарну читальність, а також підтримує використання імперативних виразів JavaScript у поєднанні з динамічним зв'язуванням атрибутів.

Використання JavaScript дозволяє обробляти та виконувати дії, що стосуються інтерфейсу безпосередньо в ньому, що дозволяє скоротити необхідність використання коду на C++ [11].

2.4 Вибір бібліотеки для отримання трансляції RTSP протоколу

Для отримання RTSP-трансляції буде використовуватись бібліотека FFmpeg.

FFmpeg – це безкоштовна бібліотека з відкритим вихідним кодом, призначена для обробки відео та аудіоданих. Вона містить безліч інструментів для роботи з форматами і кодеками мультимедіа, включаючи RTSP. Переваги вибору бібліотеки FFmpeg для роботи з RTSP [12]:

- підтримка безлічі форматів, FFmpeg підтримує величезну кількість форматів аудіо та відео, включаючи ті, які використовуються в RTSP-потоках, такі як H.264 і MPEG-4. Завдяки цьому, FFmpeg може бути використаний для отримання та обробки даних з більшості IP-камер та відеореєстраторів, що підтримують RTSP;
- кросплатформеність, FFmpeg може бути використаний на різних операційних системах, включаючи Windows, Linux, MacOS, Android та iOS, що робить її зручною для створення кросплатформових застосунків, що працюють з RTSP;
- висока продуктивність, FFmpeg – це бібліотека з високою продуктивністю, оптимізована для роботи з мультимедіа-даними. Вона використовує оптимізований код мовою C для декодування та кодування відео та аудіоданих, що дозволяє досягати високої продуктивності навіть на пристроях з обмеженими ресурсами;
- розширюваність, FFmpeg є модульною бібліотекою, що дозволяє додавати нові формати та кодеки мультимедіа до системи. Це дозволяє використовувати FFmpeg з новими IP-камерами та відеореєстраторами, які підтримують інші формати та кодеки, без необхідності переписувати програму;
- легкість у використанні, FFmpeg надає простий та зручний API для роботи з мультимедіа-даними. Вона дозволяє розробникам легко отримувати та обробляти RTSP-потоки, а також зберігати їх у різних форматах.

2.5 Вибір бібліотеки для відтворення RTSP трансляції в інтерфейсі

Для програвання RTSP-трансляції в інтерфейсі користувача використовуватиметься проєкт з відкритим вихідним кодом “QmlAV”, це потоковий програвач QML у реальному часі на основі бібліотеки FFmpeg, доступний на веб-сервісі GitHub.

Такий вибір обумовлений відсутністю підтримки протоколу RTSP стандартним програвачем відео Qt Multimedia.

2.6 Висновок за розділом 2

У розділі було проаналізовано та обрано:

- середовище розробки Qt Creator;
- мову програмування C++;
- QML яз засіб для створення інтерфейсу;
- бібліотеку FFmpeg для отримання трансляції RTSP протоколу;
- бібліотеку QmlAV для відтворення RTSP трансляції в інтерфейсі.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Опис файлів та папок програми

На рисунку 3.1 наведено файлову структуру програмного забезпечення.

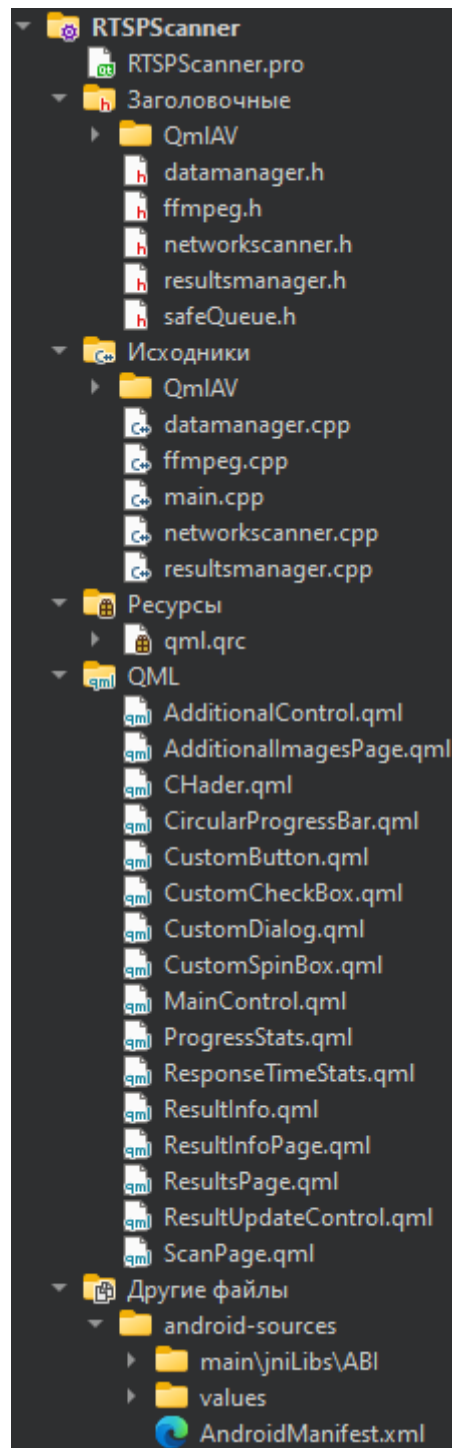


Рисунок 3.1 – Структура файлів проекту

RTSPScanner.pro - файл проєкту, який містить налаштування проєкту такі як: шляхи до додаткових бібліотек, модулі Qt framework, компоненти сторінок і самі сторінки QML.

datamanager – клас для взаємодії з даними: збереження результатів, отримання результатів, видалення результатів, отримання списку шляхів до RTSP-трансляцій, створення необхідних файлів та директорій.

ffmpeg - клас для захоплення та збереження зображення з RTSP потоку.

main - код програми, що запускається при старті, в якому виконується запуск інтерфейсу і підключення до нього необхідних компонентів.

networkscanner - клас у якому реалізований функціонал взаємодії інтерфейсу користувача сканування мережі, він реалізує наступний функціонал:

- пошук ір-адрес з відкритим 554 портом;
- перевірка наявності та доступності RTSP трансляції;
- збереження отриманих результатів, використовуючи раніше описаний клас datamanager;
- оновлення інформації про процес сканування в інтерфейсі користувача, шляхом сигналізації про зміну значень;
- управління процесом сканування шляхом обробки сигналів від інтерфейсу користувача.

resultsmanager - клас у якому реалізований функціонал взаємодії інтерфейсу користувача для перегляду, оновлення, редагування та видалення результатів проведених сканувань.

main.qml - код інтерфейсу користувача, що запускається при старті, в якому виконується перехід між вікнами програми і обробка подій повернення.

ScanPage.qml - код інтерфейсу користувача, який представляє головне вікно програми в якому виконується:

- вибір файлу з діапазонами IP-адрес;
- задання параметрів процесу сканування: кількість потоків для сканування порту, кількість потоків для перевірки наявності та відкритості протоколу RTSP за заданою кількістю шляхів до ресурсу, час очікування

відповіді на запит при скануванні відкритих портів, вибір кількості шляхів до ресурсів RTSP-трансляції;

- відображення поточних результатів сканування, таких як: кількість IP-адрес з відкритим портом, кількість IP-адрес, що залишилися, кількість пройдених IP-адрес за хвилину, кількість пройдених IP-адрес, кількість IP-адрес у черзі на перевірку відкритості порту, кількість IP-адрес у черзі на перевірку наявності та відкритості RTSP-трансляції за заданою кількістю шляхів до ресурсу, час, що залишився до закінчення процесу сканування, кількість серверів, що повернули код відповіді 200, кількість трансляцій до яких вдалося підключитися і отримати зображення;

- відображення статистики часу очікування відповіді на запит перевірки відкритості порту: мінімальний, середній, максимальний;

- запуск, зупинка, призупинення процесу сканування та застосування внесених змін до кількості потоків та часу очікування відповіді.

ResultsPage.qml - код інтерфейсу користувача, який представляє список результатів сканування, що відображає загальну кількість IP-адрес, кількість трансляцій з наявністю зображення, кількість трансляцій з наявністю аудіо і кількість трансляцій позначених як обране. Також знизу під основною кількістю відображається кількість активних трансляцій.

ResultInfoPage.qml - код інтерфейсу користувача представляє інформацію про обраний результат сканування, який додатково містить список знімків трансляцій з можливістю скопіювати RSTP-посилання трансляції оновити знімок трансляції, додати трансляцію в обрані, видалити трансляцію, перейти до історії знімків, підключитися до трансляції. Функція видалення історії знімків трансляцій, залишивши лише один знімок, для не помічених як обрані. Оновлення всіх трансляцій із вибором кількості потоків та параметрами оновлення: чи оновлювати зображення, оновити лише доступні, оновити лише із зображенням, оновити лише обрані.

AdditionalImagesPage.qml - код інтерфейсу користувача, який представляє список історії знімків з можливістю їх видалення.

PlayerPage.qml - код інтерфейсу користувача який представляє плеєр для відтворення RSTP-трансляції та функціонал збільшення різкості зображення.

Модулі інтерфейсу:

- CircularProgressBar.qml – для відображення прогресу пошуку у відсотках;
- CustomButton.qml – налаштована під дизайн інтерфейсу кнопка;
- CustomCheckBox.qml – чекбокс налаштований під дизайн інтерфейсу;
- CustomSpinBox.qml – спінбокс налаштований під дизайн інтерфейсу;
- CHader.qml – шапка головного вікна ScanPage.qml;
- CustomDialog.qml – стилізоване діалогове вікно;
- MainControl.qml – блок головного вікна ScanPage з елементами вибору кількості потоків сканування порту, кількості потоків перевірки наявності та відкритості трансляції, вибору кількості шляхів до ресурсу трансляції, вибору часу очікування відповіді на запит, кнопка запуску та зупинення процесу сканування;
- ProgressStats.qml - блок з інформацією про хід сканування головного вікна ScanPage;
- ResponseTimeStats.qml - блок зі статистичною інформацією часу відгуку на підключення до IP-адреси під час сканування портів головного вікна ScanPage;
- ResultInfo.qml – блок вікна ResultInfo з інформацією про обраний результат;
- ResultUpdateControl.qml – блок вікна ResultInfo з елементами вибору кількості потоків для оновлення результатів та параметрами оновлення: чи оновлювати зображення, оновити лише доступні, оновити лише із зображенням, оновити лише обрані.

3.2 Алгоритм пошуку та відображення відеотрансляцій

Алгоритм пошуку та відображення відео трансляцій наведено на рисунку 3.2.

Пошук відеотрансляцій здійснюється шляхом сканування ір-адрес четвертої версії за заданим діапазоном у форматі від-до. Пошук виконується за протоколом RTSP.

Кожна ір-адреса перевіряється на наявність відкритого порту 554, якщо порт відкритий, на ір-адресу виконується запит методом DESCRIBE, якщо у відповідь на запит повертається код “200 OK”, то виконується спроба отримання знімка відеотрансляції. При вдалому отриманні знімка, він буде відображено в інтерфейсі користувача.

Усі ір-адреси, що повернули код “200 OK”, зберігаються навіть при невдалому отриманні знімка, оскільки причиною цього можуть бути нестабільне підключення або перевантаженість сервера. У такому разі зображення може бути отримане під час оновлення результатів.

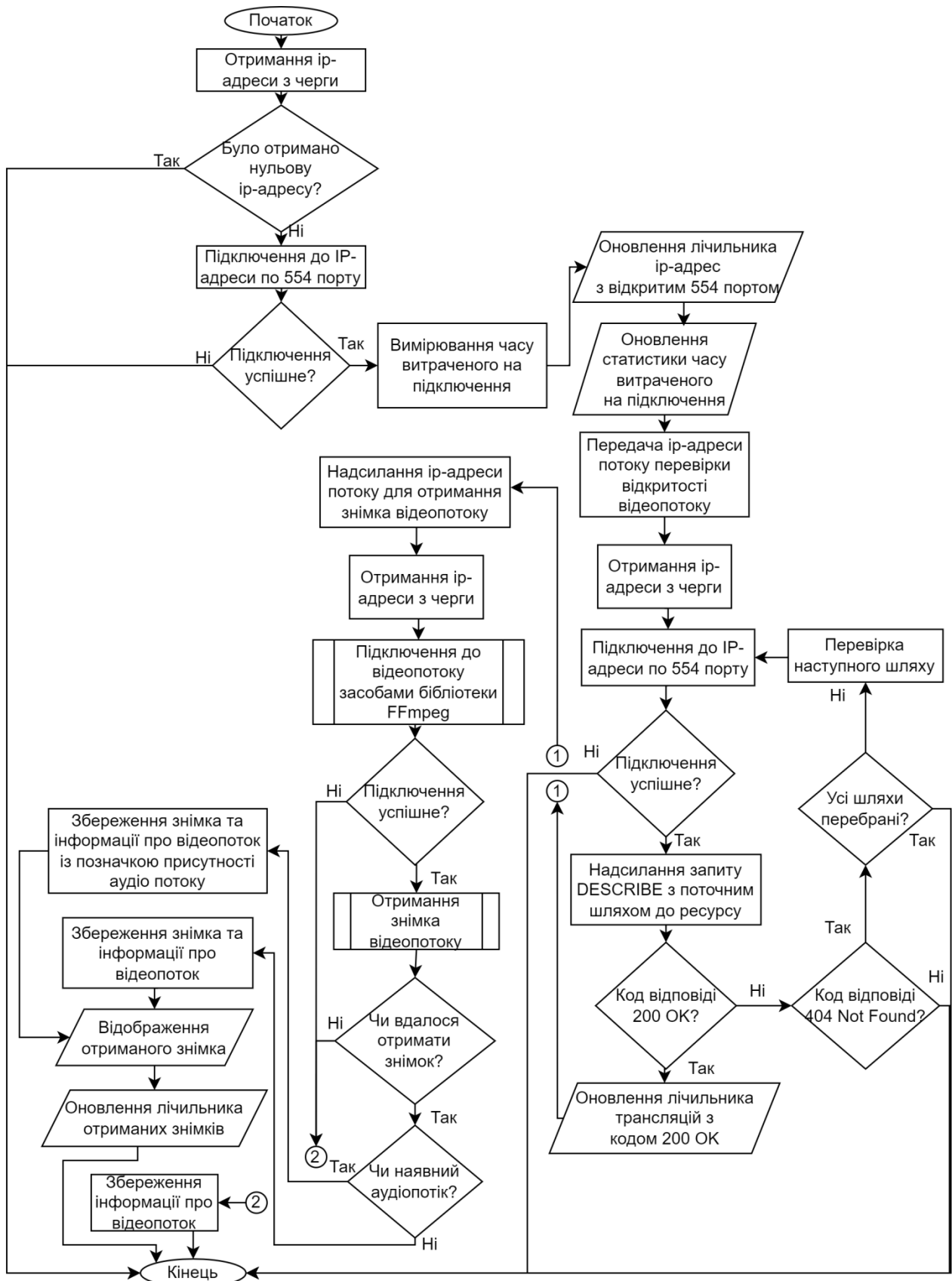


Рисунок 3.2 – Алгоритм пошуку та відображення відеотрансляцій

3.3 Алгоритм згортки різкості кадрів відео трансляції

Для збільшення різкості кадрів відеотрансляції до бібліотеки QmlAV було внесено такі доповнення:

- в клас QmlAVPlayer було додано поле логічного типу `m_sharpness` для встановлення стану, увімкнено або вимкнено, з інтерфейсу користувача, встановлення значення здійснюється за допомогою методу слота `setSharpness`, обробка зміни значення за допомогою методу сигналу `sharpnessChanged`;
- до класу QmlAVDemuxer було додано метод `setSharpness` для передачі значення атрибуту `m_sharpness` з класу QmlAVPlayer, класу QmlAVVideoDecoder;
- до класу QmlAVVideoDecoder був доданий метод `setSharpness` для встановлення значення атрибуту `m_sharpness`, яке у методі `frame` перед декодуванням кадру передається екземпляру об'єкта QmlAVVideoFrame, який представляє кадр зображення;
- в клас QmlAVFrame був доданий метод `convolutionMatrix`, який застосовується до кадру для збільшення його різкості, як параметри метод приймає індекс початку та кінця діапазону ширини зображення, що дозволяє викликати метод у декількох потоках розділяючи між ними пікселі зображення за шириною, що прискорює застосування матриці згортки.

Алгоритм методу `convolutionMatrix` класу QmlAVFrame наведено на рисунку 3.3.

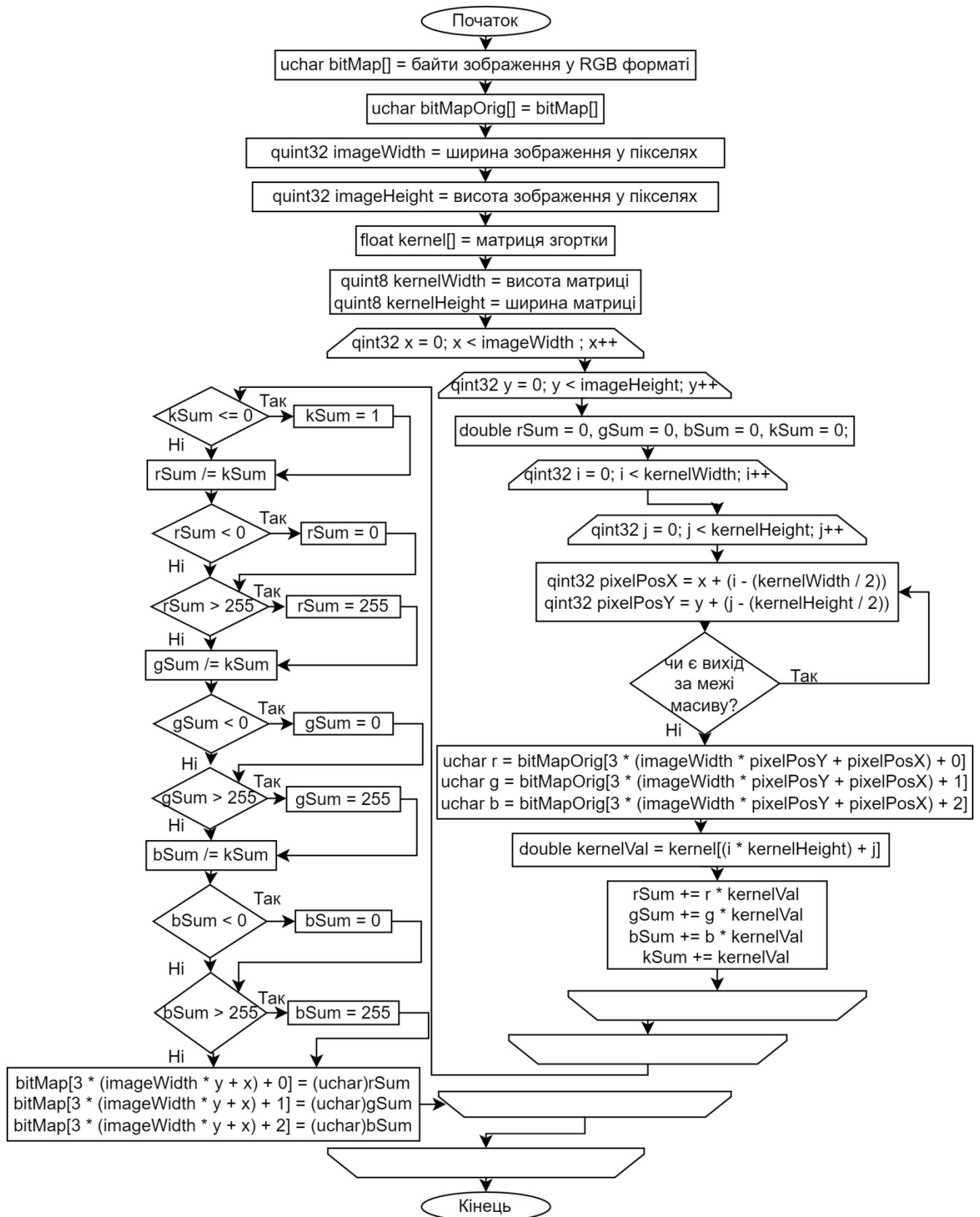


Рисунок 3.3 – Алгоритм методу “convolutionMatrix” класу “QmlAVFrame”

3.4 Розгляд класу DataManager

Клас для взаємодії з даними: збереження результатів, отримання результатів, видалення результатів, отримання списку шляхів до RTSP-трансляцій, створення необхідних файлів та директорій.

