НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних систем і технологій

_____

_____ Швиденко Михайло Зіновійович
Підпис ініціали та прізвище

___ _____202__р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Розробка застосунку для передачі потоку даних в реальному часі

Спеціальність 126 “Інформаційні системи та технології”

Гарант освітньої програми
Канд.екон.наук, доцент _____ Мокрієв Максим Володимирович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи
Доцент, канд. пед. наук _____ Волошина Тетяна Володимирівна
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи
Асистент _____ Понзель Ярослав Юрійович
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Філатов Ростислав Анатолійович
(підпис)

Київ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Завідувач кафедри інформаційних систем і технологій

_____ Швиденко Михайло Зіновійович

Підпис ініціали та прізвище

___ _____202__р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

Філатова Ростислава Анатолійовича

Спеціальності 126 “Інформаційні системи та технології”

1. Тема роботи: “**Розробка застосунку для передачі потоку даних в реальному часі”**
   Затверджена наказом ректора від 16.12.2024 р. № 2245.С
2. Термін подання завершеної роботи на кафедру _____
   (рік, місяць, число)
3. Вихідні дані до проєкту: технічна документація, теоретичні дані.
4. Перелік питань, що розглядаються:
   1. Аналіз предметної області
   2. Огляд технології для розробки системи
   3. Реалізація застосунку
   4. Дослідження та аналіз розробленої системи

№ з/п	**Назва етапів роботи**	**Термін виконання етапів роботи**	**Примітка**
1	Затвердження теми проєкту	01.01.2025 – 10.01.2025	
2	Вивчення та аналіз завдання	11.01.2025 – 20.01.2025	
3	Розробка архітектури та загальної структури системи	21.01.2025 – 31.01.2025	
4	Розробка структур окремих підсистем	01.02.2025 – 15.02.2025	
5	Програмна реалізація системи	16.02.2025 – 29.02.2025	
6	Оформлення пояснювальної записки	26.04.2025 – 01.06.2025	
7	Завершальні перевірки: антиплагіат, предзахист, основний захист	02.06.2025 – 16.06.2025	

Керівник кваліфікаційної роботи _____ / Волошина Т. В., доцент, канд. пед. наук

підпис ПІБ, вчене звання та ступінь

Асистент _____ / Понзель Я. Ю, асистент

підпис ПІБ, вчене звання та ступінь

Завдання прийняв до виконання _____ / Філатов Ростислав Анатолійович

підпис ПІБ

Дата отримання завдання ___.___._____ р.

АНОТАЦІЯ

Цей дипломний проєкт присвячений розробці веб-застосунку для трансляції відео з вебкамери користувача на сервер у режимі реального часу з використанням технологій ASP.NET Core MVC, SignalR та WebRTC. Мета робота полягає в створенні та дослідженні системи, яка забезпечить передачі відеоданих з веб-камери у реальному часі. В програмі реалізовано клієнтську та серверну частини, які передають між собою потік даних, можливість запускати та зупиняти трансляцію. Проєкт демонструє кілька підходів до реалізації веб-застосунків із підтримкою мультимедійного контенту, поєднуючи можливості серверних технологій та інструментів браузера для досягнення високої ефективності в реальному часі.

Annotation

This diploma project is dedicated to the development of a web application for streaming video from the user's webcam to a server in real time using ASP.NET Core MVC, SignalR, and WebRTC technologies. The aim of the work is to create and study a system that enables the real-time transmission of video data from a webcam. The application implements both client-side and server-side components that exchange a data stream and allow starting and stopping the broadcast. The project demonstrates several approaches to implementing web applications that support multimedia content, combining the capabilities of server-side technologies and browser tools to achieve high real-time performance.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ……1

ВСТУП……2

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ……..4

1.1. Сучасні підходи до передачі відео через Інтернет 4

1.1.1 Передача кадрів у вигляді зображень 4

1.1.2 Потокова передача через спеціалізовані протоколи 5

1.1.3 Проблеми, які вирішує WebRTC 5

1.2 Порівняння технологій передачі відео у реальному часі 6

1.2.1 WebSockets 6

1.2.2 SignalR 7

1.2.3 RTMP / HLS 7

1.2.4 WebRTC 8

1.2.5 Обґрунтування вибору WebRTC + SignalR 8

1.3 Принципи роботи WebRTC 8

1.3.1 SDP 9

1.3.2 ICE 9

1.3.3 STUN і TURN сервери 10

1.3.4 Peer-to-Peer модель 10

1.3.5 Обхід NAT 10

1.3.6 Обмін медіа-треками 11

1.4 Огляд бібліотек та API для роботи з WebRTC 11

1.4.1 WebRTC API (getUserMedia, RTCPeerConnection) 12

1.4.2 Обробка відео в браузері 13

1.4.3 Canvas, MediaRecorder 13

1.4.4 SignalR як сигнальний сервер 14

1.5 Приклади застосування подібних систем 15

ВИСНОВОК ДО РОЗДІЛУ 1……...18

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ РОЗРОБКИ……...19
2.1 ASP.NET Core MVC..... 19
2.2 SignalR..... 21
2.3 WebRTC 23
2.4 JavaScript..... 24
2.5 Розробка інтерфейсу користувача..... 27
2.6 IDE Microsoft Visual Studio 28
ВИСНОВОК ДО РОЗДІЛУ 2…….31
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ……..34
3.1 Загальна структура застосунку..... 34
3.2 Реалізація SignalR-потоку 53
3.3 Реалізація WebRTC-потоку..... 54
3.4 Взаємодія сторін..... 55
ВИСНОВОК ДО РОЗДІЛУ 3…….56
РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗР СИСТЕМИ……..58
ВИСНОВОК ДО РОЗДІЛУ 4…….61
ВИСНОВКИ…….62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……...63
ДОДАТОК……...64

ПЕРЕЛІК СКОРОЧЕНЬ

SDP - (Session Description Protocol) Протокол опису сеансу

ICE - (Interactive Connectivity Establishment) Встановлення інтерактивного з'єднання

WebRTC - (Web Real-Time Communication) Веб-комунікація в реальному часі

RTMP - (Real-Time Messaging Protocol) Протокол обміну повідомленнями в реальному часі

HLS – (HTTP Live Streaming) Потокове передавання в реальному часі через HTTP

WebSockets - (Web Sockets Protocol) Протокол веб-сокетів

SignalR - (Signal Real-time) Бібліотека для обміну даними в реальному часі

STUN - (Session Traversal Utilities for NAT) Утиліти для проходження NAT у сеансах

TURN - (Traversal Using Relays around NAT) Проходження NAT за допомогою ретрансляторів

Peer-to-Peer - (P2P) Піринговe з'єднання (взаємодія напряму між пристроями)

NAT - (Network Address Translation) Перетворення мережевих адрес

API - (Application Programming Interface) Інтерфейс прикладного програмування

ВСТУП

Актуальність теми:

У нинішньому світі технології передачі потоку даних в реальному часі таких як - відео даних мають важливе значення у найрізноманітніших сферах діяльності — від відеозв’язку та онлайн-консультацій до систем відеоспостереження, телемедицини та дистанційного навчання. Розвиток інтернет-інфраструктури, збільшення швидкості мережевих з’єднань, а також широке впровадження веб-камер у побуті створили потребу для створення веб-застосунків, здатних обробляти мультимедійні потоки без залучення складного спеціалізованого програмного забезпечення.

Особливої актуальності набувають веб-рішення, що реалізуються в браузерах, без необхідності встановлення сторонніх додатків. Завдяки сучасним веб-API, а також таким технологіям, як WebRTC і SignalR, стало можливим створювати живу трансляцію відео прямо між клієнтами, або ж — при використанні посередника у вигляді сервера — будувати централізовані системи відео передачі. Це відкриває широкі можливості для застосування у багатьох сферах, зокрема в тих, де важливими є швидкість передачі, надійність зв’язку та простота використання.

Таким чином, створення веб-системи, яка дозволяє користувачу транслювати відео з власної вебкамери на сервер у реальному часі, є актуальним завданням, що поєднує в собі аспекти мережевого програмування, розробки фронтенду та бекенду, обробки мультимедіа та використання хмарних технологій.

Мета та завдання дослідження:

Метою дипломного проєкту є розробка ефективного веб-застосунку, який забезпечить передачу відео з вебкамери користувача до серверної частини програми, а також дозволить переглядати це відео іншим користувачам.

Для досягнення цієї мети необхідно вирішити декілька завдань:

- Провести аналіз сучасних технологій, що дозволяють реалізувати відео трансляцію в реальному часі;
- Визначити найбільш оптимальний підхід до передачі відео для браузерних клієнтів;
- Розробити серверну частину на основі ASP.NET Core MVC з підтримкою SignalR;
- Забезпечити інтерфейс керування трансляцією;
- Провести тестування системи на різних пристроях і браузерах;
- Здійснити аналіз ефективності та стабільності роботи системи.

Структура роботи:

Дипломна робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел та додатків. У першому розділі виконується аналіз предметної області, розглядаються існуючі методи передачі відео та технології, які можуть бути використані для реалізації задачі. Другий розділ присвячено аналізу інструментів та технологій розробки, зокрема фреймворків ASP.NET Core MVC, бібліотеки SignalR, технології WebRTC та JavaScript API. У третьому розділі описується процес реалізації програмної системи, її структура, основні компоненти, інтерфейс користувача та логіка взаємодії між клієнтом і сервером.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Сучасні підходи до передачі відео через Інтернет

Передача відео через мережу Інтернет є однією з найбільш затребуваних та тих, що динамічно розвиваються галузей інформаційних технологій. Відео трансляції використовуються у відеозв'язку, стрімінгових платформах, дистанційному навчанні та інших сферах. Ключовим технічним завданням у цих системах є забезпечення стійкого, безперервного, якісного відеопотоку з мінімальною затримкою, що надходить від одного користувача до іншого або до серверної інфраструктури.

1.1.1 Передача кадрів у вигляді зображень

Одним із найпростіших методів передачі відео заключається в періодичному захопленню зображень з відеопотоку у вигляді окремих кадрів, які перетворюються у формат base64 і надсилаються через HTTP, WebSocket або SignalR. Такий метод зазвичай використовується в системах, де реальна швидкість відео не є критичною, наприклад у системах відеоспостереження, де важливіші стабільність і точність фіксації подій.

Переваги цього підходу:

- Простота реалізації;
- Висока сумісність із браузерами;

Недоліки:

- Обмеження частоти кадрів (FPS), що залежить від потужності клієнта та мережі;
- Неможливість передачі звуку.

1.1.2 Потокова передача через спеціалізовані протоколи

Традиційні стрімінгові сервіси, такі як Youtube або Twitch, використовують протоколи RTMP, HLS, DASH та інші. Ці протоколи забезпечують розділення відео на сегменти, кодування у певному форматі (наприклад, H.264), буферизацію та доставку до глядача з адаптацією до швидкості з'єднання.

Хоч ці технології і широко використовуються для масових трансляцій, їх реалізація у звичайному веб-додатку є досить складним: потрібно налаштування окремого медіасервера, кодування відео у реальному часі, генерація плейлістів і тому подібне.

1.1.3 Проблеми, які вирішує WebRTC

WebRTC — це сучасна технологія, яка дозволяє організовувати обмін аудіо, відео та іншими потоковими даними напряму між браузерами без участі сервера під час передачі даних. Це технологія призначена для забезпечення максимальної швидкості та низької затримки при спілкуванні або відеотрансляції.

WebRTC використовує внутрішні API браузера (getUserMedia, RTCPeerConnection, MediaStream) та потребує так званої сигнальної взаємодії (signaling), щоб домовитися про з'єднання між сторонами.

Переваги WebRTC:

- Живе peer-to-peer відео з мінімальною затримкою;
- Підтримка аудіо та відео одночасно;
- Стандартна підтримка в основних браузерах.

Недоліки:

- Складність ініціалізації з'єднання;
- Потреба у STUN/TURN-серверах для роботи через NAT;
- Неможливість контролювати чи змінювати потік на сервері (якщо не використовуються сервери ретрансляції).

1.2. Порівняння технологій передачі відео у реальному часі

Для створення системи відеотрансляції у веб-середовищі ключовим аспектом є вибір найбільш ефективної технології передачі мультимедійних даних з клієнта на сервер або інший клієнт у режимі реального часу. Існує декілька популярних підходів до вирішення цієї задачі, кожен з яких має власні переваги, обмеження та особливості реалізації.

У цьому підрозділі розглядаються основні технології, які можуть бути застосовані у контексті веб-застосунку, зокрема WebSockets, SignalR, RTMP/HLS/DASH та WebRTC, а також наведено аргументацію щодо доцільності використання WebRTC у поєднанні з SignalR у рамках даної дипломної роботи.

1.2.1 WebSockets

WebSocket — це двонаправлений протокол передачі даних між браузером і сервером, який дозволяє підтримувати постійне з’єднання та обмінюватися повідомленнями в обох напрямках без необхідності повторного встановлення HTTP-запитів. Цей протокол широко використовується для чатів, біржевих додатків, онлайн-ігор тощо.

У контексті передачі відео WebSocket може бути використаний для передачі зображень або оброблених фрагментів відео, однак його ефективність значно поступається спеціалізованим мультимедійним технологіям. WebSocket не оптимізований для потокової передачі великих мультимедіа-даних, таких як аудіо чи відео, тому використовується здебільшого для керуючої інформації або сигналізації.

