Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук та технологій
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олександр ЛИТВИНЕНКО
"_____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: "Браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ"

Виконавець: Кирило КАРМАЗІН

Керівник: Ольга СУПРУН

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук та технологій
Кафедра комп’ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр ЛИТВИНЕНКО
“_____” _____2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Кармазіна Кирила Віталійовича
(прізвище, ім'я, по батькові здобувача вищої освіти в родовому відмінку)

1. Тема кваліфікаційної роботи “Браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ”
затверджена наказом ректора від « 28 » серпня 2023 р. № 1494/ст
2. Термін виконання роботи (проєкту): з 02.10.2023 по 31.12.2023
3. Вихідні дані до роботи (проєкту): середовище розробки *Visual Studio Code*, мова програмування *JavaScript*
4. Зміст пояснювальної записки:
1) аналіз існуючих сервісів для одночасного проведення прямих трансляцій на кілька платформ;
2) опис використаних технологій;
3) розробка браузерного відео стрімінгового сервісу.
5. Перелік обов’язкового графічного (ілюстративного) матеріалу:
1) взаємодія модулів браузерного сервісу;
2) діаграма пакетів браузерного сервісу;
3) робота браузерного сервісу (схема алгоритму);
4) діаграма послідовності створення з’єднання;

5) вікна браузерного сервісу.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Аналіз існуючих сервісів для одночасного проведення прямих трансляцій на кілька платформ	02.10.2023 – 08.10.2023	
2	Написання першого розділу пояснювальної записки	09.10.2023 – 16.10.2023	
3	Проведення огляду літературних джерел	17.10.2023 – 23.10.2023	
4	Написання другого розділу пояснювальної записки	24.10.2023 – 31.10.2023	
5	Розробка браузерного відео стрімінгового сервісу	01.11.2023 – 07.11.2023	
6	Написання третього розділу пояснювальної записки	08.11.2023 – 15.11.2023	
7	Оформлення пояснювальної записки	16.11.2023 – 22.11.2023	
8	Підготовка графічних матеріалів та презентації	23.11.2023 – 30.11.2023	

7. Дата видачі завдання: «_02_» _жовтня_ 2023 р.

Керівник кваліфікаційної роботи _____ Супрун О.М.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Кармазін К.В.
(підпис здобувача вищої освіти) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ»: 83 сторінки, 31 рисунок, 25 літературних джерел.

Ключові слова: СТРІМІНГОВИЙ СЕРВІС, ТРАНСЛЯЦІЯ, *HTML*, *CSS*, *JAVASCRIPT*, *REACT.JS*, *NODE.JS*, *REDUX*, *WEBRTC*, *FFMPEG*, *SOCKET.IO*.

Об’єкт дослідження – процес одночасного проведення прямих трансляцій на кілька платформ.

Предмет дослідження – браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ.

Мета дослідження – розробка браузерного відео стрімінгового сервісу, який дозволить одночасно проводити прямі трансляції на кілька платформ.

Методи дослідження: аналіз аналогічних відео стрімінгових сервісів для одночасного проведення прямих трансляцій на кілька платформ, проектування та розробка браузерного сервісу.

Наукова новизна: вперше використано технології *WebRTC* та *Socket.IO* для забезпечення потокової передачі медіа потоку; використання фреймворку *FFmpeg* для потокової обробки мультимедіа.

Важливість даної розробки полягає у створенні браузерного сервісу, який дозволить досягти аудиторії на різних платформах, таких як *YouTube*, *Twitch*, *Facebook* і т.д., забезпечуючи більше охоплення глядачів. Це особливо важливо, оскільки різні аудиторії можуть користуватися різними платформами для перегляду трансляцій.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

WebRTC – Web Real Time Communications

FFMPEG – Fast Forward Moving Picture Experts Group

SDP – Session Description Protocol

ICE – Interactive Connectivity Establishment

NAT – Network Address Translation

STUN – Session Traversal Utilities for NAT

CORS – Cross Origin Resource Sharing

ВСТУП

За останнє десятиліття у світі з’явилося безліч відео стрімінгових платформ. Дані платформи дозволяють людям легко створювати трансляцію та ділитися нею зі світом. Проте для розширення своєї аудиторії необхідно вести трансляції на декількох платформах одразу. Щоб цього досягти, необхідно впровадити браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ.

Ключовою перевагою даного сервісу є його браузерний характер. Він працює безпосередньо в веб-браузері, не потребуючи додаткового програмного забезпечення чи встановлення спеціальних додатків. Це забезпечує широку сумісність та доступність для різних пристроїв та операційних систем. Крім того, браузерний стрімінговий сервіс спрощує процес запуску трансляцій, дозволяючи користувачам швидко та легко ділитися своїм контентом.

Однією з основних функцій браузерного відео стрімінгового сервісу є можливість одночасної трансляції на різні платформи, такі як *YouTube*, *Facebook*, *Twitch* та інші. Це вирішує проблему розсіяності аудиторії, дозволяючи створювачам залучати глядачів з різних сервісів в один момент часу. Такий підхід допомагає ефективно збільшувати покриття та розширювати глядацьку базу. Важливість даного браузерного сервісу полягає в:

– Розширення аудиторії. Браузерний відео стрімінговий сервіс, який підтримує одночасні прямі трансляції на кілька платформ, надає можливість досягти широкої аудиторії. За допомогою цієї функції користувач може одночасно входити на різні соціальні мережі чи інші онлайн-платформи, забезпечуючи широкий охоплення та залучення глядачів з різних джерел.
– Взаємодія з глядачами. Завдяки можливості проводити прямі трансляції на кілька платформ одночасно, ведучий отримує можливість взаємодіяти з різними аудиторіями одночасно. Це сприяє підвищенню кількості

коментарів, питань і реакцій від глядачів, що в свою чергу створює активну та залучену спільноту.