Приватні методи класу datamanager:

- safeResult - зберігає буфер з результатами у файл;
- safeChangesResult - зберігає буфер змінених результатів у файл;
- safeScanInfo - зберігає інформацію про сканування з буфера у файл;
- writeIpResult - записує структуру IpResult, що передається за посиланням, у файл за допомогою об'єкта QDataStream, що передається за посиланням;
- writeScanInfo- записує структуру ScanInfo, що передається за посиланням, у файл за допомогою об'єкта QDataStream, що передається за посиланням;
- readIpResult - зчитує з файлу структуру IpResult, що передається за посиланням за допомогою об'єкта QDataStream, що передається за посиланням;
- readScanInfo - зчитує з файлу структуру ScanInfo, що передається за посиланням, за допомогою об'єкта QDataStream, що передається за посиланням;
- setUp - створює робочу директорію програми та файли необхідні для її функціонування;
- removeImages - видаляє задану кількість зображень з ім'ям, що передається в параметр fileName в кількості, що передається в параметр counterImages.

Публічні методи класу datamanager:

- safeAll - зберігає буфер результату та інформацію про поточне сканування у файл;
- setUpScanningData - завантажує інформацію за заданою назвою сканування, якщо такої назви немає, вона буде створена;

- `addResult` - додає структуру `IpResult`, що передається за посиланням, в буфер результатів;
- `addChangeRes` - додає структуру `IpResult`, що передається за посиланням, в буфер результатів для збереження змін;
- `fillUpdateQueue` - заповнює чергу `safeQueue`, що передається за посиланням, елементами для оновлення які відповідають зазначеним параметрам фільтра;
- `fillIpQueue` - заповнює чергу `safeQueue`, що передається за посиланням, IP-адресами для сканування порту;
- `setFavorite` - додає та видаляє вказану трансляцію з обраних;
- `setResAlive` - встановлює значення доступності зазначеної трансляції;
- `removeAdditionalImages` - видаляє додаткові зображення поточного результату сканування;
- `removeItemResult` - видаляє вказану трансляцію;
- `removeScanData` - видаляє результати зазначеного сканування;
- `removeSingleImage` - видаляє вказаний знімок трансляції;
- `getScanInfo` - повертає структуру даних `ScanInfo`, що представляє інформацію про сканування за вказаною назвою;
- `getAllScans` - повертає вектор з інформацією про всі сканування;
- `getAllResultWithImage` - заповнює переданий вектор інформацією про трансляції із зображенням за вказаною назвою сканування;
- `getResultWithImageNotAlive` - заповнює переданий вектор інформацією про трансляції із зображенням, які не активні, за вказаною назвою сканування;
- `getResultWithImageAliveNotFavorite` - заповнює переданий вектор інформацією про трансляції із зображенням, які не є обраними, за вказаною назвою сканування;
- `safeContinue` - зберігає не проскановані IP-адреси з черги на сканування у файл;

- `getFilenamesIpRanges` - повертає вектор із назвами файлів у директорії з діапазонами IP-адрес;
- `getIpAmount` - повертає кількість IP-адрес у зазначеному файлі діапазонів IP-адрес;
- `getImagesPath` - повертає повний шлях до директорії із зображеннями;
- `loadScanInfo` - завантажує інформацію про сканування;
- `loadPaths` - заповнює переданий вектор зазначеною кількістю шляхів із файлу "Paths.txt";

3.5 Розгляд класу **FFmpeg**

Клас для захоплення та збереження зображення з RTSP потоку.

Приватні методи класу `FFmpeg`:

- `decode_packet` - декодує та зберігає вказаний пакет зображення по вказаному шляху із зазначеною назвою файлу;
- `clear` – очищає виділену пам'ять.

Клас має один публічний метод `capture`, що приймає RTSP-посилання, шлях для збереження та назву зображення, як результат повертає 1 при вдалому захопленні без наявності аудіо, 2 при вдалому захопленні та наявності аудіо і 0 при відсутності зображення.

3.6 Розгляд класу **NetworkScanner**

Клас у якому реалізований функціонал взаємодії інтерфейсу користувача сканування мережі, він реалізує наступний функціонал:

- пошук ір-адрес з відкритим 554 портом;
- перевірка наявності та доступності RTSP трансляції шляхом запиту `DESCRIBE`;

- збереження отриманих результатів, використовуючи раніше описаний клас `datamanager`;
- оновлення інформації про процес сканування в інтерфейсі користувача шляхом сигналізації про зміну значень;
- управління процесом сканування шляхом обробки сигналів від інтерфейсу користувача.

Приватні методи класу `NetworkScanner`:

- `connect_ex` - робить спробу підключення до IP-адреси по 554 порту, при успішному підключенні повертає `true`;
- `turnOff` - зупиняє всі запущені процеси, чекаючи їх зупинки, після чого виконується збереження даних;
- `resultCatcherIpSenderThreadFunc` - функція, яка виконується в окремому потоці, яка виконує заповнення черги IP-адресами для сканування, вилучення результатів з черги та їх обробку, збереження та оновлення інформації в інтерфейсі користувача;
- `portScanThreadFunc` - функція, яка виконується в окремому потоці, яка виконує вилучення IP-адреси з черги на перевірку відкритості порту, якщо порт відкритий то IP-адреса передається в чергу для перевірки наявності RTSP-трансляції та її відкритості;
- `pathCheckThreadFunc` - функція, яка виконується в окремому потоці, яка виконує перевірку наявності RTSP-трансляції та її відкритості перебираючи зазначений список шляхів до трансляції, при знаходженні відкритої трансляції інформація про неї передається в чергу для отримання зображення;
- `captureThreadFunc` - функція, яка виконується в окремому потоці, яка виконує знімок зображення з трансляції, інформація про результат отримання знімка передається у чергу з результатами;
- `applyThreadsQuantityThreadFunc` - функція, яка виконується в окремому потоці для застосування зміни кількості потоків;
- `calculateTimeStat` - розраховує статистику часу, що залишився до закінчення сканування і оновлює значення в інтерфейсі користувача;

- `calculateAverageResponseTime` - розраховує середній час відповіді сервера та оновлює інформацію в інтерфейсі користувача;
- `safeFromQueues` - зберігає дані з черг у файл;
- `checkPath` - здійснює підключення за вказаним шляхом з поверненням коду відповіді, за відсутності відповіді повертається нульове значення;
- `numberFormatting` - форматування числа, розділяючи його пробілами через кожні 3 символи.

Публічні методи класу `NetworkScanner`:

- `setScanningRanges` - обробка сигналу про завершення вибору діапазону IP-адрес від інтерфейсу;
- `start` – обробка сигналу старту сканування від інтерфейсу;
- `stop` – обробка сигналу зупинки сканування від інтерфейсу;
- `stopFilling` – обробка сигналу зупинки заповнення черги для сканування від інтерфейсу;
- `pause` – обробка сигналу паузи процесу сканування від інтерфейсу;
- `contin` – обробка сигналу продовження процесу сканування від інтерфейсу;
- `loadIpRangesModel` – обробка сигналу для завантаження списку діапазонів від інтерфейсу;
- `apply` – обробка сигналу застосування змін параметрів процесу сканування від інтерфейсу.

3.7 Розгляд класу `ResultsManager`

Клас у якому реалізований функціонал взаємодії інтерфейсу користувача для перегляду, оновлення, редагування та видалення результатів проведених сканувань.

Приватні методи класу `ResultsManager`:

- resultCatcherIpSenderThreadFunc - функція виконувана в окремому потоці, яка виконує заповнення черги IP-адресами для оновлення трансляцій, отримання результатів з черги, їх обробку та збереження;
- updateThreadFunc - функція, що виконується в окремому потоці яка виконує підключення до трансляції, інформація про результат передається в чергу з результатами;
- updateSinglResultThreadFunc - функція, що виконується в окремому потоці, яка виконує оновлення знімка за зазначеним RSTP-посиланням;
- updateInfo - оновлює інформацію у інтерфейсі користувача;
- connect_ex - робить спробу підключення до IP-адреси по 554 порту, при успішному підключенні повертає true;
- checkPath - здійснює підключення за вказаним шляхом з поверненням коду відповіді, за відсутності відповіді повертається 0;
- numberFormatting - форматування числа, розділяючи його пробілами через кожні 3 символи.

Публічні методи класу ResultsManager:

- start – обробка сигналу старту оновлення від інтерфейсу;
- stop – обробка сигналу зупинки оновлення від інтерфейсу;
- setFavorite - обробка сигналу додавання та видалення трансляції з обраних від інтерфейсу;
- updateImage – обробка сигналу запуску потоку оновлення знімка певної трансляції від інтерфейсу;
- loadResultsInfo - обробка сигналу завантаження списку інформації сканувань від інтерфейсу;
- loadResultsImages - обробка сигналу завантаження списку трансляцій від інтерфейсу;
- loadAdditionalImages – обробка сигналу завантаження історії знімків трансляції від інтерфейсу;
- setCurrentResultInfo - обробка сигналу встановлення вибраної інформації про сканування від інтерфейсу;

- removeScanData - обробка сигналу видалення інформації про сканування від інтерфейсу;
- removeAdditionalImages - обробка сигналу видалення додаткових знімків трансляцій від інтерфейсу;
- removeItemResult - обробка сигналу видалення вказаної трансляції від інтерфейсу;
- removeSingleAdditionalImage - обробка сигналу видалення вказаного знімка від інтерфейсу.

3.8 Зберігання даних

Збереження даних здійснюється за допомогою серіалізації та десеріалізації структур даних.

Інформація про трансляцію зберігається як структура IpResult (рис. 3.4) і містить таку інформацію:

- ip - IP-адреса у форматі позитивного цілого числа;
- url - URL-посилання трансляції;
- counterImages - кількість знімків трансляції;
- counterNotAlive - кількість послідовних невдалих спроб підключення;
- audio - чи є аудіо;
- inFavorites - чи знаходиться в обраних.

Інформація про сканування зберігається як структура ScanInfo (рис. 3.5) і містить таку інформацію:

- name - назва сканування;
- totalRes - кількість знайдених трансляцій;
- aliveRes - кількість працюючих трансляцій;
- captured - кількість трансляцій, у яких був взятий знімок;
- aliveCaptured - кількість працюючих трансляцій у яких був взятий знімок;

- audio - кількість трансляцій зі звуком;
- aliveAudio - кількість працюючих трансляцій зі звуком;
- favorites - кількість обраних трансляцій;
- aliveFavorites - кількість працюючих обраних трансляцій.

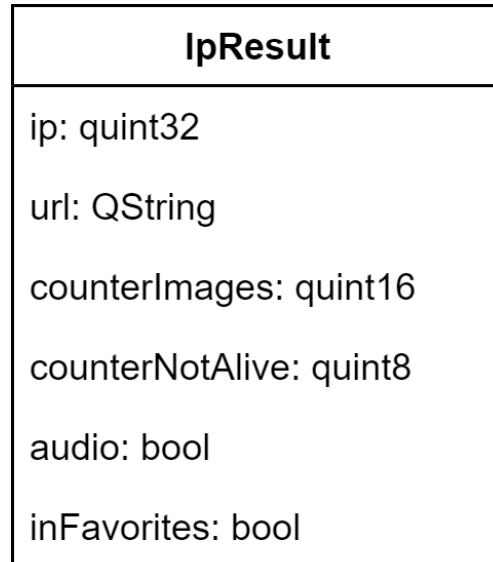


Рисунок 3.4 – Діаграма структури IpResult з назвою полів та їх типом

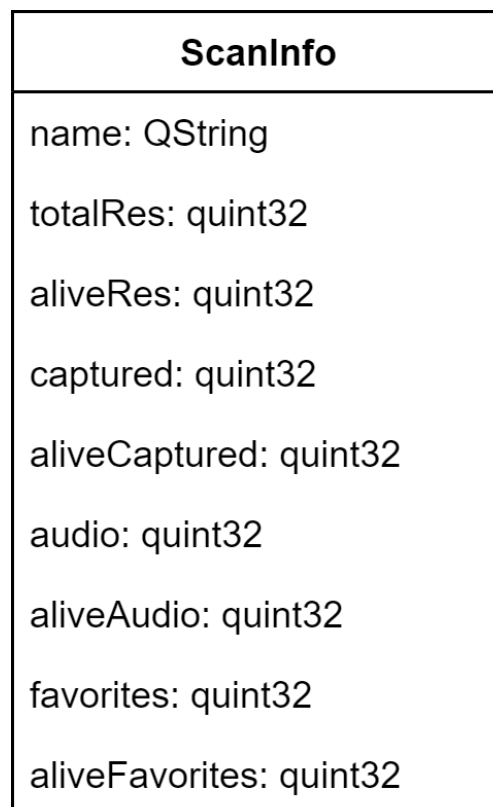


Рисунок 3.5 – Діаграма структури ScanInfo з назвою полів та їх типом

На рисунку 3.6 зображено файлову структуру робочої директорії програми.

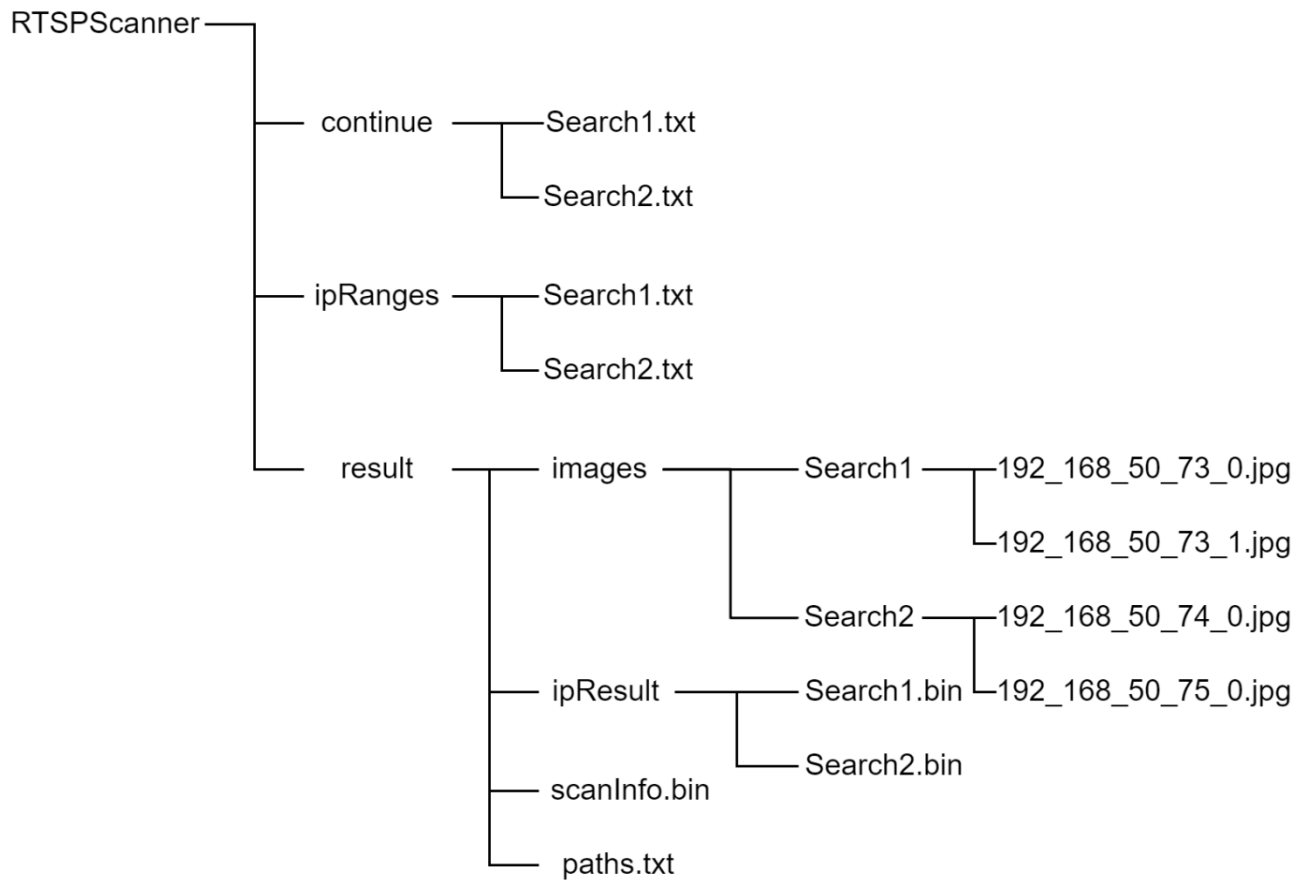


Рисунок 3.6 – Файлова структура робочої директорії програми

Файли з трансляціями зберігаються в папці ipResult з назвою, яка відповідає назві сканування з розширенням ".bin"

Файли з інформацією про сканування зберігаються в папці result з назвою scanInfo і розширенням ".bin"

Знімки трансляцій зберігаються в папці, яка відповідає назві сканування в папці images, назва зображень складається з 5 цифр розділених знаком нижнього підкреслення, перші 4 цифри відповідають цифрам IP-адреси, п'ята цифра відповідає номеру знімка даної трансляції.

Шляхи трансляцій зберігаються в папці result у файлі "paths.txt".

Діапазон IP-адрес має знаходитися в папці ipRanges у форматі txt, назва

файлу буде відповідати назві сканування.

При зупинці процесу сканування необроблені IP-адреси з черги зберігаються в папку continue у форматі txt, назва файлу буде відповідати назві сканування.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення й умови застосування програми

ПЗ «RTSPScanner» призначене для пошуку та відображення відкритих RTSP-трансляцій з можливістю збільшення чіткості зображення трансляції.

Технічні вимоги:

- операційна система – Android 11 і вище;
- архітектура процесора ARMv8;
- вільне місце на жорсткому диску не менше ніж 512 мб;
- оперативна пам'ять щонайменше 3 гб;
- наявність wi-fi або ethernet.

Програмні вимоги:

- доступ до мережного підключення;
- доступ до керування файлами на пристрої.

4.2 Характеристика програми

Робоче ім'я програми "RTSPScanner".

Початковий обсяг файлів програми має розмір приблизно 25 мб.

Програма розроблена за допомогою мови програмування C++ у середовищі розробки Qt Creator, інтерфейс реалізований за допомогою мови QML.

Основним файлом, що встановлює програму, є «RTSPScanner.apk».

Директорії програми:

- "RTSPScanner \results\images" - необхідна для збереження знімків трансляції;
- "RTSPScanner \results\ipResult" - необхідна для збереження інформації про трансляції у файлі з розширенням "bin", назва файлу відповідає назві сканування;

- “RTSPScanner \continue” - необхідна для збереження непройдених ір-адрес;
- “RTSPScanner \ipRanges” - необхідна для збереження діапазонів ір-адрес.

Файли даних, що знаходяться в папці "RTSPScanner \results":

- paths.txt – містить шляхи до ресурсів RTSP-трансляцій;
- scanInfo.bin – містить інформацію про результати сканування.

4.3 Звернення до програми

Для звернення до програми необхідно встановити Qt Creator версії «1.15», після чого необхідно перейти в папку з проектом “RTSPScanner ” (рис. 4.1) та запустити файл “ RTSPScanner.pro” який запустить проект у середовищі розробки, після чого можна редагувати програму, щоб запустити програму необхідно натиснути кнопку “Пуск” та мати запущений емулятор або підключений пристрій з увімкненим режимом розробника та налагодження.

	ProgressStats.qml	30.05.2023 22:52	Qt Quick Markup I...	6 КБ
	qml.qrc	20.03.2023 04:29	Файл "QRC"	2 КБ
	ResponseTimeStats.qml	30.05.2023 22:52	Qt Quick Markup I...	4 КБ
	ResultInfo.qml	30.05.2023 22:52	Qt Quick Markup I...	7 КБ
	ResultInfoPage.qml	23.11.2023 04:36	Qt Quick Markup I...	22 КБ
	resultsmanager.cpp	30.05.2023 22:52	C++ Source file	18 КБ
	resultsmanager.h	30.05.2023 22:52	C++ Header file	5 КБ
	ResultsPage.qml	30.05.2023 22:52	Qt Quick Markup I...	13 КБ
	ResultUpdateControl.qml	30.05.2023 22:52	Qt Quick Markup I...	6 КБ
	RTSPScanner.pro	20.03.2023 04:33	Qt Project file	4 КБ
	RTSPScanner.pro.user	24.11.2023 04:14	Файл "USER"	38 КБ

Рисунок 4.1 – Вміст папки проекту “ RTSPScanner ”

4.4 Початкові та вихідні дані

Початковими даними є:

- ір-адреси у вигляді діапазону;
- кількість потоків сканування;
- кількість потоків перевірки наявності RTSP-трансляцій та її відкритості;

- час очікування відповіді на запит від ір-адреси;
- кількість шляхів до RTSP-трансляцій.