1.2.2 SignalR

SignalR — це бібліотека для ASP.NET Core, яка реалізує високорівневу абстракцію над WebSocket, Server-Sent Events і Long Polling. Вона дозволяє дуже просто організувати двостороннє спілкування між браузером і сервером в реальному часі, автоматично вибираючи оптимальний транспорт залежно від можливостей браузера та сервера.

SignalR активно використовується у веб-застосунках, що потребують миттєвого оновлення даних — наприклад, у чатах, інформаційних панелях, колективному редагуванні документів. У рамках даної системи SignalR застосовується у двох напрямах:

- Передача відео кадрів — клієнт захоплює зображення з відеопотоку через canvas, конвертує у формат base64 та надсилає через SignalR-хаб на сервер. Це дозволяє реалізувати базовий режим трансляції без складної обробки.
- Сигналізація для WebRTC — SignalR використовується як канал для обміну службовими повідомленнями між двома клієнтами для ініціалізації WebRTC-з’єднання (SDP, ICE).

Таким чином, SignalR є важливою складовою обох режимів роботи системи.

1.2.3 RTMP / HLS

RTMP (Real-Time Messaging Protocol), HLS (HTTP Live Streaming) та MPEG-DASH — це мультимедійні протоколи, які використовуються великими відеосервісами для трансляції відео великій кількості глядачів. У таких системах використовується складна інфраструктура:

- Спеціальні медіасервери (наприклад, NGINX RTMP або Wowza);
- Серйозне попереднє кодування та буферизація;
- Генерація плейлістів і сегментів відео.

Хоча ці протоколи забезпечують високу якість та масштабованість, вони не є оптимальними для одноразових peer-to-peer трансляцій у браузері, які вимагають простоти реалізації та низької затримки.

1.2.4 WebRTC

WebRTC — це сучасний відкритий стандарт для браузерів, який дозволяє організовувати прямий обмін відео, аудіо та іншими даними між клієнтами без залучення серверів для ретрансляції медіапотоків. Ця технологія є ідеальним вибором для задач, що вимагають низької затримки, високої якості відео та звуку, а також простоти інтеграції в браузерне середовище.

WebRTC дозволяє:

- Захоплювати відео/аудіо з вебкамери (getUserMedia);
- Створювати P2P-з’єднання (RTCPeerConnection);
- Передавати медіапотік без буферизації;
- Використовувати протоколи NAT traversal (ICE, STUN, TURN).

1.2.5 Обґрунтування вибору WebRTC + SignalR

Комбінація WebRTC та SignalR є оптимальним вибором для створення сучасної веб-системи відеотрансляції у реальному часі з точки зору:

- Простоти: SignalR дозволяє швидко організувати комунікацію між клієнтами.
- Якісного відео: WebRTC забезпечує передачу потокового відео без буферизації.
- Гнучкості: обидві технології підтримуються в усіх сучасних браузерах.

1.3. Принципи роботи WebRTC

Технологія WebRTC (Web Real-Time Communication) дозволяє браузерам встановлювати прямий (peer-to-peer) зв’язок між клієнтами для обміну медіаданими — відео, аудіо та іншими потоками — без необхідності в проміжному сервері для передачі самих даних. Проте, попри зовнішню

простоту використання, всередині WebRTC функціонує складна система узгодження параметрів, налаштування маршрутизації трафіку та забезпечення сумісності між пристроями.

1.3.1 SDP

SDP (Session Description Protocol) — це формат, який використовується WebRTC для опису параметрів з'єднання між двома сторонами: кодеків, типів медіа, портів, IP-адрес, протоколів та іншої службової інформації. Його мета — дозволити клієнтам домовитися про те, як саме вони будуть обмінюватися медіапотоками.

Коли один з учасників ініціює з’єднання, він створює SDP-повідомлення типу "offer", що містить усі необхідні параметри. Інший учасник, приймаючи запрошення, відповідає "answer" з власними параметрами, що узгоджуються з параметрами ініціатора.

SDP — це основа сигнального процесу в WebRTC, але сама передача SDP відбувається за допомогою зовнішнього каналу сигналізації.

1.3.2 ICE

ICE (Interactive Connectivity Establishment) — це технологія, яка допомагає двом WebRTC-клієнтам знайти оптимальний маршрут для встановлення з’єднання. Вона виконує роль "маршрутизатора", надаючи список потенційних мережевих шляхів (IP-адрес, портів), які можуть бути використані для прямого з’єднання. Процес ICE включає:

- Створення переліку локальних адрес (host candidates);
- Запит адрес у STUN-серверів (server-reflexive candidates);
- Підключення до TURN-сервера за потреби (relayed candidates);
- Випробування всіх кандидатів і вибір найкращого маршруту для з’єднання.

ICE дозволяє враховувати складні умови: динамічні IP-адреси, NAT, firewall, VPN та інші мережеві бар’єри.

1.3.3 STUN і TURN сервери

Для успішної роботи ICE WebRTC покладається на допоміжні сервери — STUN і TURN, які дозволяють встановити з'єднання у складних мережевих умовах.

STUN (Session Traversal Utilities for NAT) — це сервер, який допомагає клієнту дізнатися, яка зовнішня (публічна) IP-адреса призначена йому NAT'ом. Після запиту клієнт отримує свою публічну адресу, яку потім використовує в SDP.

TURN (Traversal Using Relay NAT) — це ретрансляційний сервер, який використовується, якщо пряме з'єднання неможливе. У цьому випадку відео передається через TURN, як посередника. Це повільніше, дорожче та менш ефективно, тому TURN застосовується лише як резерв.

1.3.4 Peer-to-Peer модель

Модель WebRTC заснована на P2P-з'єднанні — тобто браузери встановлюють прямий зв'язок одне з одним, без участі сервера в обробці мультимедіа. Це означає:

- Мінімальна затримка (latency);
- Висока пропускна здатність;
- Відсутність навантаження на сервер;
- Немає потреби зберігати потік на сервері (якщо не потрібно).

1.3.5 Обхід NAT

Більшість користувачів підключені до Інтернету через NAT (Network Address Translation) — технологію, яка приховує реальні IP-адреси внутрішньої мережі. Це ускладнює встановлення P2P-з'єднань, бо клієнти не знають, як достукатися одне до одного.

WebRTC вирішує цю проблему за допомогою:

- STUN — щоб дізнатися свою публічну адресу;
- ICE — щоб зібрати кандидати з різних джерел;
- TURN — для обхідних сценаріїв.

1.3.6 Обмін медіа-треками

Після встановлення з'єднання між двома RTCPeerConnection об'єктами, WebRTC автоматично починає передавати медіа-треки — відео або аудіо потоки. Для цього використовується:

- API getUserMedia — для захоплення відео з вебкамери;
- Метод addTrack — для додавання треків у peer-з'єднання;
- Подія ontrack — для прийому треків на іншій стороні.

Лістинг коду клієнта: *Лістинг 1.1*

```
const stream = await navigator.mediaDevices.getUserMedia({ video: true });
stream.getTracks().forEach(track => peerConnection.addTrack(track,stream));
```

Лістинг коду приймача:

```
peerConnection.ontrack = event => {
   remoteVideo.srcObject = event.streams[0];
};
```

Медіапотік відображається в <video>-елементі й може бути зупинений або перезапущений в будь-який момент через JavaScript.

1.4. Огляд бібліотек та API для роботи з WebRTC

Сучасні браузери оснащені вбудованими інтерфейсами програмування, які дозволяють розробникам реалізовувати функціонал для захоплення

медіа, створення з’єднань у реальному часі, відображення відео, а також обробки потоку на клієнтській стороні. При виконанні дипломної роботи основна увага приділяється таким ключовим API, як WebRTC API, HTML5 Canvas, інструменту MediaRecorder та бібліотеці SignalR, яка використовується для організації сигнального обміну між клієнтами.

1.4.1 WebRTC API (getUserMedia, RTCPeerConnection)

WebRTC складається з декількох базових інтерфейсів, вбудованих у браузер, які забезпечують доступ до медіа та встановлення з’єднання.

getUserMedia() — це API, яке дозволяє отримати доступ до відео- та аудіопристроїв користувача (наприклад, вебкамери чи мікрофона). Це перший етап у WebRTC-обміні: за його допомогою отримується локальний потік, який потім передається у з’єднання.

```
navigator.mediaDevices.getUserMedia({ video: true, audio: false })
  .then(stream => {
    localVideo.srcObject = stream;
  });
```

Метод повертає MediaStream, який може бути доданий до RTCPeerConnection.

RTCPeerConnection — головний об’єкт у WebRTC, який відповідає за:

- налаштування з’єднання;
- обмін SDP-повідомленнями;
- обмін ICE-кандидатами;
- передачу медіатреками;
- взаємодію зі STUN/TURN-серверами.

```
const pc = new RTCPeerConnection(configuration);
stream.getTracks().forEach(track => pc.addTrack(track, stream));
```

Також RTCPeerConnection має обробник подій ontrack, який дозволяє приймати відео:

```
pc.ontrack = event => {
   remoteVideo.srcObject = event.streams[0];
};
```

Таким чином, ці два API (getUserMedia + RTCPeerConnection) є основою для створення функціонального відеозв'язку між клієнтами у браузері.

1.4.2 Обробка відео в браузері

Отриманий відеопотік від getUserMedia() — це об'єкт типу MediaStream, який містить набір треків (аудіо та/або відео). У браузері з цим потоком можна виконувати наступні дії:

- Відображення у <video> елементі;
- Захоплення кадрів через <canvas>;
- Запис через MediaRecorder;
- Передача через WebRTC;
- Аналіз (наприклад, через нейромережі у TensorFlow.js).

Таким чином, браузер не лише приймає відео, але й має можливість його обробляти, трансформувати або зберігати.

Для створеної системи основні варіанти обробки є:

- у SignalR-режимі: кадри з відео зчитуються у canvas, кодуються в base64 і надсилаються;
- у WebRTC-режимі: відео потік передається напряму між клієнтами, без обробки.

1.4.3 Canvas, MediaRecorder

Canvas — HTML5-елемент, який дозволяє програмно працювати з пікселями зображень або відео. У застосунку він використовується в режимі SignalR для витягування окремих кадрів із відео та перетворення їх у зображення (base64).

Лістинг коду 1.2

```
const canvas = document.createElement("canvas");
canvas.width = 480;
const ctx = canvas.getContext("2d");
ctx.drawImage(video, 0, 0, canvas.width, canvas.height);
const image = canvas.toDataURL("image/webp");
```

Цей підхід дозволяє реалізувати стрім через передачу зображень (кадрів), що є простим способом організації псевдо-відеопотоку.

MediaRecorder — ще один API, який дозволяє записувати відео з MediaStream у форматі .webm, .mp4 тощо. Це особливо корисно для функції запису трансляції.

```
const recorder = new MediaRecorder(stream);
recorder.ondataavailable = e => chunks.push(e.data);
recorder.start();
```

1.4.4 SignalR як сигнальний сервер

Оскільки WebRTC не реалізує обмін службовою інформацією самостійно, потрібен окремий канал для передачі:

- SDP-повідомлень ("offer" та "answer");
- ICE-кандидатів.

У системі для цього використовується SignalR — бібліотека ASP.NET Core, яка реалізує двосторонню передачу повідомлень між клієнтом і сервером через WebSocket або інші механізми.

На сервері визначено клас StreamHub, який реалізує методи:

- public async Task SendOffer(object offer)
- public async Task SendAnswer(object answer)
- public async Task SendCandidate(object candidate)

Ці методи викликаються з JavaScript на клієнті, і пересилають відповідну інформацію іншим клієнтам. Завдяки цьому WebRTC-з'єднання може бути ініціалізовано, навіть якщо клієнти не знають про існування одне одного наперед.

SignalR також дозволяє реалізувати обробку декількох потоків, підтримку кімнат (груп), повідомлень тощо.

1.5. Приклади застосування подібних систем

Технології потокової передачі відео в реальному часі, зокрема WebRTC, широко застосовуються в різних сферах діяльності. У зв'язку з глобальним поширенням дистанційної роботи, онлайн-освіти, та відеоспостереження, з'являється все більше прикладів інтеграції відеозв'язку безпосередньо у веббраузери без потреби встановлення додаткового програмного забезпечення. Нижче наведено найбільш поширені сценарії застосування WebRTC-подібних систем.

Сучасні сервіси миттєвих повідомлень, такі як Google Meet, Discord, Microsoft Teams та Zoom, активно використовують WebRTC як базову технологію для організації відеодзвінків у браузері.

Особливості таких систем:

- Використання WebRTC для захоплення відео/аудіо;
- Наявність сигнального серверу (власна реалізація або WebSocket/SignalR);
- Можливість обміну в реальному часі не лише відео, але й текстовими повідомленнями, файлами, екранами;
- Захист даних за допомогою шифрування (DTLS, SRTP).

Ці системи демонструють потенціал WebRTC як масштабованої та надійної технології для персонального та корпоративного відеозв'язку.

У сфері охорони здоров’я WebRTC став основою для телемедичних платформ, які дозволяють лікарям консультувати пацієнтів дистанційно. Наприклад:

- Doxy.me, VSee, MDLIVE — сервіси, що забезпечують відеозв’язок лікар-пацієнт без встановлення клієнтів;
- Підтримка запису сесій, обміну даними пацієнта, ведення чату;
- Висока безпека передачі (HIPAA-сумісність у США);
- Робота в браузері на будь-якому пристрої.