Вихідними даними є:

- кількість ір-адрес з відкритим 554 портом;
- кількість відкритих RTSP-трансляцій;
- кількість відкритих RTSP-трансляцій, з яких вдалося отримати зображення;
- зображення з RTSP-трансляцій;
- відео RTSP-трансляцій;
- відео RTSP-трансляцій зі збільшеною чіткістю.

4.5 Повідомлення

Під час виконання програми в режимі налагодження, в консоль може виводиться налагоджувальна інформація та повідомлення про помилки. Відлагоджувальна інформація може являти собою повідомлення про запуск або завершення процесів та потоків, дана інформація необхідна для проведення налагоджувальних робіт при пошуку помилок або відстеження ходу виконання ключових операцій.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Розроблене програмне забезпечення «RTSPScanner» надає можливість виконувати пошук відкритих RTSP-трансляцій за вказаним діапазоном IP-адрес, знайдені трансляції користувач може переглядати, робити знімки, оновлювати, переглядати та очищати історію знімків, видаляти, додавати до обраного, видаляти результати сканування. Під час перегляду трансляції можна збільшити чіткість зображення.

Функції, які виконує програма:

- обробка діапазонів IP-адрес;
- вибір кількості потоків для виконання пошуку;
- пошук відкритих RTSP-трансляцій;
- можливість поставити процес на паузу;
- можливість завершити процес із збереженням непройдених ір-адрес;
- відстеження результатів виконання процесу;
- можливість виведення зображення з RTSP-трансляцій;
- перегляд результатів пошуку;
- видалення результатів пошуку;
- оновлювати знімок трансляції;
- оновлювати знімок трансляцій за заданими параметрами;
- отримувати URL-адресу трансляції;
- переглядати історію знімків;
- очищати історію знімків;
- можливість збільшити чіткість зображення під час перегляду трансляції;
- додавати трансляції до обраних.

5.2 Умови виконання програми

Технічні вимоги:

- операційна система – Android 11 і вище;
- архітектура процесора ARMv8;
- вільне місце на жорсткому диску не менше ніж 512 мб;
- оперативна пам'ять щонайменше 3 гб;
- наявність wi-fi або ethernet.

Програмні вимоги:

- доступ до мережного підключення;
- доступ до керування всіма файлами на пристрої.

5.3 Виконання програми

Спочатку потрібно встановити програму на пристрій під управлінням Android, та запустити його. Для виконання програми необхідно мати доступ до інтернету та повний доступ до пам'яті пристрою.

Для запуску процесу пошуку відкритих трансляцій необхідно створити файл із діапазонами IP-адрес у папці “RTSPScanner\ipRanges”, назва файлу буде відповідати назві сканування. Діапазони IP-адрес повинні мати такий вигляд: “1.0.0.0-1.0.0.255” і бути розділеними переходом на новий рядок (рис. 5.1).

Редагування списку шляхів до трансляцій здійснюється через редагування текстового файлу “paths.txt” будь-яким текстовим редактором (рис. 5.2).

```

1.0.0.0-1.0.0.255
1.0.4.0-1.0.7.255
1.1.1.0-1.1.1.255
1.2.3.0-1.2.3.255
1.4.0.0-1.4.0.255
1.16.64.0-1.18.115.255
1.40.0.0-1.40.0.255
1.40.1.0-1.40.5.255
1.40.6.0-1.40.6.255
1.40.7.0-1.40.11.255
1.40.12.0-1.40.12.255
1.40.13.0-1.40.13.255
1.40.14.0-1.40.14.255
1.40.15.0-1.40.28.255
1.40.29.0-1.40.29.255

```

Рисунок 5.1 – Приклад файлу з діапазоном IP-адрес

```

/11
/mpeg4/ch01/main/av_stream
/cam/realmonitor?channel=1&subtype=0
/Streaming/Channels/101
/axis-media/media.amp
/stream1
/live1.sdp

```

Рисунок 5.2 – Приклад файлу із шляхами до ресурсу трансляцій

Для запуску програми необхідно перейти до встановлених застосунків та торкнутися іконки з назвою RTSPScanner, після чого відкриється головне вікно програми (рис. 5.3).

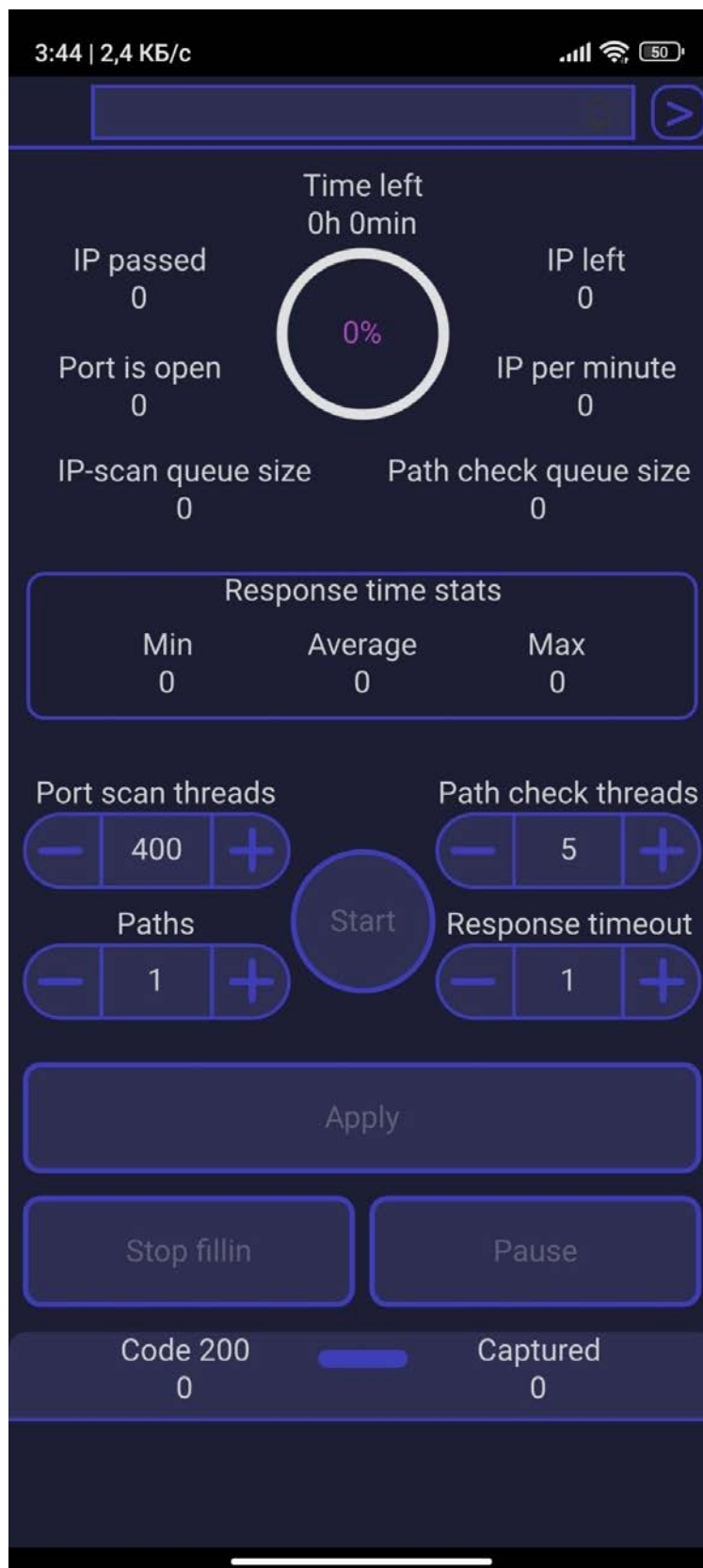


Рисунок 5.3 – Головне вікно програми

Для запуску процесу пошуку відкритих трансляцій, необхідно вибрати текстовий файл з IP-діапазонами зі списку на самому верху вікна (рис. 5.4).

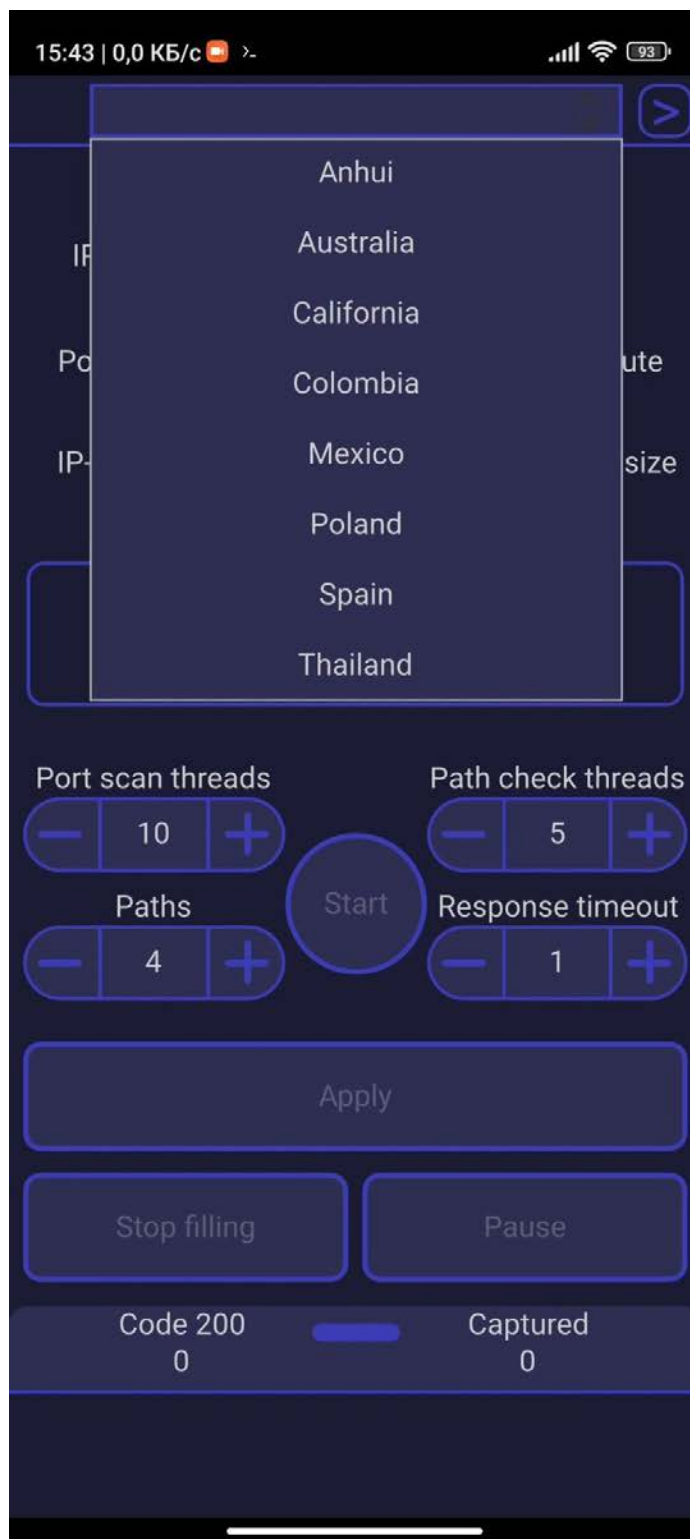


Рисунок 5.4 – Список, що випадає з назвами файлів IP-діапазонів

Також перед запуском можна вибрати кількість шляхів до ресурсу, кількість потоків для сканування та пошуку шляху, обрати час очікування відповіді на запит відкритості 554 порту у IP-адреси в секундах і натиснути кнопку “Start” (рис. 5.5).

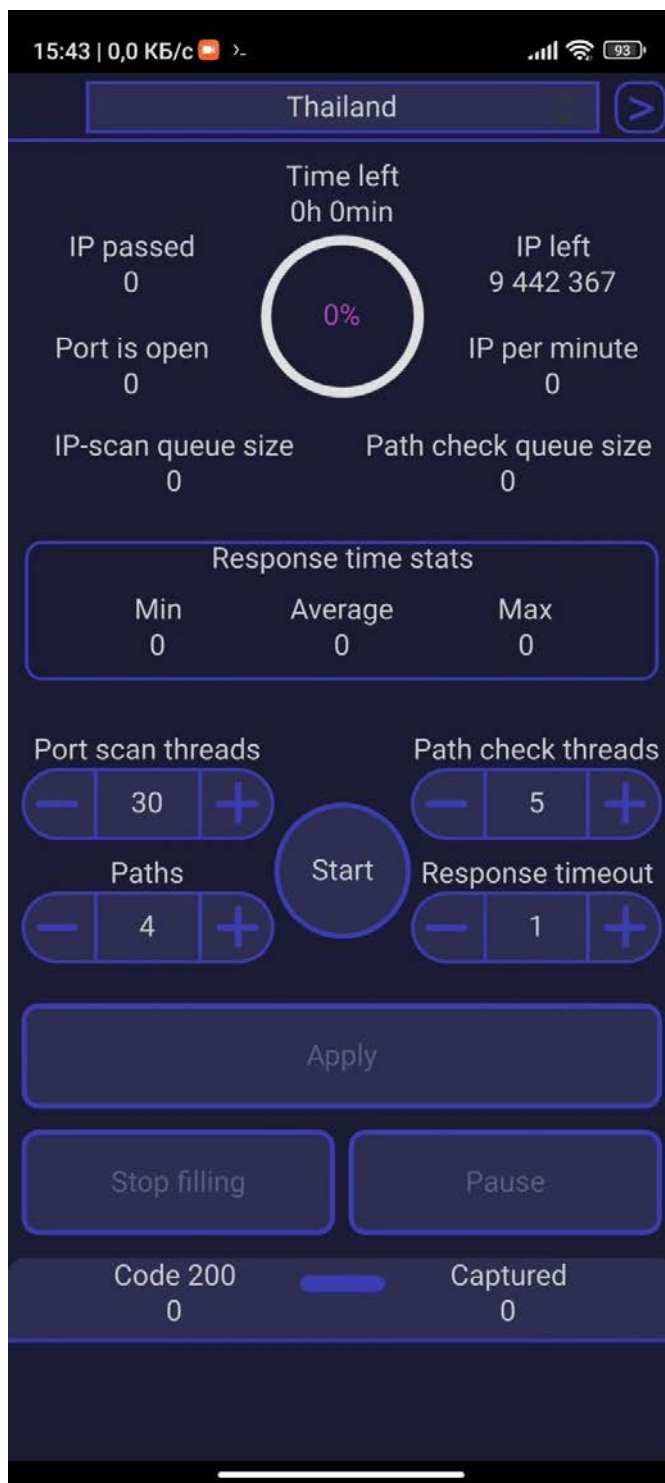


Рисунок 5.5 – Результат вибору файлу IP-діапазонів “Thailand” та зміни кількості потоків сканування порту

Після натискання кнопки “Start” буде виконано запуск усіх необхідних потоків. Від обраної кількості потоків залежатиме час виконання процесу сканування IP-адрес. При активному процесі сканування будуть активні кнопки “Stop” та “Pause” (рис. 5.6).

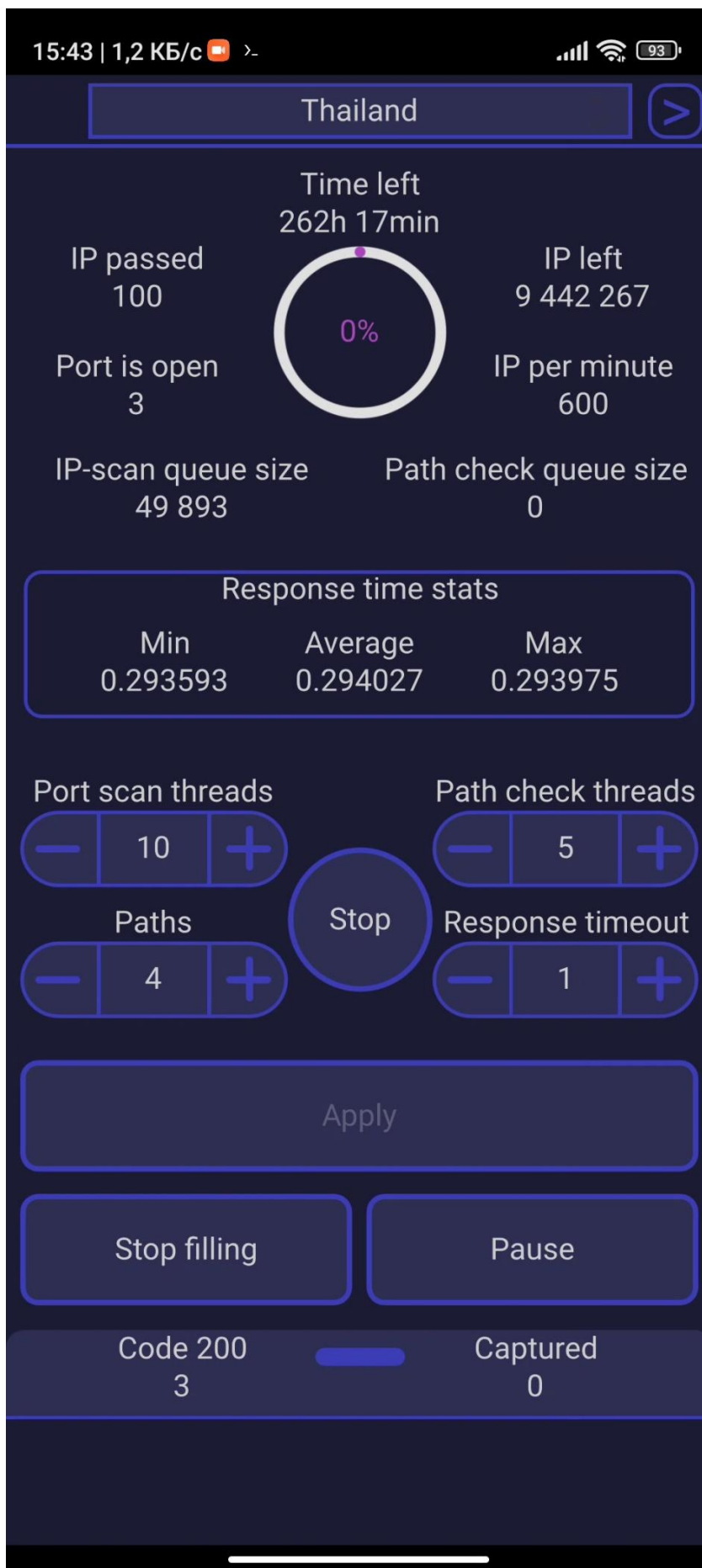


Рисунок 5.6 – Процес сканування активний

При натисканні на кнопку “Pause” процес пошуку буде поставлено на паузу, а при натисканні “Continue” процес буде продовжено (рис. 5.7).