У цьому контексті WebRTC дозволяє створити легкий у використанні, безпечний і якісний канал зв’язку, що критично важливо у галузі медицини.

Веброзробники все частіше інтегрують WebRTC у власні застосунки для:

- внутрішнього моніторингу персоналу (екран + вебкамера),
- зв’язку техпідтримки з клієнтами на сайті,
- кастомних стрімінгових рішень (наприклад, із мобільної камери на сервер).

Цей дипломний проєкт реалізує саме таке рішення — спеціалізовану систему відеотрансляції, яку можна використовувати у локальній мережі або через інтернет для зв’язку між клієнтом та сервером.

З урахуванням наведених прикладів, можна стверджувати, що WebRTC є стратегічно важливою технологією для майбутнього відеозв’язку в Інтернеті. Її підтримка браузерами, низька затримка та простота використання роблять її ідеальною для широкого спектра застосунків — від соціальних до промислових.

Створення власного вебзастосунку з використанням WebRTC дозволяє глибше зрозуміти принципи роботи відеопотоків, мережевого узгодження та взаємодії між пристроями у реальному часі, що підтверджує актуальність і цінність даного дипломного проєкту.

Підхід	Затримка	Якість	Складність реалізації	Передача звуку
SignalR	Висока	Низька	Низька	-
RTMP/HLS	Середня	Висока	Висока	+
WebRTC	Низька	Висока	Середня	+

Таблиця 1.1 – порівняльна таблиця SignalR, RTMP/HLS, WebRTC.

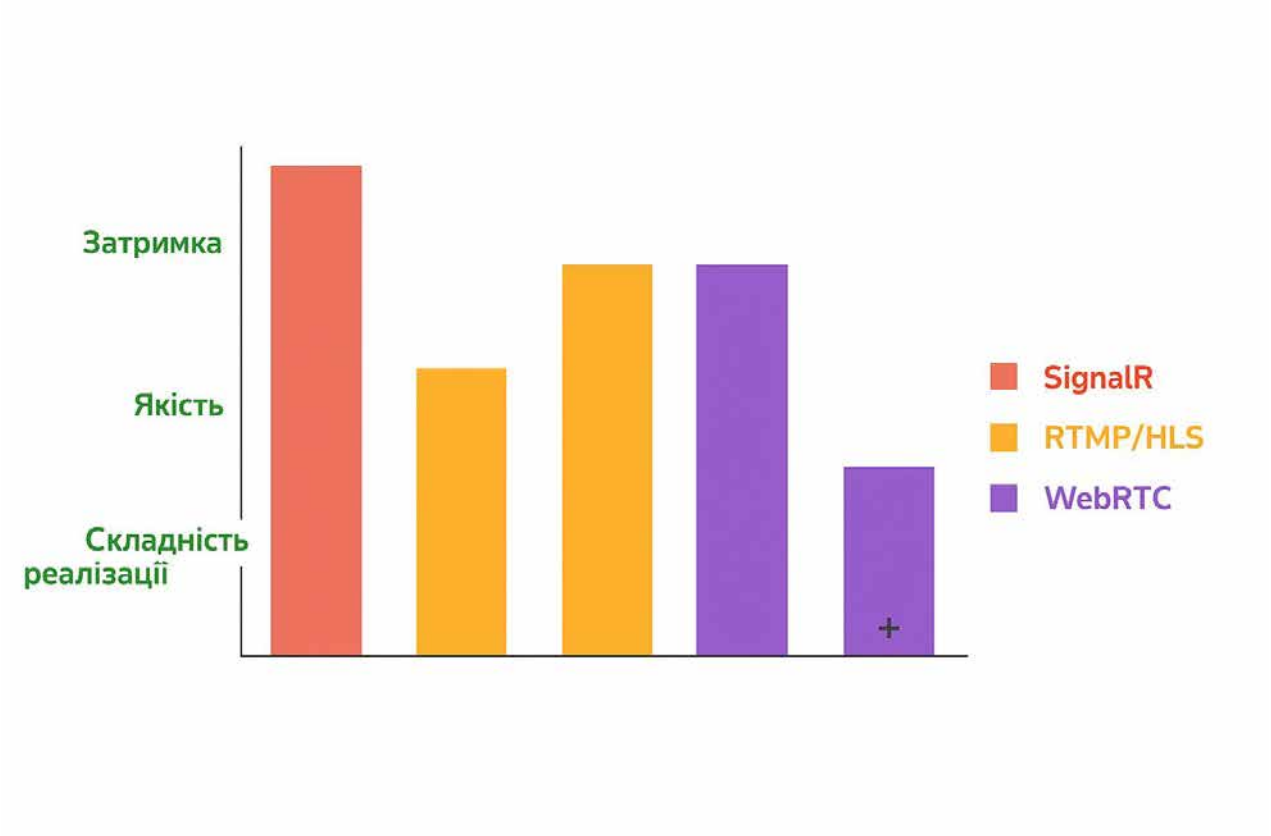

Рисунок 1.1 – діаграма до порівняльної таблиці.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було проведено глибокий аналіз предметної області, пов’язаної з передачею відео у веб-середовищі в режимі реального часу. Встановлено, що з розвитком інтернет-інфраструктури, широким впровадженням браузерних технологій і зростанням потреби в онлайн-комунікації, передача аудіо- та відеопотоків напряму між користувачами стала не лише можливою, а й критично важливою для багатьох прикладних задач — від телемедицини до онлайн-освіти.

Також описано основні підходи до передачі відео, серед яких особливу увагу було приділено WebRTC — як сучасній технології прямого обміну медіаданими між клієнтами.

Порівняльний аналіз довів, що WebRTC є оптимальним вибором для розробки легкого та ефективного вебзастосунку, а його поєднання із SignalR дозволяє вирішити проблему сигналізації без потреби у складній серверній інфраструктурі.

Було детально описано механізми роботи WebRTC, зокрема протокол SDP, процес ICE, використання STUN/TURN-серверів, особливості обходу NAT та безпосередній обмін медіатреками. Це дозволило сформувати чітке уявлення про те, як відбувається встановлення та підтримка WebRTC-з’єднання між клієнтами у браузерах.

Розглянуто відповідні бібліотеки і браузерні API у які показують, що сучасні вебтехнології надають всі необхідні інструменти для реалізації WebRTC-застосунку: від захоплення відео до створення P2P-з’єднання та передачі службових повідомлень. Згаданий набір інструментів є кросплатформним, безкоштовним, не потребує стороннього ПЗ та може бути реалізований у межах браузера.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ РОЗРОБКИ

2.1. ASP.NET Core MVC

ASP.NET Core MVC — це високопродуктивний, кросплатформний фреймворк, створений компанією Microsoft для розробки динамічних вебзастосунків. Його архітектура базується на шаблоні MVC (Model–View–Controller), який забезпечує чітке розділення логіки обробки даних, представлення та керування діями користувача.

У контексті розробленої системи, ASP.NET Core MVC використовується як основа серверної частини, яка:

- обробляє HTTP-запити клієнтів;
- маршрутизує користувачів до різних режимів роботи (SignalR або WebRTC);
- організовує обмін інформацією через SignalR Hub;
- надає HTML-подання (Views), у які вбудовані клієнтські скрипти.

Архітектура MVC в створеній програмі виглядає наступним чином:

- Модель (Model) — містить опис параметрів переданих даних та DTO-структури (наприклад, об’єкти для передачі SDP чи кандидатів).
- Контролер (Controller) — в основному це HomeController, який обробляє маршрути:
  /Home/SignalRStream — сторінка передачі кадрів через SignalR;
  /Home/WebRTC — клієнт WebRTC;
  /Home/WebRTCServer — перегляд WebRTC потоку на сервері;
  /Home/Index — головна сторінка з навігацією.
- Подання (View) — HTML-сторінки, згенеровані Razor-синтаксисом, у які інтегровані JavaScript-файли. Подання повністю

адаптивні, підтримують підключення JavaScript-бібліотек для роботи з WebRTC та SignalR.

Це забезпечує зручність розширення системи, підтримку декількох сценаріїв роботи, а також високу швидкодію завдяки асинхронній обробці запитів.

ASP.NET Core підтримує високопродуктивні сценарії у реальному часі завдяки:

- вбудованій підтримці SignalR;
- оптимізованій обробці WebSocket-з'єднань;
- асинхронній моделі обробки подій;
- масштабованості через Kestrel, IIS або хмарні сервіси (Azure, AWS).

Також ASP.NET Core дозволяє легко розгортати застосунок як на Windows, так і на Linux, що робить його зручним для кросплатформного використання.

Вибір ASP.NET Core MVC як основи серверної частини системи був зумовлений такими перевагами:

- Надійність та стабільність — технологія розроблена і підтримується Microsoft;
- Гнучкість — легка інтеграція сторонніх бібліотек;
- Продуктивність — висока швидкодія при великій кількості підключень;
- Кросплатформність — можливість розгортання на будь-якій ОС;
- Інтеграція з сучасними IDE — зокрема Visual Studio;
- Розширюваність — зручне створення нових маршрутів, подань і API.

2.2. SignalR

SignalR — це бібліотека з відкритим вихідним кодом, розроблена Microsoft для ASP.NET Core, яка надає можливість організовувати двосторонній зв’язок у реальному часі між клієнтом і сервером. Вона абстрагує розробника від деталей транспортного протоколу, автоматично вибираючи найкращий доступний спосіб зв’язку: WebSockets, Server-Sent Events або Long Polling.

SignalR ідеально підходить для застосунків, де важливо оновлювати інтерфейс користувача без повторного надсилання HTTP-запитів, зокрема: онлайн-чати, біржові платформи, мультиплеєрні ігри, спільне редагування документів тощо.

У цьому дипломному проєкті SignalR виконує дві ключові ролі:

- Як транспортний канал для передачі кадрів зображення з вебкамери (у SignalR-режимі);
- Як сигнальний сервер для встановлення WebRTC-з’єднання між двома браузерами (у WebRTC-режимі).

SignalR використовує модель зв’язку "клієнт-сервер" через Hub — серверний клас, який обробляє вхідні повідомлення від клієнтів і надсилає відповіді. На клієнтській стороні застосовується JavaScript-бібліотека, яка дозволяє викликати методи на сервері та приймати виклики з нього у режимі реального часу.

Основні етапи роботи:

- Клієнт підключається до SignalR-хабу.
- Встановлюється постійне з'єднання (через WebSocket або інший механізм).
- Клієнт викликає методи на сервері (наприклад, SendFrame, SendOffer).

- Сервер, у свою чергу, може викликати методи на клієнті (ReceiveFrame, ReceiveAnswer).

У проєкті реалізовано SignalR-хаб з назвою StreamHub, який виконує дві функції залежно від режиму роботи системи:

1. Режим "SignalR Stream" (трансляція кадрів)

У цьому режимі клієнт:

- захоплює відео з вебкамери;
- малює кадри на <canvas>;
- перетворює зображення у base64;
- надсилає їх до SignalR-хабу.
- Сервер викликає метод ReceiveFrame у підключеного глядача, передаючи base64-зображення. В результаті зображення відображається як відео у режимі низького FPS.

Такий підхід дозволяє реалізувати базовий режим відеотрансляції, який працює навіть у браузерах без підтримки WebRTC.

2. Режим "WebRTC Stream" (живий потік)

У цьому випадку SignalR використовується не для передачі медіа, а як канал сигналізації, який передає:

- SDP-повідомлення (SendOffer, SendAnswer);
- ICE-кандидати (SendCandidate).

Ці повідомлення надходять до потрібного клієнта, який, отримавши дані, ініціалізує WebRTC-з’єднання. Сам потік відео передається напряму між браузерами, без участі сервера в обробці трафіку.

Використання SignalR у системі трансляції забезпечує:

- Швидку реалізацію без складної конфігурації;

- Автоматичну адаптацію транспорту (WebSockets / SSE / Long Polling);
- Підтримку множинних клієнтів та широкомовної передачі;
- Зниження навантаження на сервер у WebRTC-режимі;
- Універсальність — одна технологія виконує роль каналу даних і сигналізації.

2.3. WebRTC

WebRTC (Web Real-Time Communication) — це набір стандартів і API, який дозволяє здійснювати аудіо-, відео- та обмін іншими потоковими даними між браузерами у режимі реального часу без необхідності в сервері для ретрансляції мультимедіа. Ця технологія підтримується сучасними браузерами (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) і працює кросплатформно — тобто без встановлення додаткових плагінів або програм.

WebRTC дає змогу:

- організовувати відеозв’язок між користувачами безпосередньо;
- обходити NAT і firewall завдяки STUN/TURN-серверам;
- працювати зі стандартним JavaScript API (getUserMedia, RTCPeerConnection);
- забезпечувати низьку затримку (latency), критичну для живої трансляції;
- передавати потоки без участі серверної частини у самій передачі — лише у фазі "узгодження".

У розробленій системі, WebRTC використовується як основна технологія для реалізації режиму живої трансляції відео між клієнтом (джерелом відео) та серверною частиною (відкритим вікном перегляду трансляції). Основна задача якого, забезпечити передачу потоку з

мінімальними втратами та затримками, без буферизації, використовуючи лише ресурси браузера.