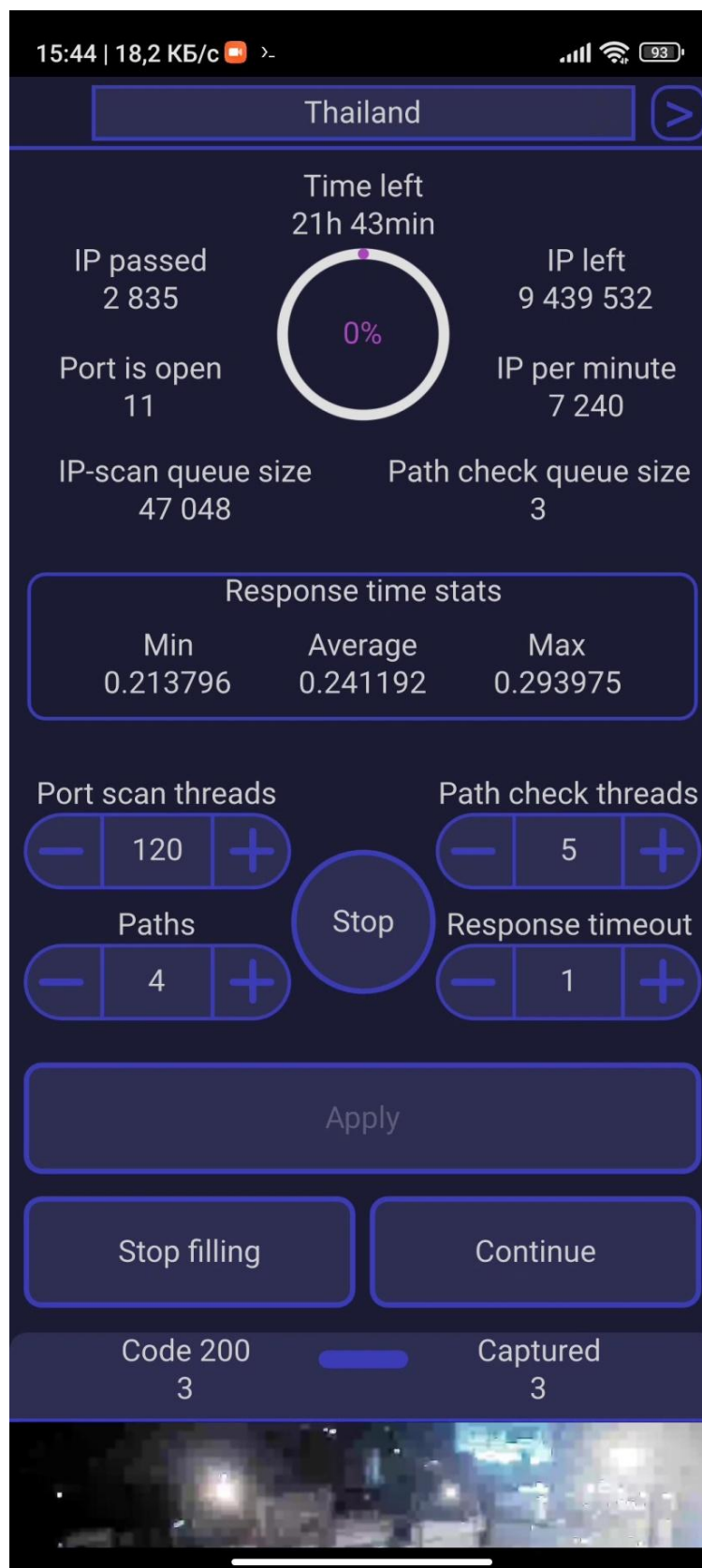


Рисунок 5.7 – Процес сканування поставлений на паузу

При натисканні на кнопку "Stop fillin" буде припинено заповнення черги для сканування порту, заповнення виконується на 50 тисяч IP-адрес якщо їх кількість в черзі менше 5 тисяч (рис. 5.8).

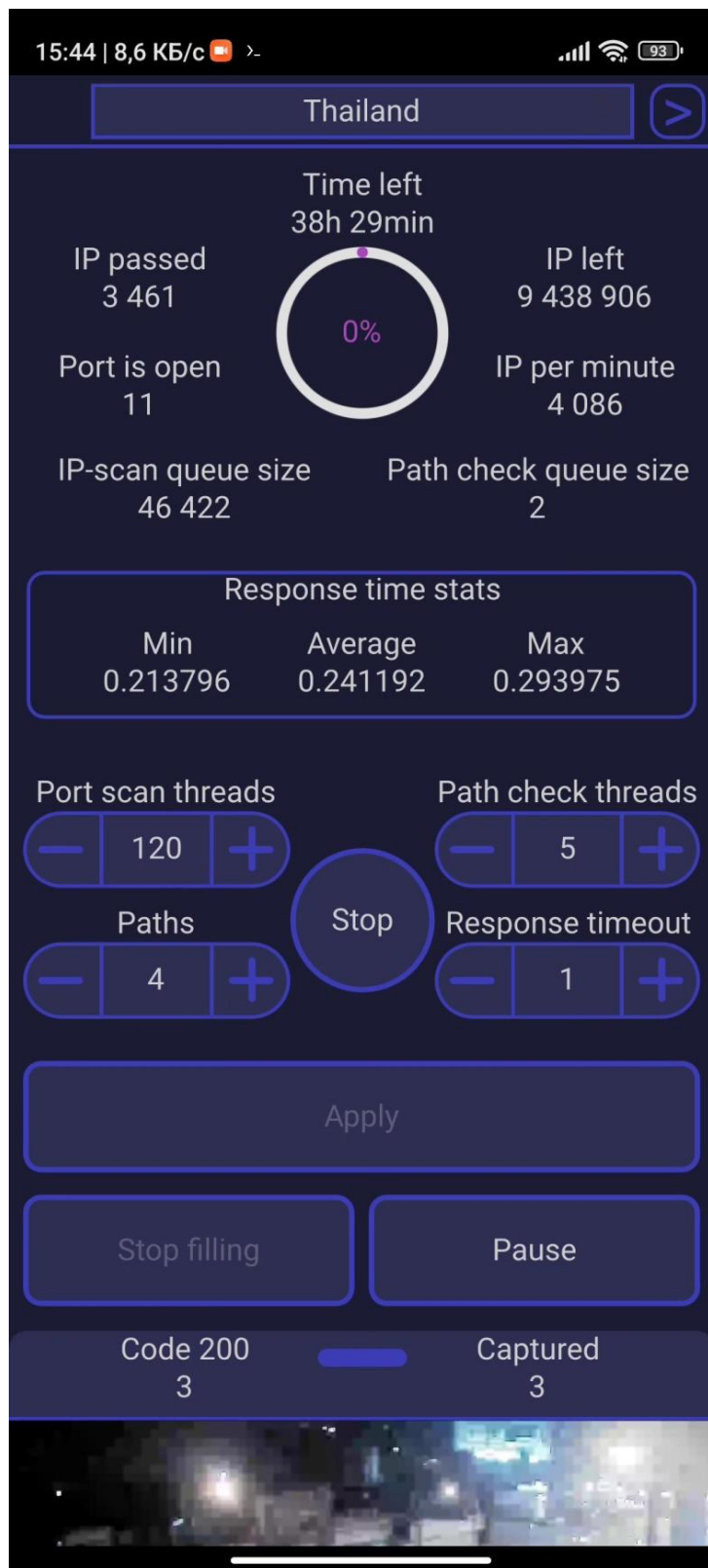


Рисунок 5.8 – Заповнення черги IP-адресами припинено

У разі зміни параметрів процесу пошуку кнопка “Apply” стає активною, при натисканні на яку зміни будуть застосовані (рис. 5.9).

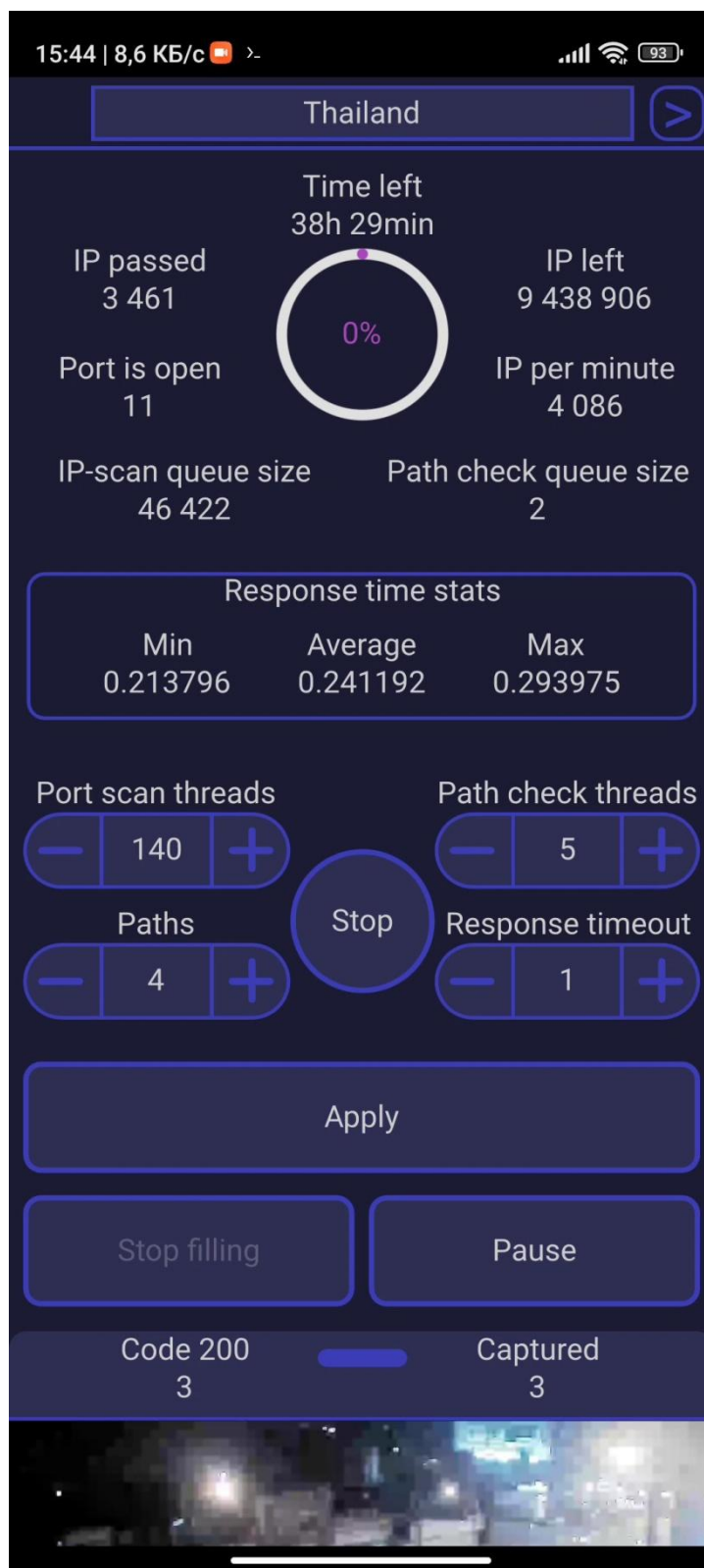


Рисунок 5.9 – Змінені параметри процесу пошуку та активна кнопка “Apply”

При натисканні на кнопку “Stop” або після закінчення IP-адрес у чергах, процес пошуку буде завершено (рис. 5.10).

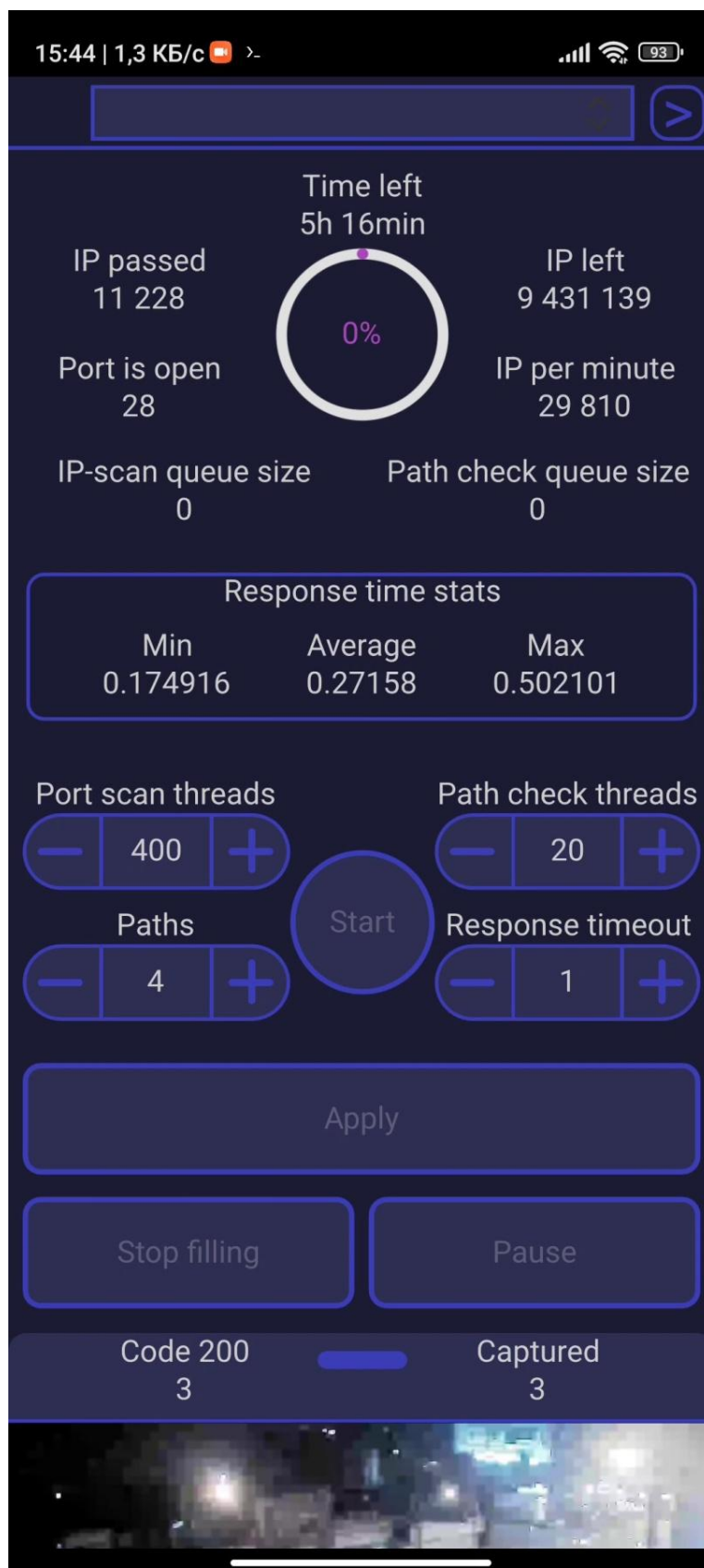


Рисунок 5.10 – Завершений процес сканування

Знизу знаходиться елемент "bottom sheet" (рис. 5.11), який складається з "bottom sheet content" і "bottom sheet peek" де відображається кількість відкритих трансляцій і кількість трансляцій у яких був отриманий знімок трансляції, при натисканні на "bottom sheet peek" елемент "bottom sheet" буде розгорнуто на весь екран при повторному натисканні елемент повернеться у вихідне становище. Елемент "bottom sheet content" містить список зі знімками трансляцій.

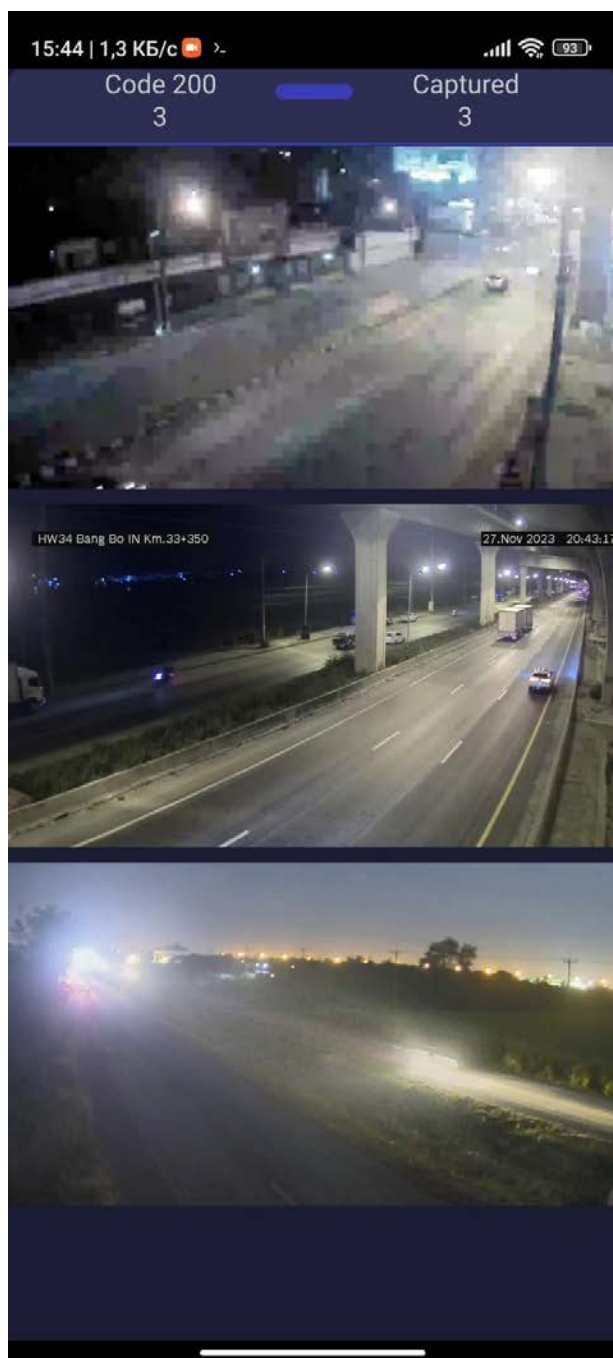
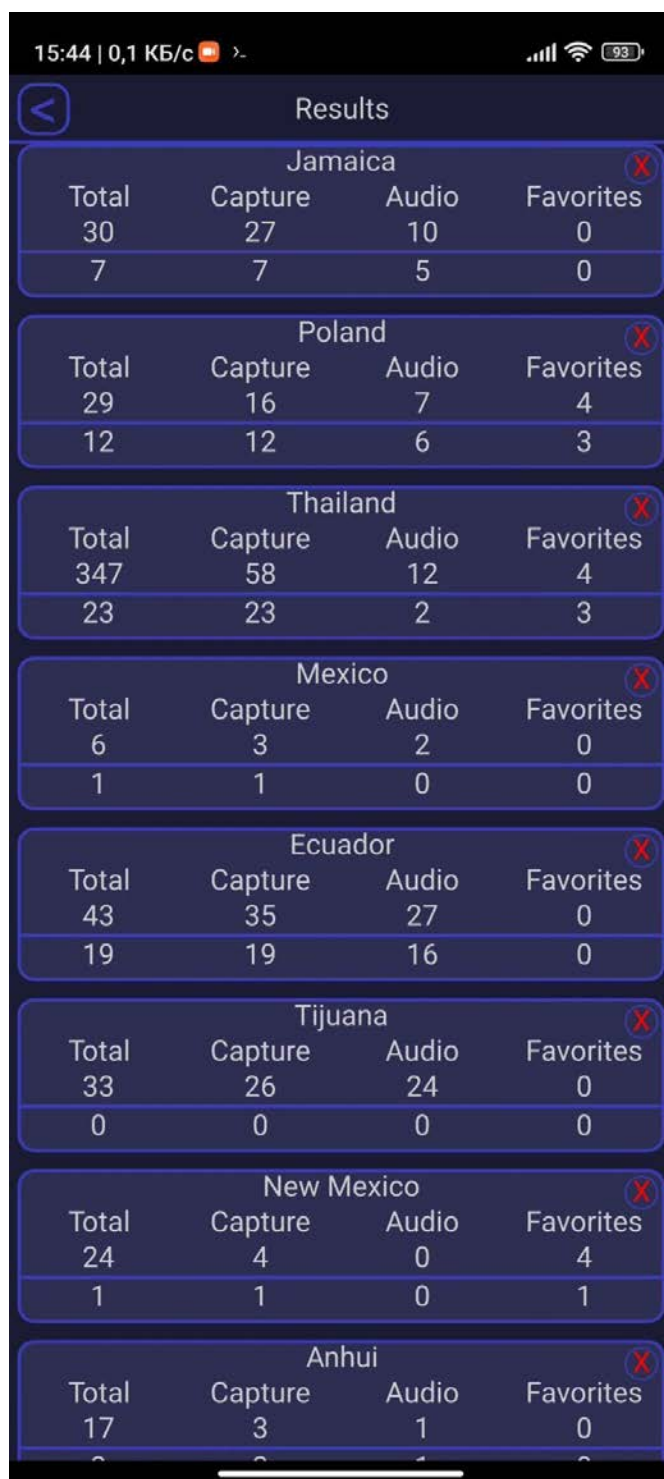


Рисунок 5.11 – Елемент “bottom sheet” розгорнутий на весь екран

При натисканні на кнопку у верхньому правому куті, відкриється вікно зі списком результатів сканування (рис. 5.12).

Кожний елемент списку відображає назву та кількість трансляцій в наступному порядку: всього, з зображенням, з аудіо, обрані.

Другий ряд чисел відображає кількість активних трансляцій.