Порядок дій у WebRTC-режимі такий:

- Клієнт отримує доступ до камери через getUserMedia();
- Потік додається у об'єкт RTCPeerConnection;
- Клієнт формує SDP-повідомлення типу offer і надсилає його іншому клієнту через SignalR;
- Одержувач відповідає answer;
- Обидва клієнти обмінюються ICE-кандидатами, визначаючи найкращий маршрут з’єднання;
- Після встановлення з’єднання медіапотік автоматично надходить у браузер одержувача.

Інтерфейс користувача (User Interface, UI) у вебзастосунку відіграє ключову роль, забезпечуючи інтуїтивну та зручну взаємодію з функціональністю системи. Основна мета при створенні інтерфейсу — забезпечити мінімалістичне, зрозуміле, доступне середовище, де користувач зможе легко:

- обрати режим трансляції (SignalR або WebRTC),
- запустити або зупинити вебкамеру,
- переглянути передане відео.

Для цього використано сучасні технології: HTML5, CSS3, JavaScript, а також Razor-шаблони, які генеруються на стороні сервера в ASP.NET Core MVC.

2.4. JavaScript

JavaScript — основна клієнтська мова програмування в розробці вебінтерфейсів. У зробленій системі вона відіграє ключову роль у забезпеченні динамічної взаємодії з камерою, відображенні відео,

встановленні WebRTC-з’єднання, передачі кадрів через SignalR та обробці подій інтерфейсу користувача.

Усі інтерактивні дії — запуск або зупинка камери, встановлення peer-to-peer з’єднання, обмін сигналами, оновлення елементів на сторінці — реалізовані саме за допомогою JavaScript.

Перший крок у трансляції відео — це доступ до вебкамери користувача. Для цього використовується Web API getUserMedia():

Лістинг коду 2.1

```
navigator.mediaDevices.getUserMedia({ video: true })
  .then(stream => {
    localVideo.srcObject = stream;
  })
  .catch(error => console.error('Помилка доступу до камери:', error));
```

Цей метод повертає MediaStream, який потім може бути:

- доданий у WebRTC-з’єднання,
- відображений у <video>-елементі,
- використаний для створення знімків через <canvas>.

Для створення з’єднання у WebRTC використовується клас RTCPeerConnection. Він дозволяє:

- додати медіатрек у з’єднання;
- створити та обмінятися SDP-повідомленнями;
- обмінятися ICE-кандидатами;
- встановити безпосередній зв’язок із віддаленим клієнтом.

```
const pc = new RTCPeerConnection({
```

```
    iceServers: [{ urls: "stun:stun.l.google.com:19302" }]
});
stream.getTracks().forEach(track => pc.addTrack(track, stream));
```

Коли потік отримано на іншому боці з’єднання, він обробляється подією:

```
pc.ontrack = event => {
    remoteVideo.srcObject = event.streams[0];
};
```

У режимі передачі зображень SignalR (імітація відео), JavaScript використовує елемент <canvas> для зчитування поточних кадрів із камери:

Лістинг коду 2.2

```
const canvas = document.createElement("canvas");
const ctx = canvas.getContext("2d");

function captureFrame() {
    ctx.drawImage(video, 0, 0, canvas.width, canvas.height);
    const base64 = canvas.toDataURL("image/webp");
    connection.invoke("SendFrame", base64);
}
```

Цей кадр у форматі base64 надсилається через SignalR до іншого клієнта, де вставляється в <img>:

```
connection.on("ReceiveFrame", base64 => {
    imageElement.src = base64;
});
```

Окрім роботи з медіа, JavaScript відповідає за:

- активацію та деактивацію кнопок;
- зміни стилів елементів;
- відображення повідомлень про статус з’єднання;
- запуск таймерів та циклів оновлення.

Наприклад, кнопка запуску трансляції:

```
startBtn.onclick = () => {
   startStreaming();
   startBtn.disabled = true;
   stopBtn.disabled = false;
};
```

2.5. Розробка інтерфейсу користувача

Інтерфейс користувача (User Interface, UI) у вебзастосунку відіграє ключову роль, забезпечуючи інтуїтивну та зручну взаємодію з функціональністю системи. Основна мета при створенні інтерфейсу — забезпечити мінімалістичне, зрозуміле, доступне середовище, де користувач зможе легко:

- обрати режим трансляції (SignalR або WebRTC),
- запустити або зупинити вебкамеру,
- переглянути передане відео.

Для цього використано сучасні технології: HTML5, CSS3, JavaScript, а також Razor-шаблони, які генеруються на стороні сервера в ASP.NET Core MVC. Інтерфейс складається з таких елементів:

Головна сторінка (/Home/Index), яка містить кнопки-посилання на обидва режими. Це реалізовано через Razor-синтаксис:

<a asp-action="SignalRStream" class="btn btn-primary">Перейти до SignalR

<a asp-action="WebRTC" class="btn btn-success">Перейти до WebRTC

Цей підхід дозволяє динамічно переходити між сторінками без складної маршрутизації.

Інтерфейс має динамічну поведінку:

- При натисканні кнопки запускається JavaScript-логіка (startStream, connectToPeer).
- Змінюється стан елементів (disabled, hidden).
- Додаються або прибираються стилі (.active, .error, .streaming).
- У реальному часі оновлюється вміст (наприклад, зображення кадрів).

Це робить інтерфейс не лише статичним, а реактивним — таким, що змінюється у відповідь на дії користувача або події системи.

2.6. IDE Microsoft Visual Studio

Microsoft Visual Studio — це потужне інтегроване середовище розробки (IDE), яке широко використовується для створення .NET-застосунків, у тому числі вебзастосунків на основі ASP.NET Core MVC. IDE надає розробнику всі необхідні інструменти: редактор коду, засоби відлагодження, менеджер пакетів, підтримку шаблонів проєктів, емуляторів браузера, системи контролю версій, а також інтеграцію з GitHub і хмарними сервісами Azure.

У межах розробки дипломного проєкту Visual Studio відіграло центральну роль як основне середовище для створення серверної частини застосунку, керування залежностями, відлагодження SignalR-хабу, підключення JavaScript та підготовки Razor-шаблонів.

Налаштування проєкту

Створення застосунку:

- При створенні нового проєкту було обрано шаблон ASP.NET Core Web App (Model-View-Controller).
- Була встановлена конфігурація з підтримкою .NET 6 / .NET 7 (залежно від конкретної версії IDE).
- Було додано підтримку SignalR через NuGet (як частину Microsoft.AspNetCore.SignalR).
- Створено структуру папок: Controllers, Views, Hubs, wwwroot, Models.

Visual Studio забезпечує:

- Інтелектуальне автодоповнення (IntelliSense) для C#, Razor та JavaScript;
- Відображення документації та підказок у реальному часі;
- Швидкий перехід до визначення методу або класу (Go To Definition);
- Автоматичний рефакторинг, перейменування та генерацію властивостей.

Це значно зменшило час на розробку та дозволило уникнути синтаксичних помилок ще до запуску застосунку.

У проєкті передбачено можливість інтеграції з Git, GitHub або іншими системами керування версіями:

- Панель "Git Changes" у Visual Studio відображає зміни в реальному часі;
- Можна створити гілки, коміти, злиття прямо з IDE;
- У перспективі проєкт можна опублікувати на Azure або іншому хостингу через вбудовану функцію «Publish».

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було всебічно досліджено та обґрунтовано вибір технологій, які лягли в основу реалізації розробленого вебзастосунку для трансляції відео в режимі реального часу. Комплексне поєднання таких інструментів, як ASP.NET Core MVC, SignalR, WebRTC, JavaScript, HTML/CSS та інтегроване середовище Microsoft Visual Studio, забезпечило не лише успішну реалізацію необхідної функціональності, але й досягнення високої продуктивності, стабільності та гнучкості системи.

Базовою платформою для створення серверної частини став фреймворк ASP.NET Core MVC, який, завдяки своїй багаторівневій архітектурі (Model–View–Controller), дозволив чітко структурувати код, розділивши логіку обробки запитів, управління даними та генерації представлення. MVC-архітектура забезпечила легке розширення та підтримку застосунку, спростила додавання нових маршрутів, подань і бізнес-логіки. Особливо важливим є той факт, що ASP.NET Core є кросплатформним, тому застосунок може бути розгорнутий як на Windows, так і на Linux-серверах.

Окрему роль у побудові системи відіграла бібліотека SignalR, яка реалізує механізм двосторонньої комунікації між клієнтом і сервером у реальному часі. SignalR використовується у проєкті у двох ключових напрямах: по-перше, як основний канал передачі відеокадрів у форматі base64 у режимі SignalR Stream; по-друге, як сигнальний сервер для ініціалізації та узгодження WebRTC-з’єднання. Такий підхід дозволяє використовувати одну технологію для різних сценаріїв, підвищуючи універсальність та знижуючи складність підтримки коду. SignalR абстрагує розробника від деталей мережевої взаємодії, автоматично вибираючи найефективніший протокол зв’язку, що є великою перевагою в умовах мінливого середовища браузерів і мереж.

Для реалізації прямого потоку відео між браузерами була використана технологія WebRTC — стандартна, сучасна, кросплатформна технологія, що дозволяє встановлювати peer-to-peer з’єднання без проміжного сервера для передачі медіа. Її головними перевагами є мінімальна затримка, висока якість переданого потоку, підтримка NAT traversal через STUN і TURN, а також можливість повної реалізації на клієнтській стороні. У межах реалізації WebRTC-режиму в системі використовуються стандартні API: getUserMedia() для отримання потоку з камери, RTCPeerConnection для встановлення з’єднання та обміну треками, обробка SDP-повідомлень та ICE-кандидатів. Це дало змогу забезпечити реальний живий відеопотік, який не навантажує сервер, працює напряму між клієнтами і забезпечує якість, придатну для сучасних відеозв’язків.

Ключову роль у клієнтській логіці відіграв JavaScript, який забезпечив інтерактивну поведінку інтерфейсу, обробку подій, контроль роботи відеопотоків та взаємодію з SignalR і WebRTC. Через JavaScript реалізовано запуск і зупинку камери, захоплення кадрів, рендеринг зображення на canvas, надсилання даних через хаб, а також створення peer-з’єднань і обробку медіатреків. Таким чином, саме завдяки JavaScript клієнтська частина застосунку стала динамічною, живою та чутливою до взаємодій користувача.

Інтерфейс користувача (UI), побудований із використанням HTML5, CSS3, Razor-шаблонів та JavaScript, забезпечив інтуїтивно зрозумілу навігацію, адаптивність і інтерактивність. Користувач може легко вибрати режим трансляції, керувати запуском і зупинкою камери, бачити результат трансляції в реальному часі. Розробка UI базувалася на принципах мінімалізму й доступності, що дозволяє ефективно використовувати систему навіть недосвідченим користувачам.

Розробка системи здійснювалася в середовищі Microsoft Visual Studio, яке забезпечило зручне створення, конфігурацію, тестування та налагодження

застосунку. Visual Studio надало широкі можливості для роботи з .NET-проєктами, зокрема — автодоповнення коду, перевірку синтаксису в реальному часі, інтеграцію з системами контролю версій (Git), генерацію Razor-шаблонів, підключення SignalR, а також швидке публікування застосунку на сервер.

Усі ці інструменти й технології були не просто використані у проєкті, але й глибоко інтегровані один з одним, утворивши цілісну, ефективну систему для вебтрансляції відео у двох режимах. Результатом стало створення продуктивної, стабільної і масштабованої вебплатформи, що здатна працювати як у простому режимі передачі зображень через сервер (SignalR Stream), так і у високоякісному режимі peer-to-peer відеопотоку (WebRTC).

Таким чином, вибір і застосування розглянутих технологій у другому розділі цілком себе виправдали та дозволили реалізувати систему, яка відповідає сучасним вимогам щодо продуктивності, інтерактивності, масштабованості та якості користувацького досвіду.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1. Загальна структура застосунку

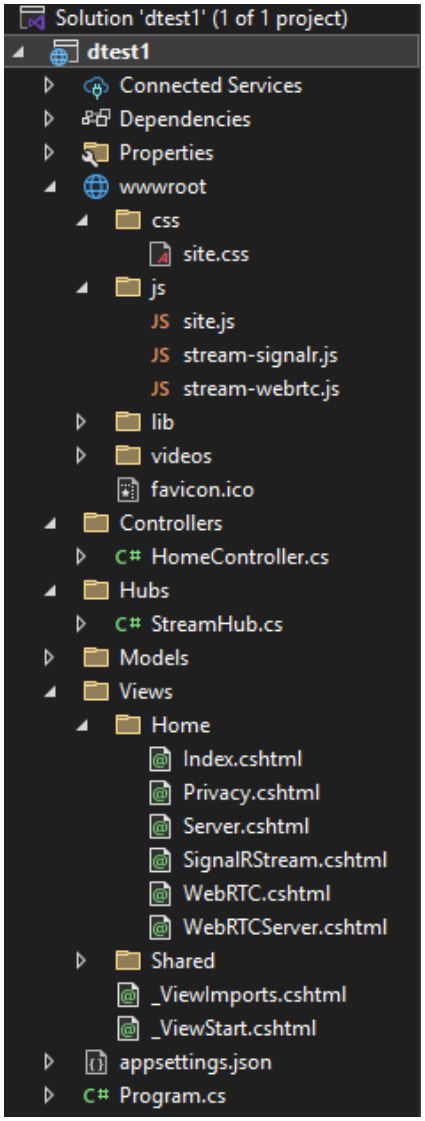

Рисунок 3.1 – структура застосунку на сервері.

Створений вебзастосунок має модульну архітектуру на основі шаблону MVC (Model–View–Controller), що забезпечує чітке розділення логіки між обробкою запитів, генерацією інтерфейсу та обробкою даних. Застосунок дозволяє користувачам передавати відеопотік з вебкамери у два режими:

SignalR-потік — послідовна передача кадрів зображення;

WebRTC-потік — прямий відеозв’язок між клієнтами через peer-to-peer.

Архітектура проєкту складається з таких ключових елементів:

Контролери:

Містить логіку маршрутизації. Основний контролер — HomeController.cs, що відповідає за рендеринг представлень і передачу користувача на відповідні сторінки. Містить такі методи:

Index() — головна сторінка з вибором режиму;

SignalRStream() — сторінка трансляції через SignalR;

Server() — сторінка перегляду SignalR-потоку.

WebRTC() — сторінка трансляції через WebRTC;

WebRTCServer() — сторінка перегляду WebRTC-потоку.

Представлення:

Сховище HTML-шаблонів Razor. Кожен режим має власне представлення з унікальним JS-кодом:

Views/Home/Index.cshtml

Views/Home/SignalRStream.cshtml

Views/Home/Server.cshtml

Views/Home/WebRTC.cshtml

Views/Home/WebRTCServer.cshtml

Ці представлення включають HTML-розмітку, кнопки керування, відеоелементи, а також підключення JavaScript-файлів.

Hubs:

Містить SignalR-хаб StreamHub.cs, який обробляє підключення, передачу кадрів та службові повідомлення для WebRTC. Методи хабу:

SendFrame() — передача зображень у режимі SignalR;

SendOffer() / SendAnswer() / SendCandidate() — сигнальні повідомлення для WebRTC.

wwwroot:

Папка зі статичними файлами: стилі CSS, скрипти JavaScript, зображення.

Файли JavaScript розділено за функціями:

site.css – стилістична розмітка сайту

stream-signalr.js — логіка SignalR-потоку;

stream-webrtc.js — логіка WebRTC-потоку.

Переглянемо кожен файл програми окремо:

HomeController.cs

Контролер серверної логіки вебзастосунку, розробленого за архітектурною моделлю MVC. Зокрема, він реалізує контролер HomeController, який відповідає за обробку запитів користувача та повернення відповідних вебсторінок (представлень) у відповідь на запити.

На початку файлу підключено необхідні простори імен. Підключення Microsoft.AspNetCore.Mvc дозволяє використовувати всі можливості MVC-фреймворку, зокрема базовий клас Controller, метод View() і тип результату IActionResult.

Клас HomeController наслідує від базового класу Controller, що дає змогу використовувати всі вбудовані механізми ASP.NET Core MVC для формування відповідей на запити. Внутрішньо клас містить п’ять методів (Action Methods), кожен з яких повертає HTML-сторінку відповідно до назви методу.

Метод Index() повертає представлення для головної сторінки. Ця сторінка зазвичай виконує роль навігаційного хаба, з якого користувач обирає потрібний режим роботи системи — трансляцію через SignalR або WebRTC.

Метод SignalRStream() повертає сторінку, де реалізовано режим передачі відео у вигляді окремих зображень через SignalR. Тут користувач запускає свою камеру, кадри з неї захоплюються, перетворюються в base64-формат і передаються іншим клієнтам.

Метод WebRTC() відображає сторінку, на якій користувач виступає в ролі джерела відеопотоку через WebRTC. Саме звідси ініціюється створення з’єднання, формування offer-повідомлення і передача відео напряму іншому клієнту.

Метод WebRTCServer() відповідає за перегляд цього потоку — це сторінка, яку відкриває інший користувач (глядач), щоб отримати сигнал, сформувати відповідь (answer) і отримати відео.

Усі методи дій використовують метод View(), який автоматично повертає Razor-сторінку з відповідною назвою з папки Views/Home. Це спрощує навігацію та дозволяє чітко розділити логіку маршрутизації (у контролері) та логіку інтерфейсу (у поданнях).

Контролер формує основу, на яку накладається вся інша функціональність — обробка відеопотоків, підключення до хабу, робота з WebRTC, передача кадрів та багато іншого.

StreamHub.cs

StreamHub успадковується від базового класу Hub, який є частиною бібліотеки Microsoft.AspNetCore.SignalR. Цей клас виступає як центральний вузол обміну повідомленнями в системі. Усі клієнти, які підключаються до хабу, можуть надсилати повідомлення на сервер, а сервер — розсилати ці повідомлення іншим клієнтам.

У даному випадку хаб виконує подвійну функцію:

- Передає кадри відео у вигляді base64-зображень (SignalR-режим).
- Забезпечує обмін службовими повідомленнями WebRTC (offer, answer, ICE-кандидати), які потрібні для встановлення peer-to-peer з’єднання між клієнтами.

Метод SendFrame() викликається клієнтами, які передають зображення (кадри з відео) у вигляді рядка base64. Отримане зображення передається всім іншим підключеним клієнтам (окрім того, хто відправив) за допомогою методу Clients.Others.SendAsync("ReceiveFrame", base64Image).

Це забезпечує імітацію відеопотоку: клієнт надсилає кадри, а інші клієнти відображають їх у вигляді оновлюваного зображення.

Метод SendOffer() приймає об'єкт offer — це SDP-повідомлення, яке генерується ініціатором WebRTC-з’єднання. За допомогою цього повідомлення інший клієнт отримує конфігурацію медіаз’єднання. Хаб розсилає offer усім іншим клієнтам через "ReceiveOffer".

Метод SendAnswer() викликається на стороні клієнта, який прийняв offer і сформував відповідь (SDP answer). Відповідь відправляється на сервер, і сервер передає її іншим клієнтам — як "ReceiveAnswer".

Цей обмін (offer → answer) є обов'язковим кроком у встановленні WebRTC-з'єднання між двома клієнтами.

Метод SendCandidate() приймає об'єкт типу ICE-кандидата. ICE-кандидати є частиною протоколу встановлення з'єднання WebRTC. Вони містять інформацію про маршрути, через які можна встановити прямий зв'язок між клієнтами, навіть якщо ті перебувають за NAT або фаєрволами.

Цей метод передає кандидата іншим клієнтам через "ReceiveCandidate", дозволяючи автоматично добудовувати зв'язок.

Усі методи працюють за принципом:

- Клієнт викликає метод хабу через SignalR-з'єднання.
- Сервер отримує дані й одразу відправляє їх всім іншим клієнтам.
- На стороні клієнтів ці повідомлення обробляються функціями JavaScript (connection.on(...)), які запускають відповідні дії: відображення зображення, обробка offer/answer, додавання кандидата тощо.

StreamHub є центральним вузлом зв'язку в системі, який забезпечує як передачу зображень у простому режимі (SignalR), так і повноцінну сигналізацію для WebRTC. Його універсальність дозволяє не створювати окремі хаби для кожного режиму — одна точка обробляє всі ключові події та мінімізує складність коду.

Таким чином, саме завдяки цьому класу реалізовано можливість зв'язку в реальному часі, без необхідності в складних або сторонніх сервісах сигналізації.

stream-signalr.js

Цей JavaScript-код реалізує режим трансляції відео з вебкамери в реальному часі через технологію SignalR, але не як потокове відео, а у вигляді послідовності зображень (кадрів), що імітують відео. Такий підхід часто називають "квазі-стрімом". Нижче докладно пояснюється, як саме працює цей код.

1. Ініціалізація елементів

Отримується доступ до HTML-елемента <video> із id="video", у якому буде відображатися відео з камери користувача.

- Змінна isStreaming вказує, чи активна трансляція.
- Змінна stream зберігатиме об’єкт MediaStream, що повертається з камери.
- interval — це ідентифікатор таймера, який буде використовуватись для періодичної відправки кадрів.

2. Підключення до SignalR-хабу

Створюється нове підключення до SignalR-хабу за URL /streamHub.

3. Викликається метод start(), який ініціює зв’язок із сервером.

Це дозволяє JavaScript-клієнту викликати методи на сервері, такі як SendFrame, і отримувати зворотні повідомлення, якщо потрібно.

4. Обробка натискання кнопки трансляції

- Коли користувач натискає кнопку з id="toggleStream", виконується обробник події.
- Обробник реалізує перемикання трансляції — вмикання або вимикання потоку залежно від поточного стану.

5. Увімкнення трансляції

- Запит до камери користувача за допомогою getUserMedia.
- Отриманий відеопотік виводиться у <video>-елемент.
- Створюється тимчасовий canvas, розміром 480x360.
- Через getContext("2d") створюється графічний контекст для малювання кадрів.
- Кожні 100 мс поточне зображення з <video> рендериться на <canvas>.
- Потім воно перетворюється на base64-рядок
- Передача на сервер через SignalR методом SendFrame.
- Це і є імітація відеопотоку: послідовність зображень надсилається на сервер з частотою 10 кадрів на секунду.

6. Вимкнення трансляції

- Зупиняється таймер, який надсилав кадри.
- Всі треки відео зупиняються, камера вимикається.
- Елемент <video> очищується.
- Статус isStreaming змінюється на false.
- Це гарантує, що трансляція припиняється повністю, і не продовжується у фоновому режимі.
- Це — проста, але ефективна реалізація квазі-стріму через SignalR, яка не потребує складної конфігурації WebRTC, але дозволяє передавати відео в обхід обмежень щодо реального потокового відео. Такий підхід зручний для тестування, локальних мереж або коли потрібна висока сумісність із різними браузерами.

stream-webrtc.js

Цей JavaScript-код реалізує повноцінний живий відеопотік через WebRTC із використанням SignalR як сигнального каналу. Код дозволяє клієнту підключитися до іншого браузера і транслювати відео у режимі реального часу через peer-to-peer з'єднання. Нижче пояснено детально, як саме працює кожна частина.

1. Отримуються DOM-елементи:

- localVideo — елемент для відображення відео з власної камери;
- remoteVideo — для показу відео, що надходить від іншого користувача;
- toggleBtn — кнопка, яка запускає або зупиняє WebRTC-потік.

2. Оголошуються глобальні змінні:

- pc — об'єкт RTCPeerConnection, що встановлює peer-to-peer з'єднання;
- localStream — потік із камери;
- connection — об'єкт SignalR-з'єднання;
- isOn — прапорець, який вказує, чи активна трансляція.

3. Функція startWebRTC

Створюється з'єднання з хабом SignalR на сервері за URL /streamHub.

Обробники отриманих повідомлень, дозволяють приймати службові повідомлення WebRTC (SDP-обмін та ICE-кандидати), які надсилаються іншими клієнтами:

- ReceiveOffer — отримання початкового опису з'єднання;
- ReceiveAnswer — прийняття відповіді на запит;
- ReceiveCandidate — додавання мережевого маршруту.

Запуск SignalR-з’єднання

Камера активується, зображення виводиться в елемент <video> на сторінці.

Створюється об’єкт для peer-to-peer з’єднання.

Кожен ICE-кандидат передається через SignalR, щоб допомогти знайти найкращий маршрут між клієнтами.

Коли інший клієнт передає потік, він з’являється у remoteVideo.

Перед тим як надіслати offer, локальні відеотреки додаються у з’єднання.

Формується offer — опис того, який потік і як користувач хоче передати. Він надсилається іншому клієнту через SignalR.

Обробка натискання кнопки

Це реальний потік відео без буферизації і з мінімальними затримками, що забезпечує високу якість зв’язку між двома браузерами. Такий підхід є сучасним стандартом для відеочатів, трансляцій, конференцій та онлайн-моніторингу.

Program.cs

Цей код є файлом стартової конфігурації ASP.NET Core застосунку, зазвичай розташованим у Program.cs. Він визначає, як запускається вебзастосунок, які сервіси підключаються, які маршрути використовуються, та як обробляються запити. Розглянемо поетапно, що саме тут відбувається.

1. Створення вебзастосунку

- var builder = WebApplication.CreateBuilder(args);

Цей рядок створює об’єкт builder, який конфігурує застосунок перед запуском. Сюди передаються параметри командного рядка, якщо є.

2. Реєстрація сервісів

У цьому фрагменті до контейнера залежностей додаються необхідні служби:

- AddControllersWithViews() — реєструє підтримку MVC-патерну, тобто використання контролерів та Razor-сторінок.
- AddSignalR() — реєструє SignalR як сервіс у застосунку, щоб дозволити використання хабів і з’єднань у реальному часі.

Цей крок критично важливий для того, щоб система підтримувала як стандартну навігацію сторінками, так і передачу даних у реальному часі.

3. Налаштування обробки HTTP-запитів

Якщо застосунок не в режимі розробки, вмикається обробник помилок, який перенаправляє користувача на сторінку /Home/Error у разі винятку.

Встановлюється HSTS (HTTP Strict Transport Security), що примушує клієнтів завжди використовувати HTTPS.

4. Інші базові middleware:

- UseHttpsRedirection() — перенаправляє всі HTTP-запити на HTTPS.
- UseStaticFiles() — дозволяє обслуговувати CSS, JavaScript, зображення та інші ресурси з папки wwwroot.