Results			
Jamaica			
Total	Capture	Audio	Favorites
30	27	10	0
7	7	5	0
Poland			
Total	Capture	Audio	Favorites
29	16	7	4
12	12	6	3
Thailand			
Total	Capture	Audio	Favorites
347	58	12	4
23	23	2	3
Mexico			
Total	Capture	Audio	Favorites
6	3	2	0
1	1	0	0
Ecuador			
Total	Capture	Audio	Favorites
43	35	27	0
19	19	16	0
Tijuana			
Total	Capture	Audio	Favorites
33	26	24	0
0	0	0	0
New Mexico			
Total	Capture	Audio	Favorites
24	4	0	4
1	1	0	1
Anhui			
Total	Capture	Audio	Favorites
17	3	1	0

Рисунок 5.12 – Вікно зі списком результатів сканування

При натисканні на один з елементів відкриється вікно з інформацією про результат сканування (рис. 5.13).

Вікно результату надає функціонал оновлення інформації про трансляції, перед запуском оновлення є можливість вибрати кількість потоків та параметри фільтра: тільки активні, із зображенням, обрані, чи оновлювати знімок.

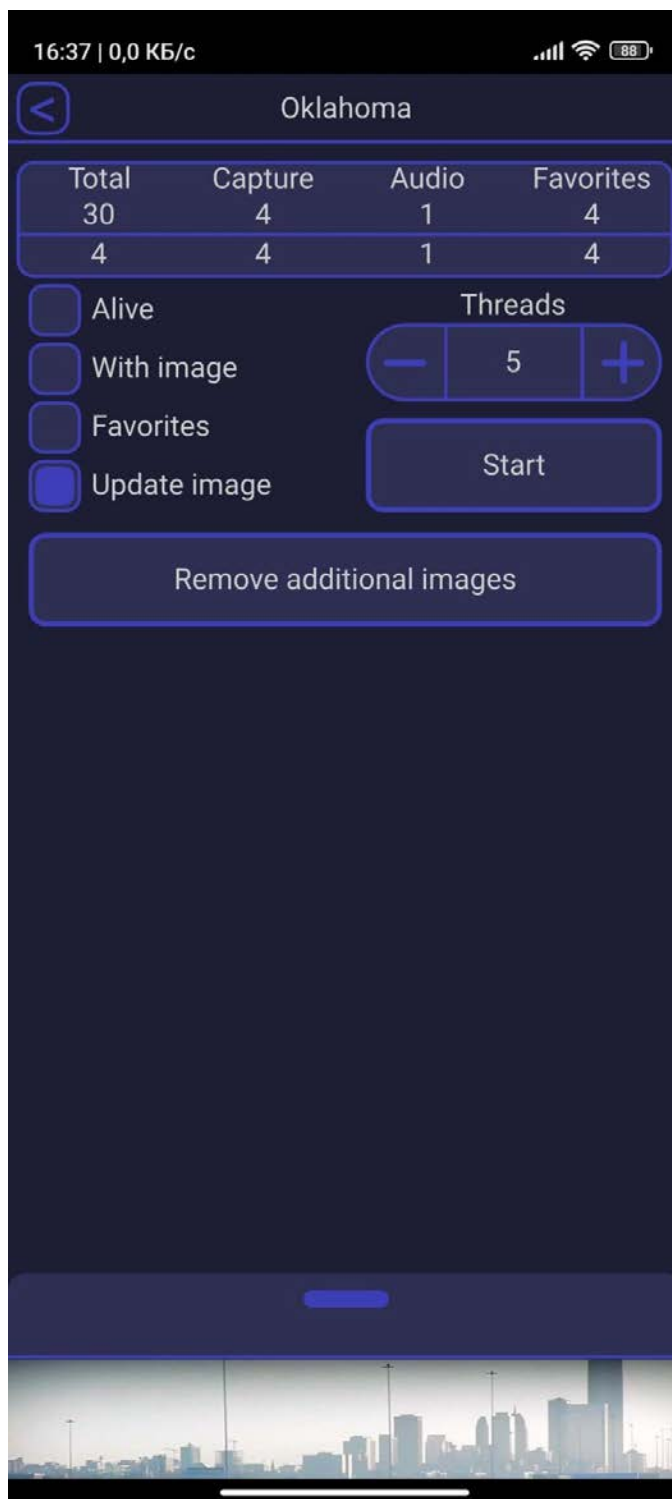


Рисунок 5.13 – Вікно з інформацією про результат сканування

Усі знімки трансляцій знаходяться в елементі "bottom sheet" (рис. 5.14).

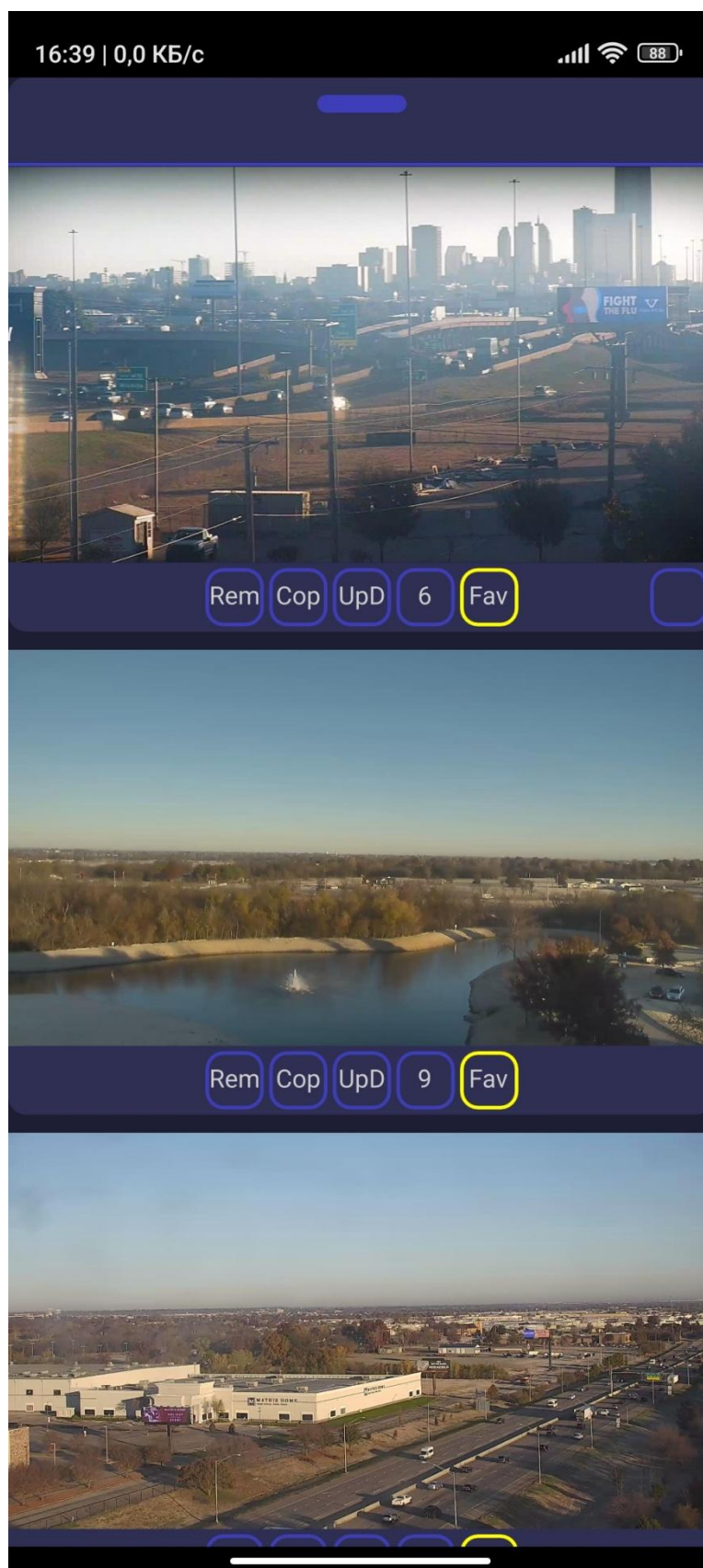


Рисунок 5.14 – Розгорнутий елемент "bottom sheet" вікна результату

Елемент списку трансляцій складається зі знімка трансляції та нижньої панелі, яка має два індикатори (рис. 5.15):

- червоний квадрат зліва відображає кількість послідовних невдалих спроб підключення до трансляції;
- синій квадрат праворуч присутній у трансляції з присутнім аудіо потоком.

Елементи посередині є кнопками, які надають наступний функціонал:

- Rem – видалення трансляції;
- Cop - копіювання URL-адреси трансляції у буфер обміну;
- UpD - оновлення знімка трансляції, під час оновлення рамка стає зеленого кольору;
- число знімків – відображення числа знімків, при натисканні виконується перехід до вікна перегляду історії знімків з можливістю їх видалення (рис. 5.16);
- Fav - додавання та видалення трансляції з обраних, при додаванні трансляції в обрані рамка стає жовтою.

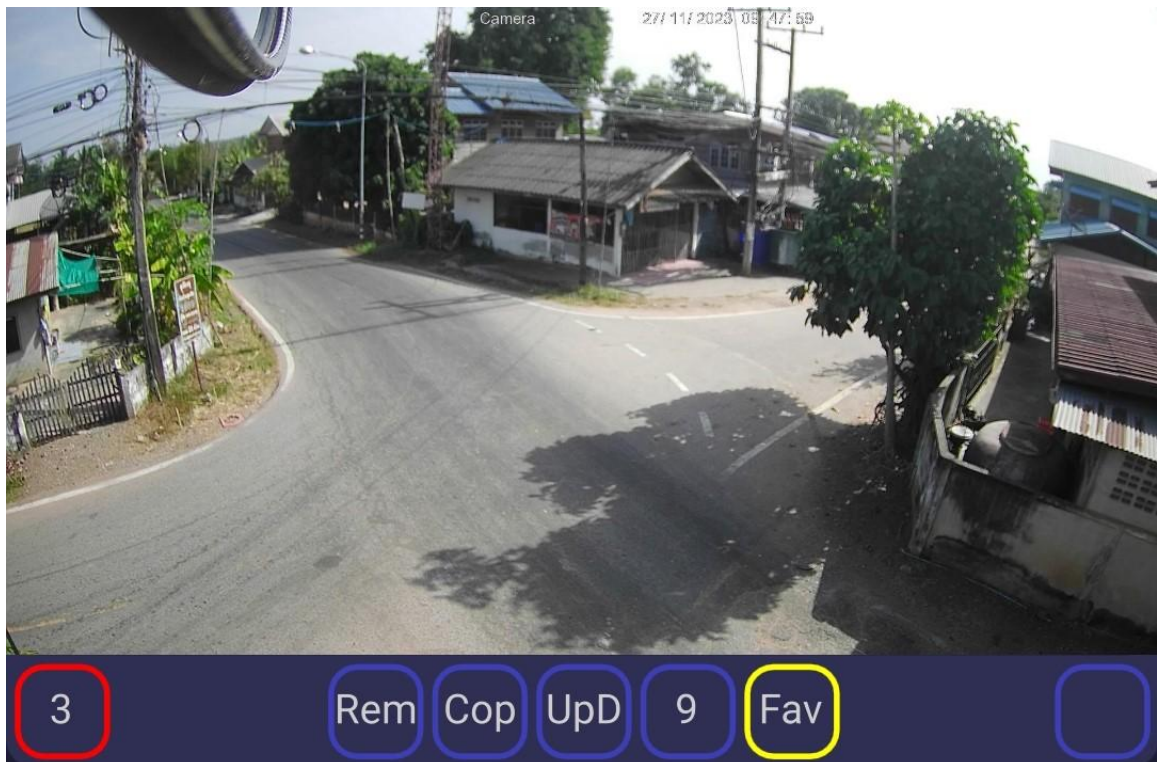


Рисунок 5.15 – Елемент списку трансляцій

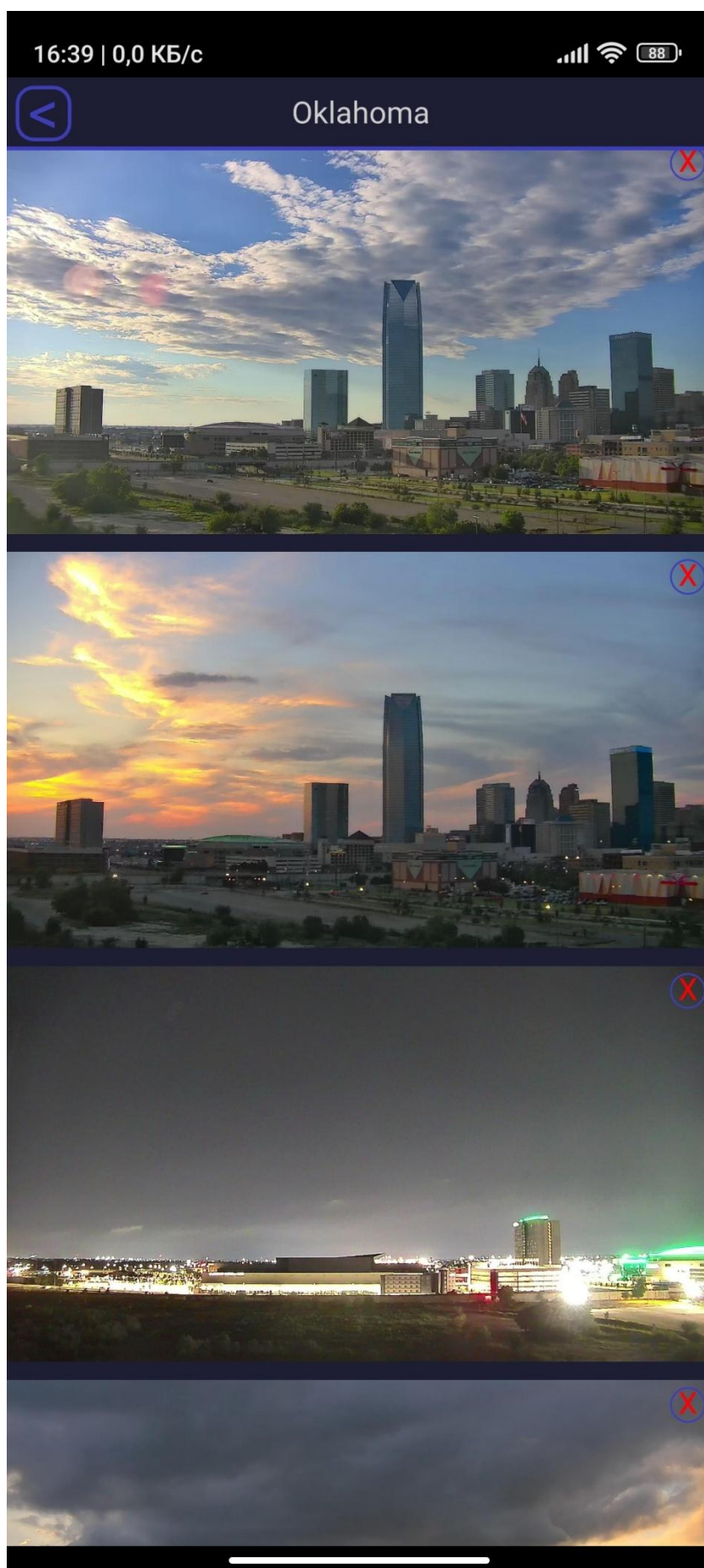


Рисунок 5.16 – Вікно перегляду історії знімків

При натисканні на зображення списку трансляцій виконується перехід до вікна відтворення трансляції (рис. 5.17).



Рисунок 5.17 – Вікно відтворення трансляції

При натисканні на екран під час відтворення відео трансляції, в лівій частині екрана на три секунди з'являється кнопка із символом “S” (рис. 5.18), при натисканні на яку вмикається режим збільшення різкості, при активному режимі збільшення різкості кнопка має жовтий колір (рис. 5.19).



Рисунок 5.18 – Вікно відтворення трансляції без збільшення різкості



Рисунок 5.19 – Вікно відтворення трансляції зі збільшенням різкості

При запуску процесу перевірка активності трансляцій, в елементі “bottom sheet peek” відображатиметься прогрес у відсотках (рис. 5.20). Також активна кнопка зупинки процесу.

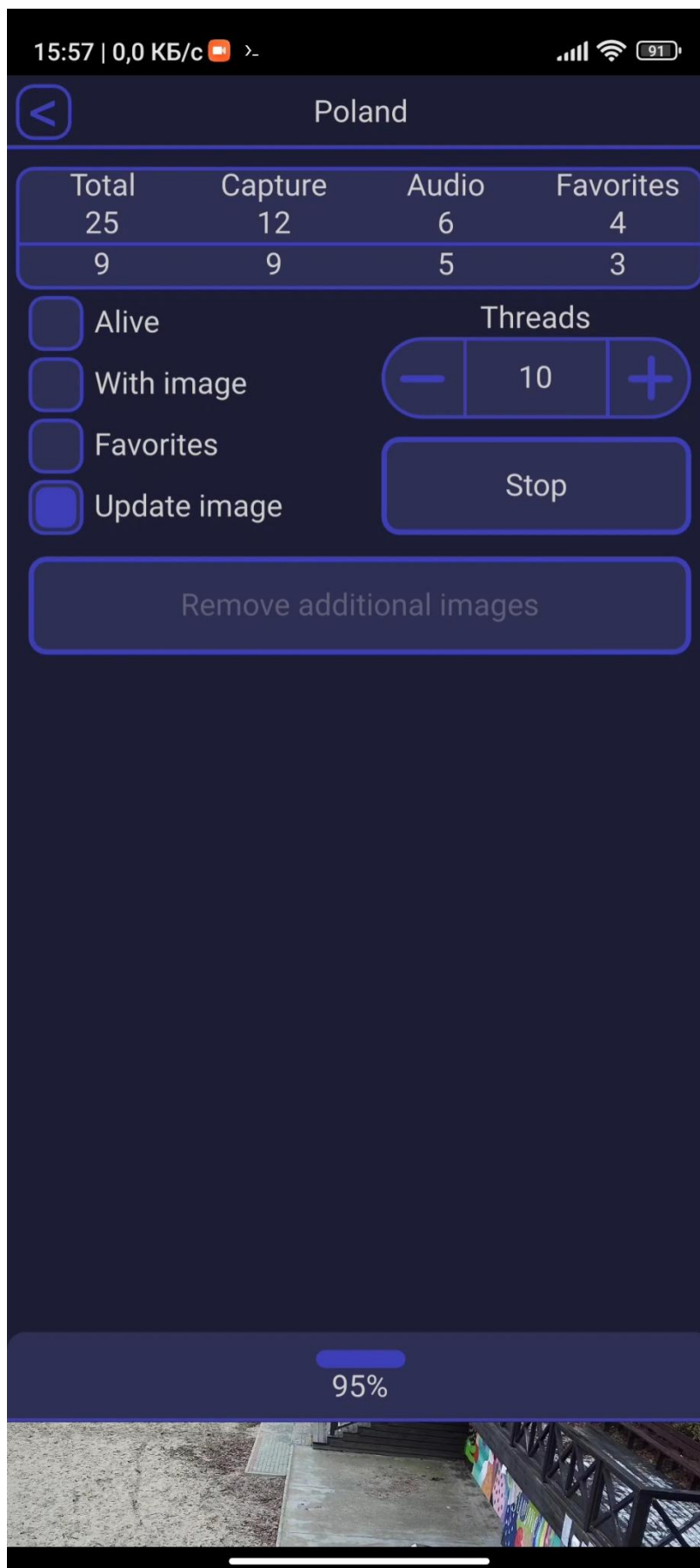


Рисунок 5.20 – Процес перевірки активності трансляцій активний з відображенням прогресу

Кожна дія пов'язана з видаленням супроводжується відображенням діалогового вікна з проханням підтвердити дію (рис. 5.21).

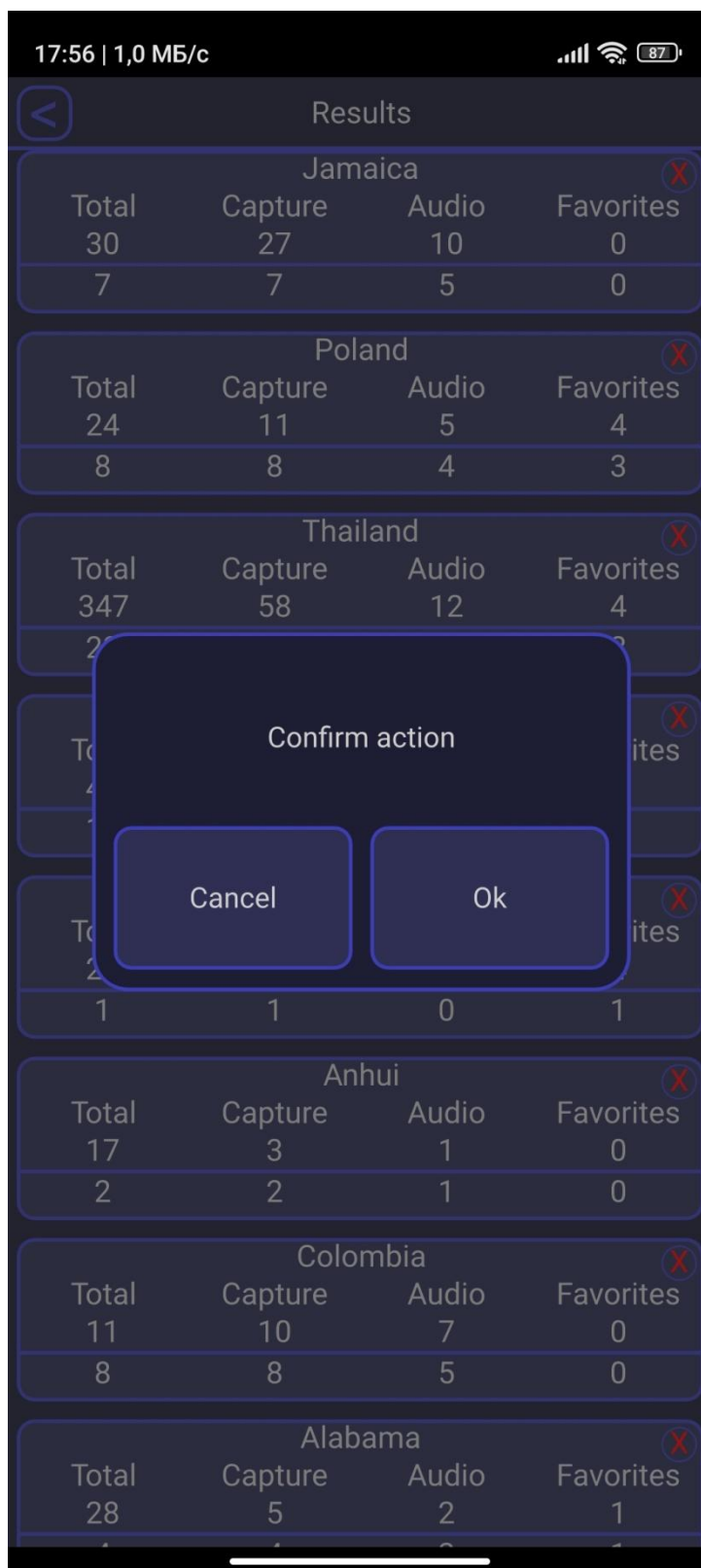


Рисунок 5.21 – Діалогове вікно для підтвердження дії видалення результату сканування

5.4 Повідомлення оператору

Кожна дія пов'язана з видаленням супроводжується відображенням діалогового вікна з проханням підтвердити дію (рис. 5.22).

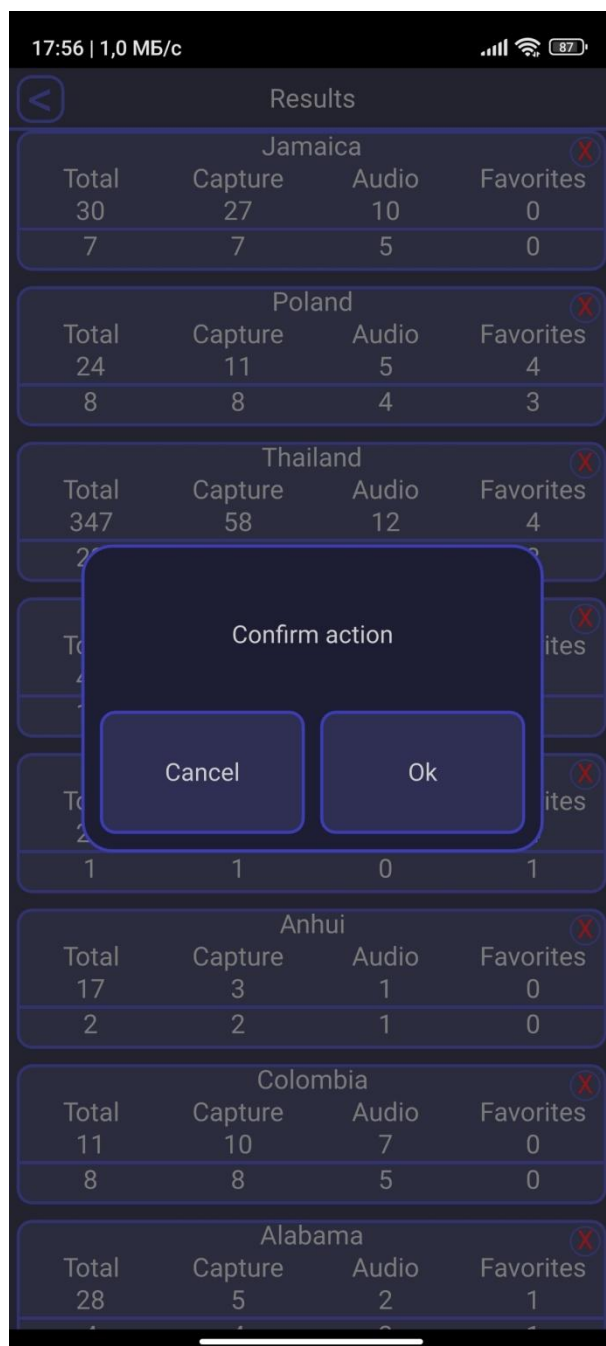


Рисунок 5.22 – Діалогове вікно для підтвердження дії видалення результату сканування

6 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

Тестування та експериментальне дослідження методів збільшення різкості буде проводитись засобами мови програмування Python.

Метою тестування є пошук найшвидшого методу для обробки зображення у реальному часі.

6.1 Тестування методу матриці згортки різкості

Тестування методу матриці згортки різкості виконується з використанням бібліотеки cv2 та її функції filter2D.

Вимірювання часу обробки зображення виконується за допомогою бібліотеки time та її функції time, за допомогою якої береться поточний час перед запуском та після завершення функції filter2D. Час виконання буде відображатися в секундах.

В якості тестових даних виступають чотири зображення різного формату: HD, Full HD, Quad HD, 4K UHD.

В якості матриці згортки використовувалася матриця зображена на рисунку 1.1, першого розділу.

Час обробки зображень наведено у таблиці 6.1.

Таблиця 6.1 – Час обробки зображень різного формату методом матриці згортки різкості

	Формат			
	HD	Full HD	Quad HD	4K UHD
Час у секундах	0.00199	0.00501	0.00799	0.01999

Результат роботи методу на прикладі зображення формату Full HD до та після застосування методу зображено на рисунку 6.1.



Рисунок 6.1 – Результат застосування матриці згортки різкості на прикладі зображення до та після

6.2 Тестування методу нерізкої маски

Тестування методу нерізкої маски виконується з використанням бібліотеки cv2 та її функцій GaussianBlur та addWeighted.

Вимірювання часу обробки зображення виконується за допомогою бібліотеки time та її функції time, за допомогою якої береться поточний час перед запуском функції GaussianBlur та після завершення addWeighted. Час виконання буде відображатися в секундах.

В якості тестових даних виступають чотири зображення різного формату: HD, Full HD, Quad HD, 4K UHD.

Як параметри методу використовувалися: радіус - 2, інтенсивність - 2.

Час обробки зображень наведено у таблиці 6.2.

Таблиця 6.2 – Час обробки зображень різного формату методом нерізкої маски

	Формат			
	HD	Full HD	Quad HD	4K UHD
Час у секундах	0.00699	0.00799	0.01399	0.02499

Результат роботи методу на прикладі зображення формату Full HD до та після застосування методу зображено на рисунку 6.2.



Рисунок 6.2 – Результат застосування методу нерізкої маски на прикладі зображення до та після

6.3 Тестування методу вейвлетного збільшення різкості

Тестування методу вейвлетного збільшення різкості виконується з використанням бібліотек cv2 та pywt.

Функції бібліотеки cv2: split - для розділення зображення на канали RGB, merge - для об'єднання каналів RGB.

Функції бібліотеки pywt: wavedec2 - для виконання 2D-вейвлет-перетворення кожного з каналів RGB, коефіцієнти яких будуть помножені на радіус, wavegrec2- для виконання зворотного вейвлет-перетворення каналів RGB.

Вимірювання часу обробки зображення виконується за допомогою бібліотеки time та її функції time, за допомогою якої береться поточний час перед запуском функції split та після завершення merge. Час виконання буде відображатися в секундах.

В якості тестових даних виступають чотири зображення різного формату: HD, Full HD, Quad HD, 4K UHD.

Як параметри методу використовувалися: радіус - 1.9, рівень вейвлет-перетворення - 2.

Час обробки зображень наведено у таблиці 6.3.

Таблиця 6.3 – Час обробки зображень різного формату методом нерізкої маски

	Формат			
	HD	Full HD	Quad HD	4K UHD
Час у секундах	0.17394	0.46086	0.92571	2.34823

Результат роботи методу на прикладі зображення формату Full HD до та після застосування методу зображено на рисунку 6.3.



Рисунок 6.3 – Результат застосування вейвлетного збільшення різкості на прикладі зображення до та після

6.4 Висновок за розділом 6

У розділі засобами мови Python були протестовані такі методи збільшення різкості як: матриця згортки різкості, нерізка маска і вейвлетне збільшення різкості. Тестування проводилося на чотирьох зображеннях різного формату: HD, Full HD, Quad HD, 4K UHD.

Найкращий результат швидкості виконання обробки показав метод матриці згортки різкості, швидкість обробки зображень різного формату у секундах протестованими методами наведено в таблиці 6.4.

Таблиця 6.4 – Швидкість обробки зображень різного формату у секундах протестованими методами

	Формат			
	HD	Full HD	Quad HD	4K UHD
Матриця згортки різкості	0.00199	0.00501	0.00799	0.01999
Нерізка маска	0.00699	0.00799	0.01399	0.02499
Вейвлетне збільшення різкості	0.17394	0.46086	0.92571	2.34823

Грунтуючись на проведених дослідженнях методів збільшення чіткості зображення засобами мови Python, можна зробити висновок про те, що найбільш підходящим методом є матриця згортки різкості, завдяки високій швидкості обробки, що особливо актуально для обробки зображення в реальному часу.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено програмне забезпечення під операційну систему Android та архітектуру процесора ARM8, яке виконує пошук та відображення відкритих RTSP-трансляцій з можливістю збільшення чіткості зображення, що відображається в реальному часі.

Основою розробки стали результати дослідження протоколів потокового відео, методів збільшення чіткості зображення та існуючих засобів пошуку відкритих відеотрансляцій.

В результаті дослідження протоколів потокового відео був обраний протокол RTSP, завдяки стандартно виділеному 554 порту і великій кількості пристроїв, які підтримують трансляцію по даному протоколу, що дозволило ефективно виконувати пошук та відображення відеотрансляцій завдяки підтримці даного протоколу багатьма бібліотеками. Для роботи з потоком було обрано бібліотеку FFmpeg.

При огляді протоколу RTSP було обрано метод DESCRIBE для виконання запитів для пошуку відкритих трансляцій.

В результаті дослідження методів збільшення чіткості зображення засобами мови Python, було зроблено висновок, що найбільш підходящим методом є матриця згортки різкості, завдяки високій швидкості обробки, що особливо актуально для обробки зображення в реальному часу.

Для створення програми була використана мова програмування C++, таке рішення дозволило знизити кількість ресурсів, що споживаються програмою.

Як середовище розробки було використано Qt Creator.

Для створення графічного інтерфейсу користувача була використана декларативна мова програмування QML, такий вибір дозволив створити сучасний і красивий інтерфейс користувача, частина функціоналу якого була реалізована за допомогою мови JavaScript всередині самого інтерфейсу.

Зберігання даних було реалізовано за допомогою серіалізації та

десеріалізації структур. Такий спосіб зберігання даних дозволив прискорити зчитування та запис даних, що позитивно позначилося на швидкості роботи програми.

Тестування розробленої програми показало повну та стабільну працездатність.

Результати роботи можна використовувати як досвід у подальших проєктах, пов'язаних з розробкою програмного забезпечення для роботи з відеопотоками. Також код програми можна використовувати в інших схожих проєктах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Протоколи для стрімінгу [Електрон. ресурс]. – Режим доступу: https://www.cdnvideo.ru/blog/protokoly_dlya_striminga/
2. Плюси та мінуси HTTP Live Streaming [Електрон. ресурс]. – Режим доступу: <https://facecast.net/ru/blog/http-live-streaming/>
3. Технологія адаптивної потокової передачі даних MPEG-DASH [Електрон. ресурс]. – Режим доступу: <https://flussonic.ru/glossary/mpeg-dash/>
4. WebRTC [Electronic resource]. – Access mode: <https://ru.wikipedia.org/wiki/WebRTC>
5. Основи WebRTC, визначення та принцип роботи [Електрон. ресурс]. – Режим доступу: <https://webinar.ru/articles/webrtc/>
6. Розуміння протоколу SRT, остаточне керівництво [Електрон. ресурс]. – Режим доступу: <https://cqr.company/ru/wiki/protocols/understanding-srt-protocol-the-ultimate-guide-for-reliable-video-streaming/>
7. Real Time Streaming Protocol [Electronic resource]. – Access mode: <https://ru.wikipedia.org/wiki/RTSP>
8. Shodan [Electronic resource]. – Access mode: [https://ru.wikipedia.org/wiki/Shodan_\(сайт\)](https://ru.wikipedia.org/wiki/Shodan_(сайт))
9. Мова програмування C++ [Електрон. ресурс]. – Режим доступу: <https://www.sravni.ru/kursy/info/yazyk-programmirovaniya-c/>
10. Qt [Electronic resource]. – Access mode: <https://ru.wikipedia.org/wiki/Qt>
11. Різниця між QtWidgets, QML, Qt Quick [Електрон. ресурс]. – Режим доступу: <https://russianblogs.com/article/9267206791/>
12. FFmpeg [Electronic resource]. – Access mode: <https://ru.wikipedia.org/wiki/FFmpeg>.

ДОДАТОК А

Текст програми

```

// networkscanner.cpp
bool NetworkScanner::connect_ex(const QString &hostName)
{
    QTcpSocket socket;
    socket.connectToHost(hostName, 554);
    auto start = std::chrono::high_resolution_clock::now();
    if (socket.waitForConnected(responseTimeout))
    {
        auto end = std::chrono::high_resolution_clock::now();
        std::chrono::duration<float> duration = end - start;
        float time = duration.count();

        socket.close();
        socket.abort();

        mutexResponseTimeStat.lock();
        if(time < minResponseTimeValue)
        {
            minResponseTimeValue = time;
            emit minResponseTimeChanged();
        }
        else if(time > maxResponseTimeValue)
        {
            maxResponseTimeValue = time;
            emit maxResponseTimeChanged();
        }
        responseTimeVector.push_back(time);
        mutexResponseTimeStat.unlock();

        return true;
    }
    socket.close();
    socket.abort();
    return false;
}

void NetworkScanner::turnOff()
{
    mutexTurnOff.lock();
    quint16 counter = 0;
    for (quint16 i = 0; i < numThreadPortScan; i++)
        portScanQueue.enqueue(0);

    while(counter != numThreadPortScan)
    {
        stopedThreads.dequeue();
        counter++;
    }
}

```

```

    }

    counter = 0;
    for (quint16 i = 0; i < numThreadBrute; i++)
        fromPortScanToBruteQueue.enqueue(0);

    while(counter != numThreadBrute)
    {
        stopedThreads.dequeue();
        counter++;
    }

    counter = 0;
    IpResult s;
    s.ip = 0;
    for (quint16 i = 0; i < numThreadCapture; i++)
        fromBruteToCaptureQueue.enqueue(s);

    while(counter != numThreadCapture)
    {
        stopedThreads.dequeue();
        counter++;
    }
    db->safeContinue(safeUnpassedFromPortScanQueue);
    safeFromQueues();

    threadsPortScan.clear();
    threadsBrute.clear();
    threadsCapture.clear();
    threadsPortScanID.clear();
    threadsBruteID.clear();
    responceTimeVector.clear();
    signal = 0;
    stopFillSignal = 0;
    mutexTurnOff.unlock();
}

void NetworkScanner::resultCatcherIpSenderThreadFunc()
{
    db->fillIpQueue(portScanQueue, maxIpInPortScanQueue);
    ipLeftAtStart = initialAmountIp - counterPasedIp;
    timer.start();
    while (true) {
        if (signal != 0) {
            while (signal == 1)
                std::this_thread::sleep_for(std::chrono::seconds(5));
            if (signal == -1)

```

```

        break;
    }
    safeFromQueues();
    calculateTimeStat();
    calculateAverageResponseTime();
    emit ipPasedChanged();
    emit ipLeftChanged();
    emit progressChanged();
    emit portScanQueueSizeChanged();
    if(fromPortScanToBruteQueue.getSize() != 0)
        emit pathCheckQueueSizeChanged();
    if (portScanQueue.getSize() <= 5000 && stopFillSignal == 0 ||
portScanQueue.isEmpty() && stopFillSignal == 0) {
        if (!db->fillIpQueue(portScanQueue, maxIpInPortScanQueue))
            stopFillSignal = 1;
    }
    else if(portScanQueue.isEmpty() && stopFillSignal != 0)
        break;
    std::this_thread::sleep_for(std::chrono::seconds(1));
}
turnOff();
emit ipPasedChanged();
emit ipLeftChanged();
emit progressChanged();
emit portScanQueueSizeChanged();
emit pathCheckQueueSizeChanged();
delete db;
db = nullptr;
counterPasedIp = 0;
emit scanEnded();
return;
}

void NetworkScanner::portScanThreadFunc()
{
    quint32 ipInt;
    QString ipStr;
    while (true) {
        ipInt = portScanQueue.dequeue();
        if (ipInt == 0){
            stopedThreads.enqueue(std::this_thread::get_id());
            return;
        }
        if(signal != 0)
        {
            while(signal == 1)
                std::this_thread::sleep_for(std::chrono::seconds(5));

```

```

        if (signal == -1)
        {
            safeUnpassedFromPortScanQueue.enqueue(ipInt);
            continue;
        }
    }

    ipStr = QHostAddress(ipInt).toString();
    if (connect_ex(ipStr))
    {
        mutexCounterOpenPort.lock();
        counterOpenPort++;
        emit openPortChanged();
        mutexCounterOpenPort.unlock();

        fromPortScanToBruteQueue.enqueue(ipInt);
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    mutexCounterPasedIp.lock();
    counterPasedIp++;
    mutexCounterPasedIp.unlock();
}

}

void NetworkScanner::pathCheckThreadFunc()
{
    quint32 ipInt;
    QString ipStr;
    quint16 res;
    while (true)
    {
        ipInt = fromPortScanToBruteQueue.dequeue();
        if (ipInt == 0){
            stopedThreads.enqueue(std::this_thread::get_id());
            return;
        }
        if(signal != 0)
        {
            while(signal == 1)
                std::this_thread::sleep_for(std::chrono::seconds(5));
            if (signal == -1)
            {
                safeUnpassedFromPortScanQueue.enqueue(ipInt);
                continue;
            }
        }
    }
}

```

```

ipStr = QHostAddress(ipInt).toString();

for(quint16 i = 0; i < combinations.size(); i++)
{
    res = checkPath(ipStr, combinations[i]);
    if(res == 200)
    {
        mutexCounterCode200.lock();
        counterCode200++;
        emit code200Changed();
        mutexCounterCode200.unlock();

        IpResult ipResult;
        ipResult.ip = ipInt;
        ipResult.url = combinations[i];
        fromBruteToCaptureQueue.enqueue(ipResult);
        break;
    }
    else if(res != 400 && res != 404)
        break;
    else
        std::this_thread::sleep_for(std::chrono::seconds(bruteTimeout));
}
}

void NetworkScanner::captureThreadFunc()
{
    quint8 res;
    QString ipStr;
    IpResult ipResult;
    //FFmpeg ffmpeg;
    while (true) {
        ipResult = fromBruteToCaptureQueue.dequeue();
        if (ipResult.ip == 0){
            stopedThreads.enqueue(std::this_thread::get_id());
            return;
        }
        if (signal != 0) {
            while(signal == 1)
                std::this_thread::sleep_for(std::chrono::seconds(5));
        }

        ipStr = QHostAddress(ipResult.ip).toString();

        QString fileName = ipStr;