5. Налаштування маршрутизації

- UseRouting() активує механізм маршрутизації.
- UseAuthorization() підключає систему авторизації (навіть якщо вона поки не використовується).

6. Визначення маршрутів

- app.MapControllerRoute();

Цей рядок задає маршрут за замовчуванням для всієї програми:

7. Реєстрація SignalR-хабу

- app.MapHub<dtest1.Hubs.StreamHub>("/streamHub");

Цей рядок дозволяє підключати SignalR-клієнтів до хабу StreamHub за адресою /streamHub.

- new signalR.HubConnectionBuilder().withUrl("/streamHub").build();

Завдяки цьому зв’язку клієнти можуть викликати методи на сервері (SendFrame, SendOffer тощо) та отримувати зворотні повідомлення.

8. Запуск застосунку

- app.Run();

Це фінальний виклик, що фактично запускає вебсервер. З цього моменту застосунок готовий обробляти HTTP-запити, обслуговувати клієнтів, підключати SignalR-з’єднання тощо.

Це — основна інфраструктура, центральною точкою запуску ASP.NET Core застосунку.

SignalRStream.cshtml

Цей код є Razor-поданням (файлом типу .cshtml) у проєкті ASP.NET Core MVC. Він відповідає за відображення сторінки, де користувач може транслювати відео з вебкамери через SignalR у вигляді кадрів (зображень). Це — клієнтська частина простого потоку на основі передачі окремих знімків.

1. Заголовок сторінки

Цей блок задає заголовок сторінки, який буде відображений у браузері. ViewData["Title"] використовується для динамічного встановлення значення <title> у шаблоні макета (_Layout.cshtml).

2. Заголовок вмісту

Це простий заголовок сторінки, який пояснює користувачу, що він перебуває на сторінці трансляції кадрів через SignalR.

3. Відеоелемент

- id="video" — потрібен для доступу до елемента з JavaScript;
- autoplay — відео відразу запускається;
- muted — відео вимкнене за замовчуванням (інакше браузери можуть блокувати його);
- width/height — розміри відображення.

У цьому елементі відображається відеопотік до того, як кадри будуть захоплені через <canvas> і відправлені через SignalR.

4. Кнопки керування

Ця кнопка активує або зупиняє трансляцію:

- при першому натисканні — запускає камеру та починає захоплення кадрів;
- при повторному — зупиняє камеру і припиняє передачу зображень.

JavaScript, прив'язаний до цієї кнопки, міститься у файлі stream-signalr.js.

Ця кнопка відкриває нову вкладку або вікно з серверною сторінкою /Home/Server, на якій, імовірно, знаходиться глядач, що приймає і відображає кадри, отримані через SignalR.

5. Підключення бібліотек

Підіключення офіційної SignalR бібліотеки для JavaScript, яка дозволяє

клієнту створювати з’єднання з SignalR-хабом, викликати методи, надсилати повідомлення, отримувати зворотні дані тощо.

Це подання відповідає за запуск потоку кадрів з вебкамери в режимі SignalR. Воно є частиною спрощеної системи відеотрансляції, яка не використовує WebRTC, але дозволяє реалізувати стрімінг зображень у реальному часі між браузерами через сервер. Такий підхід є ефективним для невеликих проєктів або локальних демонстрацій.

Server.cshtml

Цей .cshtml файл є Razor-представленням ASP.NET Core MVC застосунку та виконує роль серверної сторінки або глядача, яка приймає та відображає кадри відео, що надходять від клієнта через SignalR. По суті, це споживач відеопотоку у вигляді послідовних зображень, що оновлюються в реальному часі.

1. Заголовок сторінки

Цей блок задає заголовок вебсторінки, який використовується в <title> макета (_Layout.cshtml). У цьому випадку заголовок — «Server View».

2. Елемент для виведення зображень

- Атрибут id="stream" дозволяє доступ до нього з JavaScript.
- Атрибути width і height задають розмір зображення.

- Кожен новий кадр, отриманий із SignalR-хабу, буде вставлятись у src цього зображення.

3. Підключення бібліотеки SignalR

Це підключення офіційної клієнтської бібліотеки SignalR JavaScript, необхідної для створення з’єднання з хабом на сервері.

4. Основна JavaScript-логіка

- Створюється посилання на елемент <img>.
- Ініціалізується нове SignalR-з’єднання до хабу /streamHub.
- Оголошується обробник для події "ReceiveFrame", яка надходить із сервера.
- Коли сервер (через хаб StreamHub) надсилає новий кадр у форматі base64, цей рядок встановлюється як src для зображення.
- Браузер миттєво оновлює картинку, що створює ефект анімації (стріму).
- Запускається SignalR-з’єднання. Коли з’єднання встановлено, клієнт готовий до приймання кадрів.

Це проста, але ефективна реалізація перегляду відеопотоку через SignalR. Вона працює в усіх сучасних браузерах без додаткових модулів. Цей підхід є зручним для реалізації локальних стрімів, демонстрацій або спостереження, хоча має обмеження у FPS та якості в порівнянні з WebRTC.

WebRTC.cshtml

Цей .cshtml файл є Razor-представленням ASP.NET Core MVC застосунку та відповідає за інтерфейс клієнтської частини WebRTC-режиму. Його завдання — забезпечити користувачу можливість запустити відеопотік із камери у режимі реального часу через WebRTC, використовуючи SignalR як

сигнальний механізм для встановлення peer-to-peer з’єднання з іншим клієнтом.

1. Заголовок сторінки

Установлює заголовок сторінки, що буде виведений у <title> у головному шаблоні сайту (_Layout.cshtml). Тут назва — "WebRTC".

2. Відеоелементи

- localVideo — елемент для перегляду власного відеопотоку з вебкамери. muted вимикає звук, щоб уникнути самовідтворення із блокуванням.
- remoteVideo — призначений для перегляду відео, яке надходить від іншого користувача після встановлення WebRTC-з’єднання.

Обидва мають autoplay, тому відео починається одразу після завантаження.

3. Кнопки керування

Ця кнопка активує або зупиняє трансляцію:

- Захоплює відео з камери;
- Ініціює з’єднання з іншим клієнтом;
- У разі повторного натискання — зупиняє трансляцію, закриває з’єднання, вимикає камеру.

JavaScript, прив'язаний до цієї кнопки, міститься у файлі stream-webrtc.js.

- Кнопка, яка відкриває новому вікні сторінку /Home/WebRTCServer. Саме там перебуватиме глядач, який отримає відеопотік від клієнта. В обох браузерах має бути активна SignalR-логіка, яка дозволить обмінятися SDP та ICE-кандидатами.

5. Підключення бібліотек

Підіключення офіційної SignalR бібліотеки для JavaScript, яка дозволяє клієнту створювати з’єднання з SignalR-хабом, викликати методи, надсилати повідомлення, отримувати зворотні дані тощо.

Це представлення реалізує клієнтську частину WebRTC-трансляції. Воно дозволяє користувачу передавати відео у реальному часі іншому користувачу (глядачу) через браузер. У поєднанні з SignalR для сигналізації ця сторінка створює повноцінний відеозв’язок без центрального медіасервера, що є сучасним і ефективним рішенням у веброзробці.

WebRTCServer.cshtml

Цей .cshtml файл реалізує серверну частину (глядача) WebRTC-режиму у застосунку. Це сторінка, на якій відображається відеопотік, що передається з іншого браузера через WebRTC peer-to-peer з’єднання. SignalR тут використовується лише як сигнальний канал для обміну службовими повідомленнями між двома клієнтами: offer, answer та ICE-кандидатами.

1. Установлюється заголовок сторінки

2. Елемент відео

- id="remoteVideo" — потрібен для JavaScript, щоб вставити потік у DOM.
- autoplay — відео запускається автоматично.
- playsinline — забезпечує відтворення в межах сторінки.
- controls — додає панель управління програвачем (пауза, гучність тощо).
- width/height — визначають розмір відео.

3. Підключення офіційної бібліотеки Microsoft SignalR для JavaScript

4. Обробка події "ReceiveOffer"

- Створює новий WebRTC-з’єднувач (RTCPeerConnection).
- Встановлює обробник ICE-кандидатів, які буде надсилати назад через SendCandidate.
- Встановлює обробник прийому медіа — коли приходить відео, воно прив’язується до remoteVideo.
- Приймає offer — тобто початкову пропозицію з'єднання.
- Створює answer — відповідь на пропозицію.
- Надсилає answer назад через SignalR.

5. Обробка події "ReceiveCandidate"

- Отримані ICE-кандидати — це маршрути для з’єднання, які потрібно додати до WebRTC-підключення. Це дозволяє WebRTC знайти найоптимальніший шлях між клієнтами, навіть через NAT або фаєрвол.

6. Запуск SignalR

Активує з’єднання з хабом SignalR. Після цього клієнт готовий отримувати offer і candidate.

Користувач відкриває цю сторінку у браузері (глядач). Коли інший користувач (транслятор) запускає WebRTC на своїй сторінці, він надсилає offer. Ця сторінка приймає offer, створює answer і повертає його. Клієнти обмінюються ICE-кандидатами. Відео починає надходити і відображається в remoteVideo.

Переваги реалізації

- Висока якість відео, оскільки передача відбувається напряму, без серверного проміжного посередника.
- Низька затримка — ідеально для відеозв’язку, нагляду або онлайн-конференцій.
- Працює повністю в браузері, без необхідності в додаткових плагінах або встановленні софту.

Це представлення реалізує сторінку WebRTC-глядача — іншого учасника peer-to-peer з’єднання, який приймає відео потік. У поєднанні зі сторінкою транслятора (/Home/WebRTC) і хабом /streamHub, ця реалізація дозволяє створити живе відеоз’єднання між двома браузерами у реальному часі, з використанням WebRTC для медіапотоку і SignalR для сигналізації.

У системі чітко розмежовано функціональність WebRTC і SignalR.

SignalR використовується у двох випадках:

- Як транспорт для передачі зображень у режимі SignalR.
- Як сигнальний канал для передачі SDP (Session Description Protocol) та ICE (Interactive Connectivity Establishment) повідомлень у режимі WebRTC.

WebRTC відповідає за передачу відео напряму між браузерами через RTCPeerConnection, використовуючи медіатреки з камери.

Таке розділення дозволяє реалізувати як просту, так і повноцінну трансляцію з оптимальною продуктивністю, залежно від обраного режиму.

3.2. Реалізація SignalR-потоку

SignalR-потік у даній системі виконує функцію базового способу трансляції відео з вебкамери користувача до іншого учасника (глядача), при цьому дані передаються не у вигляді відеопотоку, а у вигляді послідовних зображень (кадрів), заздалегідь закодованих у формат base64.

Цей підхід має меншу вимогу до сумісності браузерів, дозволяє працювати без WebRTC, та може бути використаний у спрощених або локальних конфігураціях, коли якість відео і FPS не критичні.

У режимі SignalR задіяні наступні компоненти:

- Отримання потоку з вебкамери
- Створення таймера (setInterval) для періодичного захоплення кадрів;
- Малювання зображення у canvas;
- Перетворення кадра у base64 (toDataURL);
- Відправлення кадра на сервер (connection.invoke("SendFrame", base64Image)).
- Сервер приймає повідомлення через метод SendFrame(string base64Image);
- Сурвер відправляє зображення всім іншим клієнтам через "ReceiveFrame".
- Користувач підключається до SignalR-хабу;
- Отримує зображення з сервера (connection.on("ReceiveFrame", ...));
- Встановлюється src атрибут у <img> елементі, який оновлює зображення.

Переваги:

- Простота реалізації;
- Сумісність з усіма браузерами;

- Можна легко реалізувати на сервері ASP.NET без сторонніх серверів;
- Дає змогу побачити роботу системи навіть без WebRTC.

Недоліки:

- Низький FPS (3–10 кадрів/сек при високому навантаженні);
- Високе навантаження на мережу при зростанні частоти кадрів;
- Відсутність звуку (тільки відео);
- Високе споживання ресурсів при тривалому використанні.

3.3. Реалізація WebRTC-потоку

WebRTC-потік реалізує живу трансляцію відео з вебкамери клієнта на інший клієнт, минаючи сервер при передачі медіа. Цей підхід забезпечує:

- мінімальну затримку (latency);
- високу якість відео;
- низьке навантаження на сервер;
- реальну P2P-комунікацію в браузері.

Режим WebRTC складається з таких частин:

- Клієнт захоплює відео з вебкамери через getUserMedia();
- Створюється об’єкт RTCPeerConnection із конфігурацією STUN-сервера;
- Відео додається в з’єднання як трек;
- Клієнт генерує offer і надсилає його через SignalR до отримувача;
- Отримувач встановлює RemoteDescription, створює answer, надсилає його назад;
- Обидві сторони обмінюються ICE-кандидатами;

- Після узгодження з’єднання встановлюється, і потік починає передаватися напряму.
- Клієнт: створення offer (відправник)
- Сервер/інший клієнт: прийом offer та відповідь
- Обмін ICE-кандидатами
- Відправка
- Відображення відео

3.4. Взаємодія сторін

У межах системи реалізовано взаємодію двох сторін — клієнта (джерела відео) та серверної/переглядової частини (глядача). Вони функціонують по-різному залежно від вибраного режиму трансляції, але загальна логіка залишає незмінною роль кожного.