```

```

        fileName.replace(".", "_");
        fileName += "_" + QString::number(ipResult.counterImages + 1);
        FFmpeg ffmpeg;

        std::this_thread::sleep_for(std::chrono::seconds(10));
        res = ffmpeg.capture("rtsp://" + ipStr + ":554" + ipResult.url, db-
>getImagesPath(), fileName);
        if(res != 0)
        {
            qDebug() << "rtsp://" + ipStr + ":554" + ipResult.url;
            ipResult.counterImages++;
            if(res == 2)
                ipResult.audio = true;

            mutexCounterCaptured.lock();
            counterCaptured++;
            emit capturedChanged();
            emit addItemModelImage("file:/// + db->getImagesPath() + fileName + ".jpg");
            mutexCounterCaptured.unlock();
        }
        else
            ipResult.counterNotAlive++;
        safeFromCaptureQueue.enqueue(ipResult);
    }
}

void NetworkScanner::applyThreadsQuantityThreadFunc()
{
    mutexTurnOff.lock();
    if(threadsPortScan.size() != newNumPortScanThreads)
        changePortScanThreadsQuantity();
    if(threadsBrute.size() != newNumBruteThreads)
        changeBruteThreadsQuantity();

    mutexTurnOff.unlock();
    qDebug() << threadsPortScan.size();
    qDebug() << threadsBrute.size();
    emit applied();
}

void NetworkScanner::calculateTimeStat()
{
    quint32 milisec = timer.elapsed();
    float time = milisec / 1000;
    if(time > 5)
    {

```

```

        timeLeftMin = (((initialAmountIp - counterPasedIp) / (ipLeftAtStart -
(initialAmountIp - counterPasedIp))) * time) / 60;
        counteripPerMinute = (60 / time) * (ipLeftAtStart - (initialAmountIp -
counterPasedIp));
        ipLeftAtStart = initialAmountIp - counterPasedIp;
        timer.restart();

        emit timeLeftChanged();
        emit ipPerMinuteChanged();
    }
}

```

```

void NetworkScanner::calculateAverageResponseTime()
{
    averageResponseTimeValue = 0;
    mutexResponseTimeStat.lock();
    for(float item : responseTimeVector)
        averageResponseTimeValue += item;
    averageResponseTimeValue /= responseTimeVector.size();
    if(responseTimeVector.size() >= 500)
        responseTimeVector.clear();
    mutexResponseTimeStat.unlock();
    emit averageResponseTimeChanged();
}

```

```

void NetworkScanner::safeFromQueues()
{
    IpResult res;

    while(safeFromCaptureQueue.getSize() != 0)
    {
        res = safeFromCaptureQueue.dequeue();
        db->addResult(res);
    }
}

```

```

quint16 NetworkScanner::checkPath(const QString &hostName, QString &path)
{
    QTcpSocket socket;
    socket.connectToHost(hostName, 554);
    quint16 code = 0;
    if (socket.waitForConnected(1500))
    {
        QString rec;
        QString rtspUrl = "rtsp://" + hostName + path;

        socket.write(rec.toLocal8Bit().data());
    }
}

```

```

socket.waitForBytesWritten(1500);
if(socket.waitForReadyRead(1500))
{
    QString respString = QString(socket.readAll());
    if(respString.contains("200 ok", Qt::CaseInsensitive))
        code = 200;
    else if(respString.contains("Bad Request", Qt::CaseInsensitive))
        code = 400;
    else if(respString.contains("Not Found", Qt::CaseInsensitive))
        code = 404;
    else if(respString.contains("Unauthorized", Qt::CaseInsensitive))
        code = 401;
    else
        code = 2;
}
else
    code = 1;

socket.close();
socket.abort();
return code;
}

socket.close();
socket.abort();
return code;
}

void NetworkScanner::changePortScanThreadsQuantity()
{
    if(threadsPortScan.size() > newNumPortScanThreads)
    {
        for(quint16 i = threadsPortScan.size() - newNumPortScanThreads; i > 0; i--)
            portScanQueue.enqueueFront(0);
        for(quint16 i = threadsPortScan.size() - newNumPortScanThreads; i > 0; i--)
        {
            std::thread::id threadId = stopedThreads.dequeue();
            std::vector<std::thread::id>::iterator j = threadsPortScanID.begin();
            for(std::vector<std::thread>::iterator x = threadsPortScan.begin(); x !=
threadsPortScan.end(); x++, j++)
            {
                if(*j == threadId)
                {
                    threadsPortScan.erase(x);
                    threadsPortScanID.erase(j);
                    break;
                }
            }
        }
    }
}

```

```

        }
    }
    else
    {
        for(quint16 i = newNumPortScanThreads - threadsPortScan.size(); i > 0; i--)
        {
            threadsPortScan.push_back(std::thread(&NetworkScanner::portScanThreadFunc,
this));
            threadsPortScanID.push_back(threadsPortScan[threadsPortScan.size() -
1].get_id());
            threadsPortScan[threadsPortScan.size() - 1].detach();
        }
    }
    numThreadPortScan = threadsPortScan.size();
    newNumPortScanThreads = 0;
}

void NetworkScanner::changeBruteThreadsQuantity()
{
    if(threadsBrute.size() > newNumBruteThreads)
    {
        for(quint16 i = threadsBrute.size() - newNumBruteThreads; i > 0; i--)
            fromPortScanToBruteQueue.enqueueFront(0);
        for(quint16 i = threadsBrute.size() - newNumBruteThreads; i > 0; i--)
        {
            std::thread::id threadId = stopedThreads.dequeue();
            std::vector<std::thread::id>::iterator j = threadsBruteID.begin();
            for(std::vector<std::thread>::iterator x = threadsBrute.begin(); x !=
threadsBrute.end(); x++, j++)
            {
                if((*j) == threadId)
                {
                    threadsBrute.erase(x);
                    threadsBruteID.erase(j);
                    break;
                }
            }
        }
    }
    else
    {
        for(quint16 i = newNumBruteThreads - threadsBrute.size(); i > 0; i--)
        {
            threadsBrute.push_back(std::thread(&NetworkScanner::pathCheckThreadFunc,
this));
            threadsBruteID.push_back(threadsBrute[threadsBrute.size() - 1].get_id());

```

```

        threadsBrute[threadsBrute.size() - 1].detach();
    }
}
numThreadBrute = threadsBrute.size();
newNumBruteThreads = 0;
}

QString NetworkScanner::numberFormatting(QString numStr)
{
    QString res;
    if(numStr.size() <= 3)
        return numStr;
    for(quint8 i = numStr.size() / 3; i > 0; i--)
    {
        //res.push_front(numStr.last(3)); ver: 6.0
        for(quint8 j = 0, j2 = numStr.size(); j < 3; j++, j2--)
            res.push_front(numStr[j2 - 1]);
        numStr = numStr.remove(numStr.size() - 3, numStr.size());
        if(numStr.size() < 3 && numStr.size() != 0)
        {
            res.push_front(" ");
            res.push_front(numStr);
            return res;
        }
        if(numStr.size() == 0)
        {
            return res;
        }
        res.push_front(" ");
    }
}

void NetworkScanner::setScanningRanges(QString scanName)
{
    if(db == nullptr)
        db = new DataManager();
    this->scanName = scanName;
    initialAmountIp = db->getIpAmount(scanName);
    emit ipLeftChanged();
}

void NetworkScanner::start(quint16 numThreadPortScan, quint16 numThreadBrute, quint16
responseTimeout, quint32 maxCombinations)
{
    counterOpenPort = 0;
    counterCode200 = 0;
    counterResult = 0;

```

```

counterCaptured = 0;
counteripPerMinute = 0;
combinations.clear();

minResponseTimeValue = 100;
averageResponseTimeValue = 0;
maxResponseTimeValue = -1;
signal = 0;
stopFillSignal = 0;
db->setUpScanningData(scanName);

this->numThreadBrute = numThreadBrute;
this->numThreadPortScan = numThreadPortScan;
this->responceTimeout = responceTimeout;
db->loadPaths(combinations, maxCombinations);

threadResultCatcherIpSender
std::thread(&NetworkScanner::resultCatcherIpSenderThreadFunc, this);
threadResultCatcherIpSender.detach();

for(quint16 i = 0; i < numThreadPortScan; i++)
{
    threadsPortScan.push_back(std::thread(&NetworkScanner::portScanThreadFunc, this));
    threadsPortScanID.push_back(threadsPortScan[i].get_id());
    threadsPortScan[i].detach();
}

for(quint16 i = 0; i < numThreadBrute; i++)
{
    threadsBrute.push_back(std::thread(&NetworkScanner::pathCheckThreadFunc, this));
    threadsBruteID.push_back(threadsBrute[i].get_id());
    threadsBrute[i].detach();
}

for(quint16 i = 0; i < numThreadCapture; i++)
{
    threadsCapture.push_back(std::thread(&NetworkScanner::captureThreadFunc, this));
    threadsCapture[i].detach();
}
}

void NetworkScanner::loadIpRangesModel()
{
    QVector<QString> names;
    DataManager dbObj;
    dbObj.getFileNamesIpRanges(names);

```

```

for(auto item : names)
{
    if(item == "")
        continue;
    emit addModelItemIpRanges(item);
}
}

void NetworkScanner::apply(quint16 numThreadPortScan, quint16 numThreadBrute, quint16
responceTimeout)
{
    this->responceTimeout = responceTimeout;
    if(threadsPortScan.size() != numThreadPortScan || threadsBrute.size() !=
numThreadBrute)
    {
        newNumPortScanThreads = numThreadPortScan;
        newNumBruteThreads = numThreadBrute;
        threadApply = std::thread(&NetworkScanner::applyThreadsQuantityThreadFunc, this);
        threadApply.detach();
    }
    else
        emit applied();
}

// метод convolutionMatrix, який застосовується до кадру для збільшення його різкості
void QmlAVVideoFrame::convolutionMatrix(quint32 imageWidthStart, quint32 imageWidthEnd)
{
    for (quint32 x = imageWidthStart; x < imageWidthEnd; x++)
    {
        for (quint32 y = 0; y < imgHeight; y++)
        {
            double redSum = 0, greenSum = 0, blueSum = 0, kernelSum = 0;

            for (quint32 i = 0; i < kernelWidth; i++)
            {
                for (quint32 j = 0; j < kernelHeight; j++)
                {

                    quint32 pixXPos = x + (i - (kernelWidth / 2));
                    quint32 pixYPos = y + (j - (kernelHeight / 2));

                    if ((pixYPos >= imgHeight) || (pixXPos >= imgWidth) || (pixYPos < 0)
|| (pixXPos < 0))
                        continue;

                    uchar red = bitMapOrig[3 * (imgWidth * pixYPos + pixXPos) + 0];

```

```

        uchar green = bitMapOrig[3 * (imgWidth * pixYPos + pixXPos) + 1];
        uchar blue = bitMapOrig[3 * (imgWidth * pixYPos + pixXPos) + 2];

        double kernelVal = kernel[(i * kernelHeight) + j];

        redSum += red * kernelVal;
        greenSum += green * kernelVal;
        blueSum += blue * kernelVal;

        kernelSum += kernelVal;
    }
}

if (kernelSum <= 0) kernelSum = 1;

blueSum /= kernelSum;
if (blueSum > 255) blueSum = 255;
if (blueSum < 0) blueSum = 0;

redSum /= kernelSum;
if (redSum > 255) redSum = 255;
if (redSum < 0) redSum = 0;

greenSum /= kernelSum;
if (greenSum > 255) greenSum = 255;
if (greenSum < 0) greenSum = 0;

bitMap[3 * (imgWidth * y + x) + 0] = (uchar)redSum;
bitMap[3 * (imgWidth * y + x) + 1] = (uchar)greenSum;
bitMap[3 * (imgWidth * y + x) + 2] = (uchar)blueSum;
}
}

//resultsmanager.cpp
void ResultsManager::resultCatcherIpSenderThreadFunc(bool alive, bool withImg, bool
favorites, quint8 numThreads)
{
    db.setUpScanningData(scanName);
    db.fillUpdateQueue(updateQueue, favorites, alive, withImg);
    initialAmountIp = updateQueue.getSize();
    counterPasedIp = 0;

    IpResult ipResult;
    ipResult.ip = 0;
    QString imagePath = db.getImagesPath();
    for(quint8 i = 0; i < numThreads; i++)

```

```

{
    updateQueue.enqueue(ipResult);
    std::thread threadUpdate = std::thread(&ResultsManager::updateThreadFunc, this);
    threadUpdate.detach();
}
while(numThreads != 0)
{
    ipResult = resultQueue.dequeue();
    if(ipResult.ip == 0)
    {
        numThreads--;
        continue;
    }
    db.addChangeRes(ipResult);
    if(ipResult.counterNotAlive == 0)
    {
        QString ipStr = QHostAddress(ipResult.ip).toString();
        QString fileName = ipStr;
        fileName.replace(".", "_");
        fileName += "_" + QString::number(ipResult.counterImages);
        emit addItemModelImageUpdated("file:///\" + imagePath + fileName + ".jpg",
                                         "rtsp://\" + ipStr + ":" + "554" + ipResult.url,
                                         QString::number(ipResult.counterImages),
                                         QString::number(ipResult.counterNotAlive),
                                         ipResult.ip,
                                         ipResult.audio,
                                         ipResult.inFavorites);
    }
    counterPasedIp++;
    emit progressChanged();
}
db.safeAll();
updateInfo();
QVector<IpResult> temp;
if(favorites)
{
    db.getResultWithImageAliveNotFavorite(temp, scanName);
    for(quint32 i = 0; i < temp.size(); i++)
    {
        QString ipStr = QHostAddress(temp[i].ip).toString();
        QString fileName = ipStr;
        fileName.replace(".", "_");
        fileName += "_" + QString::number(temp[i].counterImages);
        emit addItemModelImageAllIp("file:///\" + imagePath + fileName + ".jpg",
                                         "rtsp://\" + ipStr + ":" + "554" + temp[i].url,
                                         QString::number(temp[i].counterImages),
                                         QString::number(temp[i].counterNotAlive),

```

```

        temp[i].ip,
        temp[i].audio,
        temp[i].inFavorites);
    }

}

temp.clear();
db.getResultWithImageNotAlive(temp, scanName);
for(quint32 i = 0; i < temp.size(); i++)
{
    QString ipStr = QHostAddress(temp[i].ip).toString();
    QString fileName = ipStr;
    fileName.replace(".", "_");
    fileName += "_" + QString::number(temp[i].counterImages);

    emit addItemModelImageAllIp("file:/// " + imagePath + fileName + ".jpg",
                                "rtsp://" + ipStr + ":" + "554" + temp[i].url,
                                QString::number(temp[i].counterImages),
                                QString::number(temp[i].counterNotAlive),
                                temp[i].ip,
                                temp[i].audio,
                                temp[i].inFavorites);
}

emit updateEnded();
}

void ResultsManager::updateThreadFunc()
{
    IpResult ipResult;
    quint8 res;
    while(true)
    {
        ipResult = updateQueue.dequeue();
        if(ipResult.ip == 0)
        {
            resultQueue.enqueue(ipResult);
            return;
        }
        if(signal == -1)
            continue;

        QString ipStr = QHostAddress(ipResult.ip).toString();
        if(isUpdateImg)
        {
            if (connect_ex(ipStr))

```

```

{
    std::this_thread::sleep_for(std::chrono::seconds(10));
    QString fileName = ipStr;
    fileName.replace(".", "_");
    fileName += "_" + QString::number(ipResult.counterImages + 1);

    FFmpeg ffmpeg;
    res = ffmpeg.capture("rtsp://" + ipStr + ":554" + ipResult.url,
db.getImagePath(), fileName);
    if(res != 0)
    {
        if(res == 2)
        {
            ipResult.audio = true;
        }
        else if(ipResult.audio == true && res != 2)
        {
            ipResult.audio = false;
        }
        ipResult.counterImages++;
        ipResult.counterNotAlive = 0;
        resultQueue.enqueue(ipResult);
        continue;
    }
}
ipResult.counterNotAlive++;
resultQueue.enqueue(ipResult);
}
else
{
    res = checkCombination(ipStr, ipResult.url);
    if(res == 200)
    {
        if(ipResult.counterImages == 0)
        {
            std::this_thread::sleep_for(std::chrono::seconds(10));
            QString fileName = ipStr;
            fileName.replace(".", "_");
            fileName += "_" + QString::number(ipResult.counterImages + 1);

            FFmpeg ffmpeg;
            res = ffmpeg.capture("rtsp://" + ipStr + ":554" + ipResult.url,
db.getImagePath(), fileName);
            if(res != 0)
            {
                if(res == 2)
                {

```

```

        ipResult.audio = true;
    }
    ipResult.counterImages++;
    ipResult.counterNotAlive = 0;
    resultQueue.enqueue(ipResult);
    continue;
}
ipResult.counterNotAlive++;
resultQueue.enqueue(ipResult);
}
else
{
    ipResult.counterNotAlive = 0;
    resultQueue.enqueue(ipResult);
}
}
else
{
    ipResult.counterNotAlive++;
    resultQueue.enqueue(ipResult);
}
}
}
}

```

```

void ResultsManager::updateSinglResultThreadFunc(quint32 index, quint32 ip, QString url,
quint16 counterImages, quint16 counterNotAlive)

```

```

{
    FFmpeg ffmpeg;
    quint8 res;
    QString ipStr = QHostAddress(ip).toString();
    QString imgPath = db.getImagesPath();
    if (connect_ex(ipStr))
    {
        std::this_thread::sleep_for(std::chrono::seconds(2));
        QString fileName = ipStr;
        fileName.replace(".", "_");
        fileName += "_" + QString::number(counterImages + 1);

        res = ffmpeg.capture(url, db.getImagesPath(), fileName);
        if(res != 0)
        {
            counterImages++;
            counterNotAlive = 0;

            mutexDB.lock();
            db.setResAlive(scanName, ip, true);

```

```

        emit updateImageInfo(index, "file://" + imgPath + fileName + ".jpg",
QString::number(counterImages), QString::number(counterNotAlive));
        mutexDB.unlock();
        updateInfo();
        return;
    }
}

    counterNotAlive++;
    mutexDB.lock();
    db.setResAlive(scanName, ip, false);
    emit updateImageInfo(index, "", QString::number(counterImages),
QString::number(counterNotAlive));
    mutexDB.unlock();
    updateInfo();
    return;
}

void ResultsManager::updateInfo()
{
    mutexDB.lock();
    scanInfo = db.getScanInfo(scanName);
    emit totalResChanged();
    emit aliveResChanged();
    emit capturedChanged();
    emit aliveCapturedChanged();
    emit audioChanged();
    emit aliveAudioChanged();
    emit favoritesChanged();
    emit aliveFavoritesChanged();
    emit updateInfoResultsList(scanInfo.name,
                                numberFormatting(QString::number(scanInfo.totalRes)),
                                numberFormatting(QString::number(scanInfo.aliveRes)),
                                numberFormatting(QString::number(scanInfo.captured)),

    numberFormatting(QString::number(scanInfo.aliveCaptured)),
                                numberFormatting(QString::number(scanInfo.audio)),
                                numberFormatting(QString::number(scanInfo.aliveAudio)),
                                numberFormatting(QString::number(scanInfo.favorites)),

    numberFormatting(QString::number(scanInfo.aliveFavorites)));
    mutexDB.unlock();
}

void ResultsManager::start(bool alive, bool withImg, bool favorites, bool isUpdateImg,
quint8 numThreads)

```

```

{
    qDebug() << "Start";
    signal = 0;
    this->isUpdateImg = isUpdateImg;
    std::thread threadUpdate =
std::thread(&ResultsManager::resultCatcherIpSenderThreadFunc, this, alive, withImg,
favorites, numThreads);
    threadUpdate.detach();
}

void ResultsManager::setFavorite(quint32 ip, bool value)
{
    db.setFavorite(scanName, ip, value);
    updateInfo();
}

void ResultsManager::updateImage(quint32 index, quint32 ip, QString url, quint16
counterImages, quint16 counterNotAlive)
{
    std::thread threadUpdate = std::thread(&ResultsManager::updateSinglResultThreadFunc,
this, index, ip, url, counterImages, counterNotAlive);
    threadUpdate.detach();
}

void ResultsManager::loadResultsInfo()
{
    QVector<ScanInfo> b;
    //qDebug() << "fdfs";
    db.getAllScans(b);
    for(quint16 i = 0; i < b.size(); i++)
    {
        emit addItemModelResultInfo(b[i].name,
                                     numberFormatting(QString::number(b[i].totalRes)),
                                     numberFormatting(QString::number(b[i].aliveRes)),
                                     numberFormatting(QString::number(b[i].captured)),

                                     numberFormatting(QString::number(b[i].aliveCaptured)),
                                     numberFormatting(QString::number(b[i].audio)),
                                     numberFormatting(QString::number(b[i].aliveAudio)),
                                     numberFormatting(QString::number(b[i].favorites)),

                                     numberFormatting(QString::number(b[i].aliveFavorites)));
    }
}

void ResultsManager::loadResultsImages(bool withNoAlive)
{

```

```

 QVector<IpResult> v;
 db.getAllResultWithImage(v, scanName);
 QString imagePath = db.getImagesPath();
 for(quint32 i = 0; i < v.size(); i++)
 {
     QString ipStr = QHostAddress(v[i].ip).toString();
     QString fileName = ipStr;
     fileName.replace(".", "_");
     fileName += "_" + QString::number(v[i].counterImages);

     if(withNoAlive)
         emit addItemModelImageAllIp("file:/// " + imagePath + fileName + ".jpg",
                                         "rtsp://" + ipStr + ":" + "554" + v[i].url,
                                         QString::number(v[i].counterImages),
                                         QString::number(v[i].counterNotAlive),
                                         v[i].ip,
                                         v[i].audio,
                                         v[i].inFavorites);
     else if(v[i].counterNotAlive == 0)
     {
         emit addItemModelImageAllIp("file:/// " + imagePath + fileName + ".jpg",
                                         "rtsp://" + ipStr + ":" + "554" + v[i].url,
                                         QString::number(v[i].counterImages),
                                         QString::number(v[i].counterNotAlive),
                                         v[i].ip,
                                         v[i].audio,
                                         v[i].inFavorites);
     }
 }
}

void ResultsManager::loadAdditionalImages(quint32 ip, QString numImages)
{
    QString fileName = QHostAddress(ip).toString().replace(".", "_");
    for(quint16 i = 0; i < numImages.toUInt(); i++)
    {
        emit addItemModelImageSingleIp("file:/// " + db.getImagesPath() + fileName + "_" +
        QString::number(i + 1) + ".jpg");
    }
}

void ResultsManager::setCurrentResultInfo(QString scanName)
{
    this->scanName = scanName;
    updateInfo();
}

```

```

void ResultsManager::removeScanData(QString scanName)
{
    db.removeScanData(scanName);
}

void ResultsManager::removeAdditionalImages()
{
    QMap<quint32, QString> newPath;
    db.removeAdditionalImages(newPath);
    QMap<quint32, QString>::const_iterator i = newPath.constBegin();
    while (i != newPath.constEnd()) {
        emit updateImagePath(i.key(), i.value());
        ++i;
    }
}

void ResultsManager::removeItemResult(quint32 ip)
{
    db.removeItemResult(ip);
    updateInfo();
}

void ResultsManager::removeSingleAdditionalImage(quint32 ip, quint16 imageIndex)
{
    QMap<quint16, QString> newPath;
    db.removeSingleImage(ip, imageIndex, newPath);
    QMap<quint16, QString>::const_iterator i = newPath.constBegin();
    while (i != newPath.constEnd()) {
        emit updateAdditionalImagePath(i.key(), i.value());
        ++i;
    }
}

```

ДОДАТОК Б
Слайди презентації

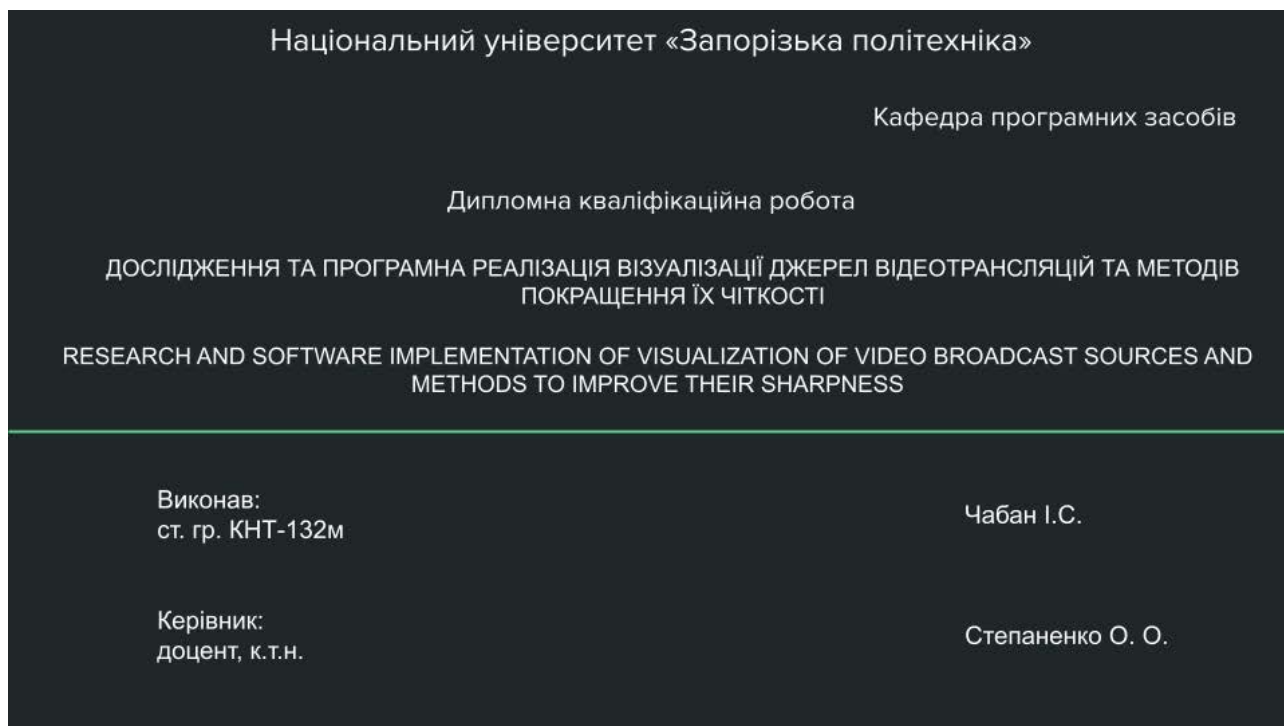


Рисунок Б.1 – Слайд №1

2

Об'єкт, предмет дослідження та мета роботи

- Об'єкт дослідження - методи пошуку та візуалізації джерел відеотрансляцій та покращення їх чіткості.
- Предмет дослідження - програмне забезпечення для пошуку та візуалізації джерел відеотрансляцій та покращення їх чіткості.
- Мета роботи - розробка програмного забезпечення для пошуку та візуалізації джерел відеотрансляцій та покращення їх чіткості.

Рисунок Б.2 – Слайд №2

Вибір протоколу

3

Для пошуку та відображення відеотрансляцій було обрано протокол RTSP

Переваги протоколу RTSP:

- стандартно виділений 554 порт
- підтримується більшістю пристроїв
- підтримується більшістю бібліотек



Результат пошукового запиту у Shodan по відкритому 554 порту та протоколу RTSP

Рисунок Б.3 – Слайд №3

Вибір методу збільшення різкості зображення

4

Для збільшення різкості зображення було обрано метод матриці згортки різкості



Рисунок Б.4 – Слайд №4

Вибір запиту для пошуку відкритих трансляцій

5

Для пошуку відкритих RTSP трансляцій обрано запит DESCRIBE

```
<===== Request =====>
"DESCRIBE rtsp://192.168.50.73/h264_pcm.sdp RTSP/1.0"
"CSeq: 1"
"User-Agent: RTSP Client"
"Accept: application/sdp"
<===== Response =====>
"RTSP/1.0 200 OK"
"Server: IP Webcam Server 0.4"
"CSeq: 1"
```

Запит DESCRIBE до відкритої трансляції з правильним шляхом,
відповідь 200 OK

```
<===== Request =====>
"DESCRIBE rtsp://192.168.50.73/h264 RTSP/1.0"
"CSeq: 1"
"User-Agent: RTSP Client"
"Accept: application/sdp"
<===== Response =====>
"RTSP/1.0 404 Not Found"
"Server: IP Webcam Server 0.4"
"CSeq: 1"
```

Запит DESCRIBE до відкритої трансляції з неправильним шляхом,
відповідь 404 Not Found

```
<===== Request =====>
"DESCRIBE rtsp://192.168.50.73/h264 RTSP/1.0"
"CSeq: 1"
"User-Agent: RTSP Client"
"Accept: application/sdp"
<===== Response =====>
"RTSP/1.0 401 Unauthorized"
"CSeq: 1"
"Content-Length: 0"
"WWW-Authenticate: Digest realm=\"IP Webcam\", nonce=\"1684180203\", qop=\"auth\""
```

Запит DESCRIBE до закритої трансляції з правильним шляхом,
відповідь 401 Unauthorized

Рисунок Б.5 – Слайд №5

6

Вибір мови програмування

Критерії	Мова програмування			
	Java	C++ (QT framework)	C#	Python
Висока швидкість розробки	-	+-	+-	+
Читабельність коду	+	+	+	-
Швидкість виконання коду	+-	+	+	-
Велика кількість готових бібліотек	-	-	-	+
Кросплатформність	+-	+	+-	+-

Рисунок Б.6 – Слайд №6

Вибір бібліотеки для отримання трансляції RTSP протоколу

7

Для отримання RTSP-трансляції було обрано бібліотеку FFmpeg.

Переваги бібліотеки FFmpeg:

- підтримка великої кількості форматів
- кроссплатформеність
- висока продуктивність
- розширюваність
- легкість у використанні

Рисунок Б.7 – Слайд №7

Вибір середовища розробки

8

Критерії	MS Visual Studio	Qt Creator IDE	Eclipse IDE
Вбудований відладчик	+	+	+
Переносимість середовища на різні платформи	+	+	+
Встановлене ПЗ на підприємстві та наявність ліцензії	+	+	-
Досвід роботи	+	+	-
Безкоштовний доступ	+	+	+

Рисунок Б.8 – Слайд №8

Алгоритм згортки різкості кадрів відео трансляції

11

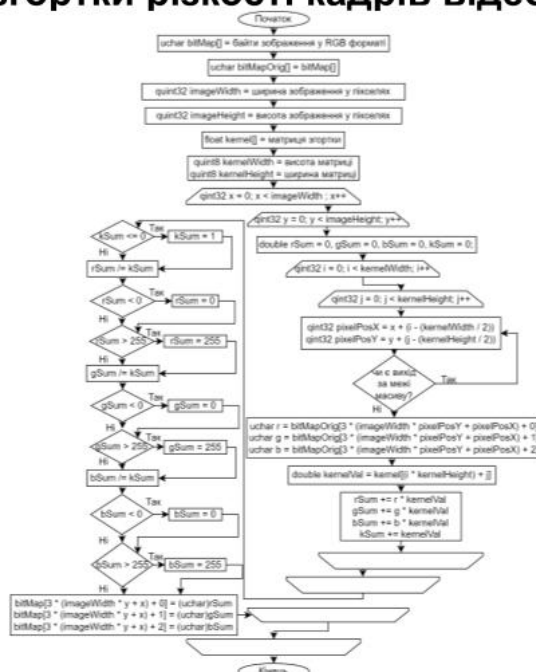


Рисунок Б.11 – Слайд №11

Файлова структура робочої директорії програми

12

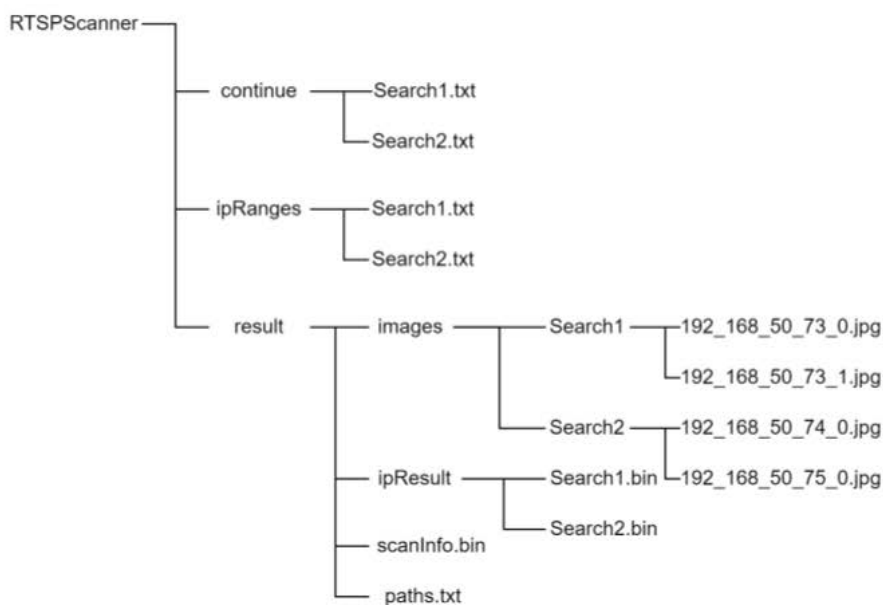


Рисунок Б.12 – Слайд №12

Зберігання даних

IpResult

ip: quint32

url: QString

counterImages: quint16

counterNotAlive: quint8

audio: bool

inFavorites: bool

Діаграма структури IpResult з назвою полів та їх типом

ScanInfo

name: QString

totalRes: quint32

aliveRes: quint32

captured: quint32

aliveCaptured: quint32

audio: quint32

aliveAudio: quint32

favorites: quint32

aliveFavorites: quint32

Діаграма структури ScanInfo з назвою полів та їх типом

1.0.0.0-1.0.0.255

1.0.4.0-1.0.7.255

1.1.1.0-1.1.1.255

1.2.3.0-1.2.3.255

1.4.0.0-1.4.0.255

1.16.64.0-1.18.115.255

1.40.0.0-1.40.0.255

1.40.1.0-1.40.5.255

1.40.6.0-1.40.6.255

1.40.7.0-1.40.11.255

1.40.12.0-1.40.12.255

1.40.13.0-1.40.13.255

1.40.14.0-1.40.14.255

1.40.15.0-1.40.28.255

1.40.29.0-1.40.29.255

Приклад файлу з діапазоном IP-адрес

/11

/mpeg4/ch01/main/av_stream

/cam/realmonitor?channel=1&subtype=0

/Streaming/Channels/101

/axis-media/media.amp

/stream1

/live1.sdp

Приклад файлу із шляхами до ресурсу трансляцій

Рисунок Б.13 – Слайд №13

Інтерфейс програми

15:43 | 0.0 KB/s

Anhui

Australia

California

Colombia

Mexico

Poland

Spain

Thailand

Port scan threads: 10

Path check threads: 5

Paths: 4

Response timeout: 1

Apply

Stop filling

Pause

Code 200: 0

Captured: 0

Список з назвами файлів IP-діапазонів

15:44 | 0.0 KB/s

Thailand

Time left: 38h 29min

IP passed: 3 461

Port is open: 11

IP-scan queue size: 46 422

Path check queue size: 2

Response time stats: Min 0.213796, Average 0.241192, Max 0.293975

Port scan threads: 140

Path check threads: 5

Paths: 4

Response timeout: 1

Apply

Stop filling

Pause

Code 200: 3

Captured: 3

Активний процес пошуку та змінні параметри з активною кнопкою Apply

15:44 | 1.3 KB/s

Code 200: 3

Captured: 3

Time left: 5h 16min

IP passed: 11 228

Port is open: 28

IP-scan queue size: 0

Path check queue size: 0

Response time stats: Min 0.174916, Average 0.27158, Max 0.502101

Port scan threads: 400

Path check threads: 20

Paths: 4

Response timeout: 1

Apply

Stop filling

Pause

Code 200: 3

Captured: 3

Розгорнутий список знімків знайдених трансляцій

15:44 | 1.3 KB/s

Time left: 5h 16min

IP passed: 11 228

Port is open: 28

IP-scan queue size: 0

Path check queue size: 0

Response time stats: Min 0.174916, Average 0.27158, Max 0.502101

Port scan threads: 400

Path check threads: 20

Paths: 4

Response timeout: 1

Apply

Stop filling

Pause

Code 200: 3

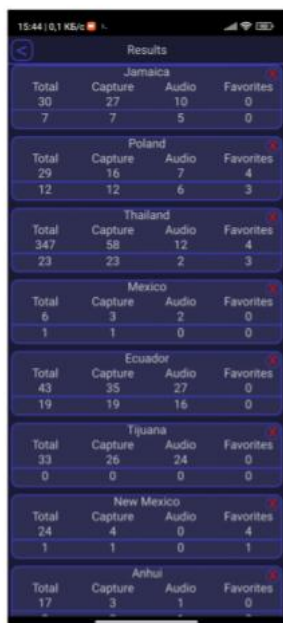
Captured: 3

Завершений процес пошуку

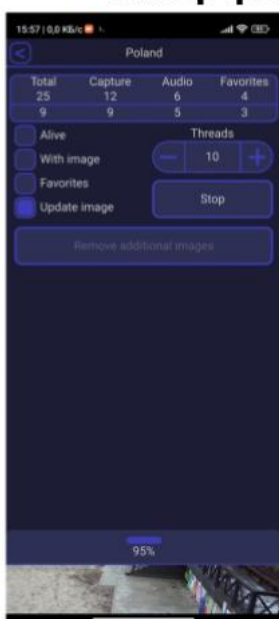
Рисунок Б.14 – Слайд №14

Інтерфейс програми

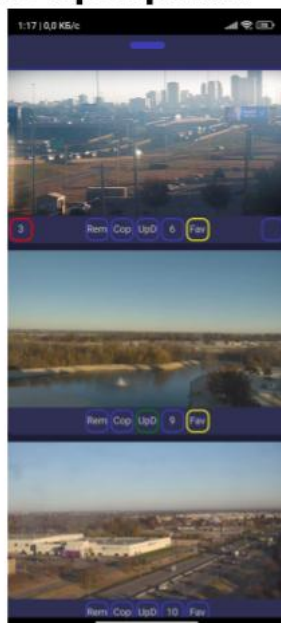
15



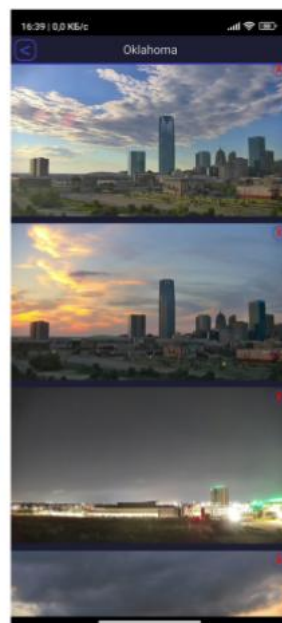
Вікно зі списком результатів сканування



Запущений процес перевірки активності трансляції



Розгорнутий список трансляцій



Вікно перегляду історії знімків

Рисунок Б.15 – Слайд №15

Інтерфейс програми

16

Вікно відтворення
трансляції без
збільшення різкості



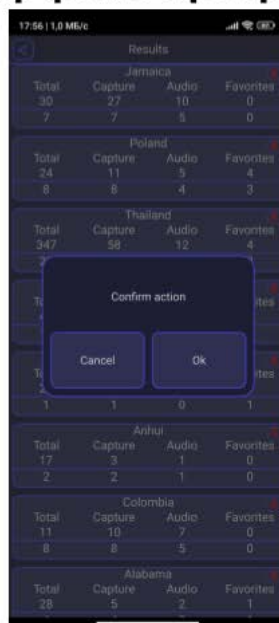
Вікно відтворення
трансляції зі
збільшенням різкості



Рисунок Б.16 – Слайд №16

Інтерфейс програми

17



Діалогове вікно для підтвердження дії видалення результату сканування

Рисунок Б.17 – Слайд №17

Висновки

- Досліджено методи пошуку та візуалізації джерел відеотрансляцій
- Досліджено методи покращення чіткості відеотрансляцій
- Розроблено програмне забезпечення, яке виконує пошук та відображення відкритих трансляцій, що використовують протокол RTSP з можливість збільшення їх чіткості методом згортки різкості

Рисунок Б.18 – Слайд №18