Клієнти (як відправник, так і глядач) підключаються до SignalR-хабу через метод HubConnectionBuilder. Вони викликають методи invoke для надсилання даних, і використовують метод on, щоб обробити вхідні повідомлення.

SignalR-хаб (StreamHub) ретранслює повідомлення від одного клієнта до всіх інших, що дозволяє обмінюватися як кадрами (у SignalR-режимі), так і сигнальними SDP/ICE повідомленнями (у WebRTC-режимі).

У кожному режимі передбачено окремі кнопки для запуску або зупинки трансляції. JavaScript обробляє кліки на ці кнопки через onclick, активуючи відповідні функції (startStreaming, stopStreaming, connectToPeer, тощо).

Стан кнопок змінюється динамічно — наприклад, після початку трансляції кнопка «Почати» вимикається, а кнопка «Зупинити» активується.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі дипломної роботи було докладно розглянуто процес реалізації системи вебтрансляції відео в браузері з використанням сучасних технологій — ASP.NET Core MVC, SignalR та WebRTC. Основна мета полягала у створенні двох повноцінних режимів відеотрансляції: через передачу окремих кадрів із використанням SignalR та через пряме peer-to-peer з'єднання з використанням WebRTC. Система реалізована з використанням сучасних технологій, зокрема ASP.NET Core MVC, SignalR, WebRTC та JavaScript API.

Реалізацію розпочато зі створення загальної структури застосунку, що базується на шаблоні MVC — тобто чіткому поділі на модель (Model), представлення (View) та контролери (Controller). У контролері HomeController реалізовано маршрутизацію між головною сторінкою, режимами трансляції та серверними вікнами для прийому потоку. Це забезпечило логічну навігацію в системі, а також зрозумілий інтерфейс користувача.

Серверна логіка обробки повідомлень у реальному часі була реалізована у вигляді SignalR-хабу StreamHub. У цьому хабі визначено набір методів — SendFrame, SendOffer, SendAnswer, SendCandidate — які дозволяють клієнтам обмінюватися як кадрами (у SignalR-режимі), так і службовими повідомленнями для встановлення WebRTC-з'єднання. Таким чином, SignalR виконує роль універсального каналу комунікації, забезпечуючи як обмін відеоданими, так і сигналізацію.

Режим передачі кадрів через SignalR реалізовано шляхом захоплення відео з вебкамери користувача за допомогою JavaScript-функції getUserMedia(), рендерингу зображення на canvas, конвертації у формат base64, і надсилання через метод SendFrame до SignalR-хабу. На стороні глядача зображення отримуються через подію ReceiveFrame і відображаються

у <img>, що створює ілюзію відеопотоку. Такий підхід хоч і простий у реалізації, але має обмеження за частотою кадрів, затримкою та якістю зображення.

Режим WebRTC більш досконалий і технічно ефективний, він створює пряме peer-to-peer з'єднання між браузерами за допомогою RTCPeerConnection і MediaStream. Ці об'єкти дозволяють захоплювати відео, додавати медіатреки, створювати SDP-повідомлення та обмінюватися ICE-кандидатами для встановлення надійного зв'язку. SignalR виконує роль сигнального каналу, забезпечуючи обмін службовою інформацією для встановлення WebRTC-з'єднання. Таким чином, SignalR відповідає за сигналізацію, а WebRTC — за передачу медіа.

Всі сторінки реалізовані як Razor-представлення з JavaScript-логікою. Інтерфейс створено на HTML5 і CSS з кнопками для керування трансляцією та елементами <video> і <img> для відображення відео. Фронтенд працює в реальному часі, взаємодіючи з сервером без перезавантаження сторінки.

Було реалізовано два окремих режими:

- SignalR-потік — як простий спосіб передачі кадрів у режимі реального часу, який добре підходить для демонстрацій, моніторингу, де висока частота не критична.
- WebRTC-потік — як сучасне рішення для високоякісної та малозатратної прямої передачі відео, яке не навантажує сервер і забезпечує найкращу якість з'єднання.

Таким чином, реалізовано два відеоканали з власними перевагами та призначенням. Система забезпечує просту й ефективну передачу відео в реальному часі і може слугувати основою для складніших рішень — відеоспостереження, онлайн-конференцій, дистанційного моніторингу тощо.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОЇ СИСТЕМИ

В цьому розділі буде розглянуто інтерфейс та роботу розробленої системи.

Спершу запуск програми з vs studio 2022 (р-4.1):

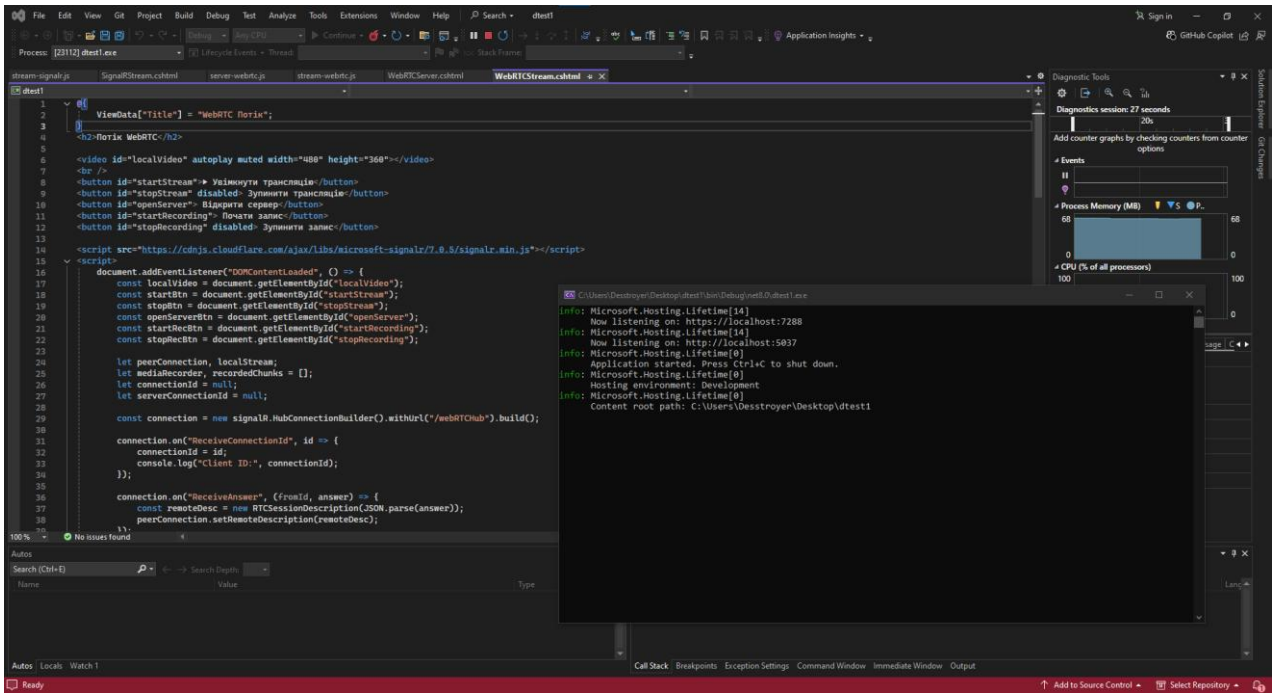

Рисунок 4.1 – запуск програми.

Система може бути розглянута на двох портах локального серверу:

- https://localhost:7015 – захищене з’єднання через HTTPS порт
- http://localhost:5096 – незахищене з’єднання через HTTP порт

Кожен метод має чотири кнопки, які реалізовують функіонал:

1. Увімкнути/Зупинити – кнопка для запуску трансляції та її зупинки.
2. Відкрити сервер – кнопка, яка відкриває нову вкладку на якій знаходиться серверна частина, яка приймає потік даних.
3. Почати запис – кнопка для запису відео.
4. Зупинити запис – кнопка для зупинки запису.

Відбулося успішне з’єднання клієнта з сервером потоком через SignalR (р-4.2):

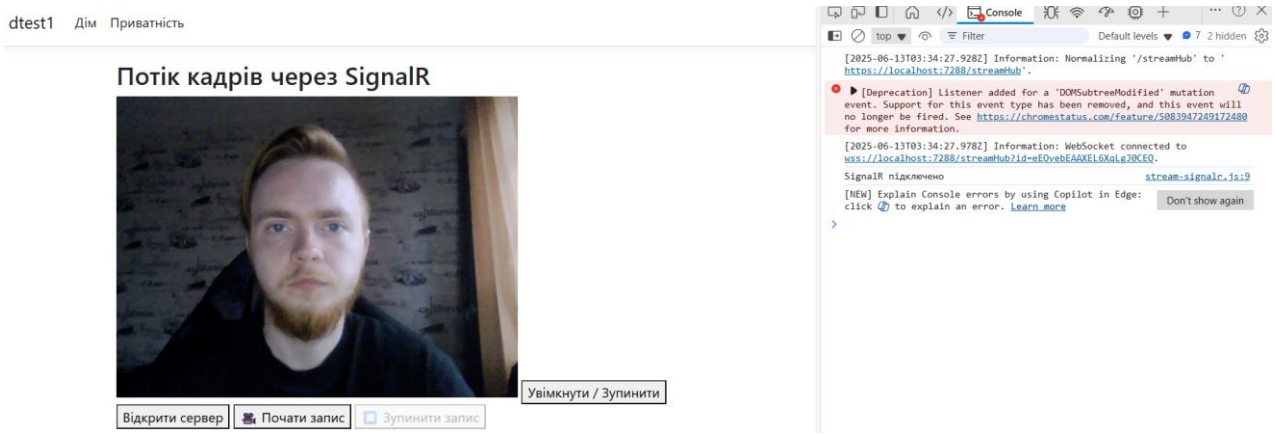

Рисунок 4.2 – успішне з’єднання клієнта з сервером через SignalR.

Огляд роботи SignalR-потоку (р-4.3):

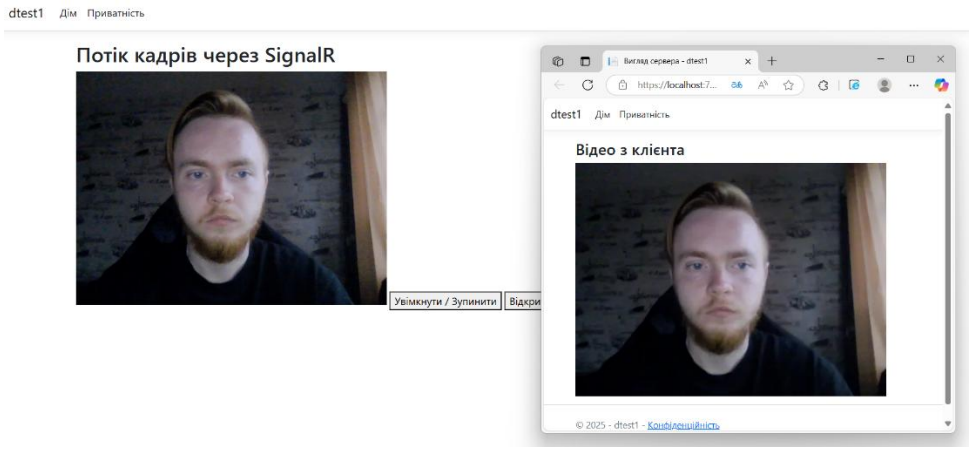

Рисунок 4.3 – результат працюючої камери через SignalR.

Запускаємо трансляцію вебкамери зі сторони клієнта, та отримуємо її на серверній частині.

Запис відео та збереження його на серверній частині у папці wwwroot/videos та у папку завантажень зі сторони клієнта (р-4.4):

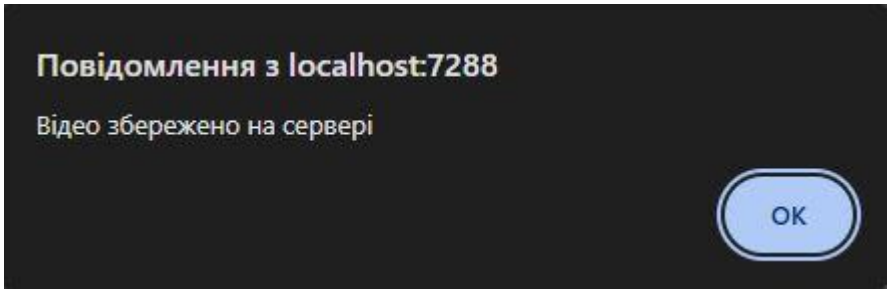

Рисунок 4.4 – результат запису відео на сервер та клієнтський пристрій.

Збереження на сервер(р-4.5):

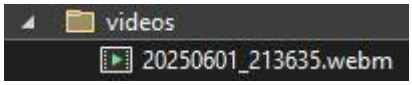

Рисунок 4.5 – відео збережено на сервер.

Запис відео та збереження його на серверній частині у папці wwwroot/videos та у папку завантажень зі сторони клієнта.

Огляд збереженого відео на клієнтському пристрої(р-4.5):

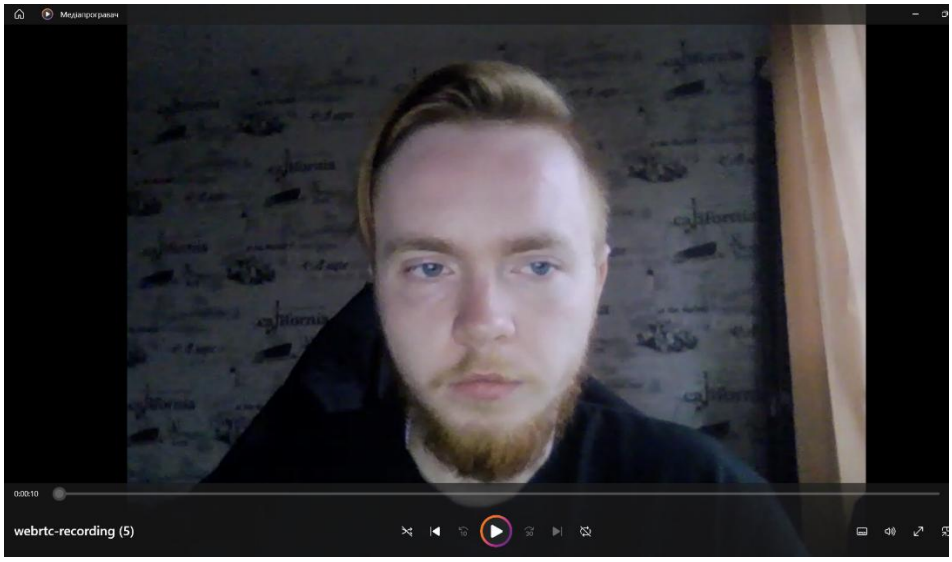

Рисунок 4.5 – збереження та огляд відео на клієнтському пристрої.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі представлено готовий застосунок, який розроблявся протягом усієї дипломної роботи. Описано загальні вказівки щодо використання застосунку, а також розглянуто можливі випадки неправильного використання, що можуть призвести до помилок або некоректної роботи.

Було проведено дослідження ефективності роботи алгоритмів для передачі відео через SignalR і WebRTC, а також оптимальні параметри для налаштування обміну відеопотоками в реальному часі між клієнтом і сервером. Розглянуті варіанти підключення, визначені кроки для ініціалізації з'єднання і обміну даними, а також аналіз продуктивності при використанні цих технологій.

Далі наведено тест-кейси для тестування функціональності застосунку, які є критично важливими для перевірки коректності роботи основних компонентів, зокрема передачі відео, підключення клієнта та сервера, а також обробки помилок у процесі взаємодії між ними.

ВИСНОВКИ

У дипломній роботі було досліджено, спроєктовано, реалізовано та протестовано вебзастосунок для передачі відеопотоків у режимі реального часу. На етапі аналізу предметної області було визначено, що технологія WebRTC є найсучаснішим та найефективнішим рішенням для реалізації прямих peer-to-peer з'єднань між клієнтами, а її поєднання із SignalR дозволяє організувати надійний сигнальний обмін без потреби у складній серверній інфраструктурі. На основі проведеного технологічного огляду було обґрунтовано вибір інструментів — ASP.NET Core MVC, SignalR, WebRTC, JavaScript, HTML/CSS — які забезпечили гнучкість, масштабованість і продуктивність системи.

Процес реалізації системи показав, що комбінація двох режимів роботи — через SignalR (передача кадрів) та WebRTC (пряме відеоз'єднання) — дозволяє створити універсальну платформу, яка підходить для різних сценаріїв використання: від простих демонстрацій до повноцінних онлайн-конференцій. Важливим етапом роботи стало тестування готового застосунку, яке підтвердило ефективність вибраних рішень, стабільність роботи основних алгоритмів, а також виявило оптимальні параметри для налаштування обміну даними в реальному часі.

Завдяки проведеній роботі вдалося досягти поставленої мети — створити сучасний, інтерактивний вебзастосунок, який відповідає вимогам якості, продуктивності та зручності використання, а також може слугувати основою для подальшого розвитку більш складних рішень у сфері відеозв'язку та онлайн-комунікацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Повний посібник з мови програмування C# 13 та платформі .NET 9 [Електронний ресурс] - Режим доступу до ресурсу: https://metanit.com/sharp/tutorial/ - Дата звернення 10.01.2025

2. Документація з мови C# [Електронний ресурс] - Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/dotnet/csharp/ - Дата звернення 12.01.2025

3. ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу:

https://learn.microsoft.com/ru-ru/dotnet/csharp/ - Дата звернення 27.01.2025

4. ASP.NET Core[Електронний ресурс] - Режим доступу до ресурсу:

https://dotnet.microsoft.com/ru-ru/apps/aspnet - Дата звернення 27.01.2025

5. Посібник. Початок роботи з ASP.NET Core SignalR [Електронний ресурс] - Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/signalr?view=aspnetcore-9.0&tabs=visual-studio – Дата звернення 02.02.2025

6. ASP.NET Core в реальному часі з SignalR[Електронний ресурс] - Режим доступу до ресурсу: https://dotnet.microsoft.com/ru-ru/apps/aspnet/signalr - Дата звернення 11.02.2025

7. WebRTC [Електронний ресурс] - Режим доступу до ресурсу: https://habr.com/ru/articles/163527/ - Дата звернення 12.02.2025

9. .NET P2P [Електронний ресурс] - Режим доступу до ресурсу: https://learn.microsoft.com/en-us/archive/msdn-magazine/2001/february/net-p2p-writing-peer-to-peer-networked-apps-with-the-microsoft-net-framework - Дата звернення 18.02.2025

10. Reddit [Електронний ресурс] - Режим доступу до ресурсу: https://www.reddit.com/ - Дата звернення 18.02.2025

11. Сучасний підручник з JavaScript [Електронний ресурс] - Режим доступу до ресурсу: https://uk.javascript.info/ - Дата звернення 25.02.2025

12. Клієнт JavaScript ASP.NET Core SignalR [Електронний ресурс] - Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/aspnet/core/signalr/javascript-client?view=aspnetcore-9.0&tabs=visual-studio – Дата звернення 08.03.2025

Додаток

1. Схема роботи веб-застосунку(р-5.1):

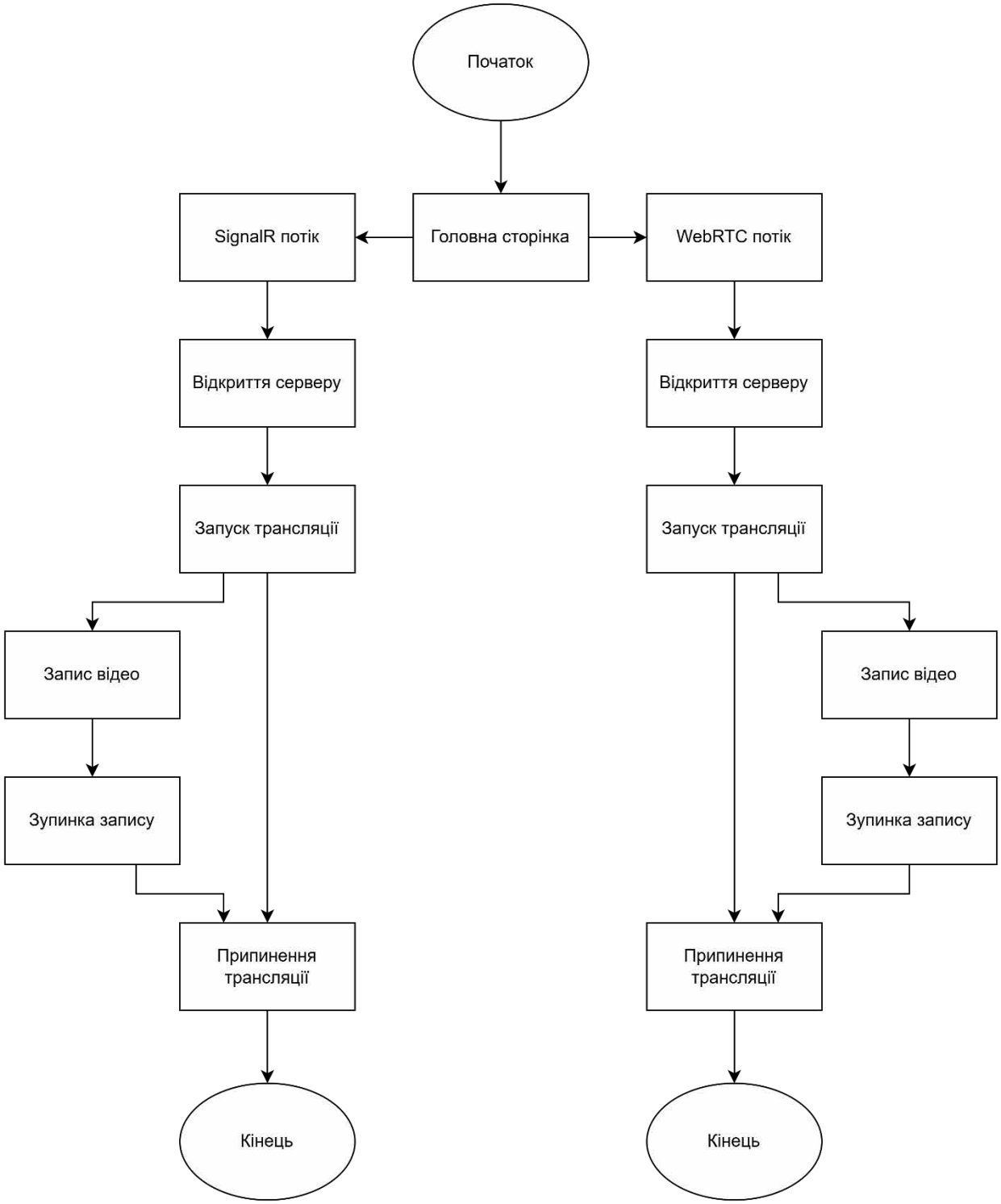

Рисунок 5.1 – робота веб-застосунку.

2. Структурна схема архітектури веб-застосунку(р-5.2):

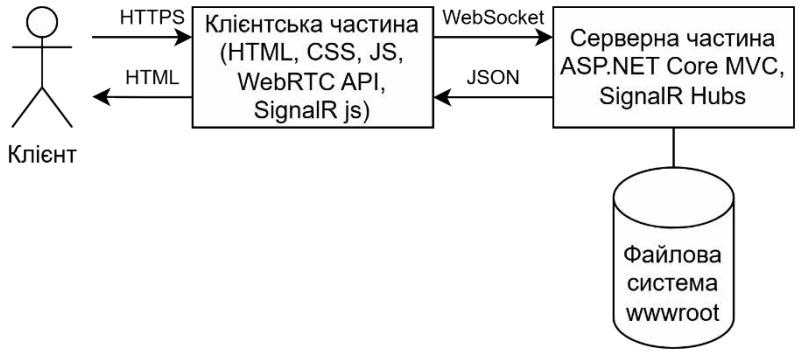

Рисунок 5.2 – архітектура веб-застосунку.

3. Схема сигналізації WebRTC потоку(р-5.3):

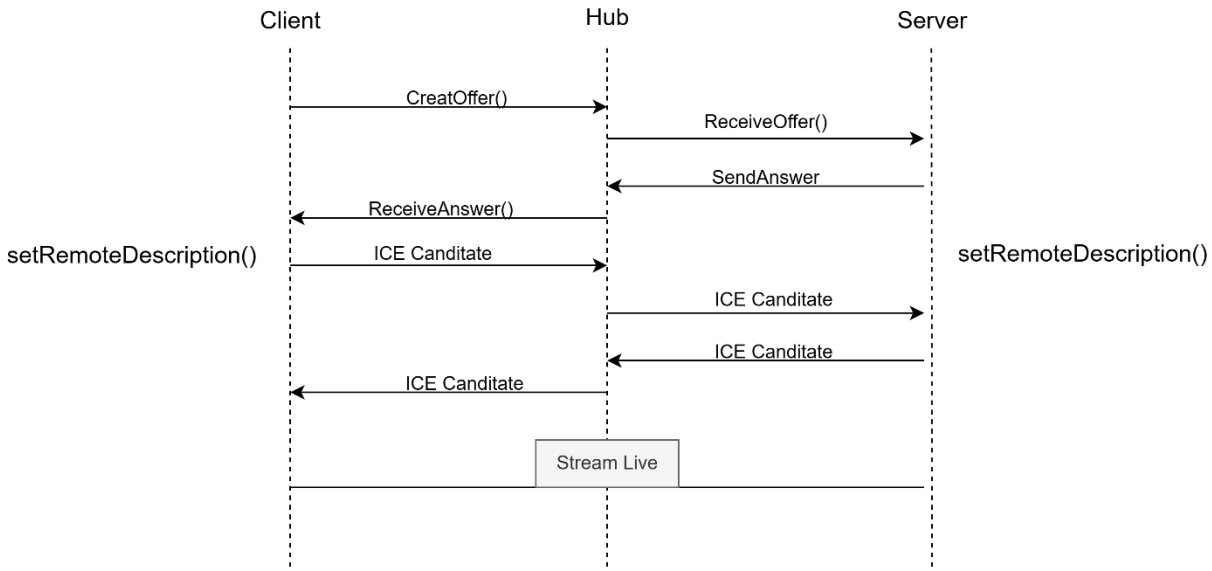

Рисунок 5.3 – сигналізація WebRTC потоку.