– Максимізація ефективності зусиль ведучого. Підтримка одночасних трансляцій на кілька платформ дозволяє ведучому ефективно використовувати свій час та зусилля. Замість того, щоб вести окремі трансляції для кожної платформи, ведучий може одночасно доставляти свої трансляції на всі свої канали, що сприяє оптимізації робочого процесу.
– Збільшення можливостей отримання прибутку. Можливість транслювати контент на різні платформи розширює можливості отримання прибутку для ведучого. Він може використовувати різні стратегії отримання прибутку, такі як реклама, пожертвування глядачів та спонсорські угоди на різних платформах, щоб забезпечити стабільний дохід.

Об’єктом дослідження є процес одночасного проведення прямих трансляцій на кілька платформ.

Предметом дослідження є браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ.

Метою даної кваліфікаційної роботи є розробка браузерного відео стрімінгового сервісу, який дозволить одночасно проводити прямі трансляції на кілька платформ.

Для досягнення поставленої мети необхідно:

1. Дослідити поняття відео трансляцій та його основні аспекти. Визначити ключові характеристики відео трансляцій, таких як роздільна здатність, частота кадрів, формат аудіо і відео даних, бітова швидкість тощо;

2. Розглянути та проаналізувати аналогічні сервіси. Провести аналіз функціональності, інтерфейсу та технічних аспектів подібних сервісів;

3. Дослідити основні технології розробки браузерних сервісів. Розглянути мови програмування, фреймворки, бібліотеки та інші технічні інструменти, які використовуються при розробці браузерних сервісів;

4. Дослідити технології передачі потокових даних. Розглянути різні протоколи передачі аудіо і відео даних, а також проаналізувати їх технічні характеристики та їхнє використання в різних сценаріях;

5. Розробити браузерний сервіс для одночасного проведення прямих відео трансляцій на кілька платформ.

6. Розробити структуру та інтерфейс браузерного сервісу.

7. Обрати технологій для забезпечення стабільної та якісної передачі відео на кілька платформ одночасно.

8. Визначити послідовність дій для створення з’єднання між клієнтом і сервером.

9. Організувати передачу медіа потоку від сервера на платформи.

10. Перевірити роботу створеного відео браузерного сервісу для одночасного проведення прямих трансляцій на кілька платформ.

Науковою новизною отриманих результатів є використання фреймворку *FFmpeg* для потокової обробки та передачі аудіо і відео даних на платформу, а також бібліотек *WebRTC* і *Socket.IO* для передачі медіа потоку між клієнтом і сервером. В порівнянні з аналогічними сервісами, дані технології було вперше використано для вирішення цих завдань.

Практичним значенням отриманих результатів є

Всі отримані результати в кваліфікаційній роботі отримані особисто здобувачем вищої освіти.

Теоретичні та практичні результати, які були отримані в процесі виконання кваліфікаційної роботи, були апробовані на науково-практичній конференції «Сучасні тенденції розвитку системного програмування», 23-24 листопада 2023 року.

Практичним значенням отриманих результатів є можливість транслювати відео одночасно на декількох платформах дозволяє розширити аудиторію і залучити глядачів з різних соціальних мереж або відео платформ.

Завдяки розробленому браузерного сервісу ведучі, компанії та бренди зможуть використовувати його для максимізації ефекту від рекламних компаній, оскільки їхні

трансляції будуть доступні для більшої кількості глядачів на різних платформах одночасно. Крім того, використання даного браузерного сервісу зменшить необхідність використовувати додаткове обладнання чи програми для кожної окремої платформи, тобто одна трансляція може бути використана для багатьох платформ, що зменшить трудомісткість та витрати. Таким чином, користувачі зможуть вибирати ті платформи, на які хочуть транслювати свої трансляції, в залежності від їх цільової аудиторії та стратегії маркетингу.

Одним з основних переваг розробленого браузерного сервісу є мультиплатформеність. Браузерний сервіс може працювати на різних операційних системах таких, як *Windows*, *macOS*, *Linux* та мобільних пристроях. Це робить його доступним для користувачів незалежно від їхнього пристрою чи операційної системи. Також даний браузерний сервіс можна використовувати без необхідності попереднього встановлення.

РОЗДІЛ 1
АНАЛІЗ АНАЛОГІЧНИХ СЕРВІСІВ

1.1. Поняття потокового мультимедіа

Потокове мультимедіа – це формат мультимедіа, який можна відтворювати за допомогою автономного або онлайн медіаплеєра. З технічної точки зору потік передається та приймається клієнтом безперервно, майже без проміжного зберігання на мережевих пристроях. Термін «потокова передача» вказує на метод доставки медіа, а не на саме медіа. Архітектуру потокового мультимедіа зображено на рис. 1.1.

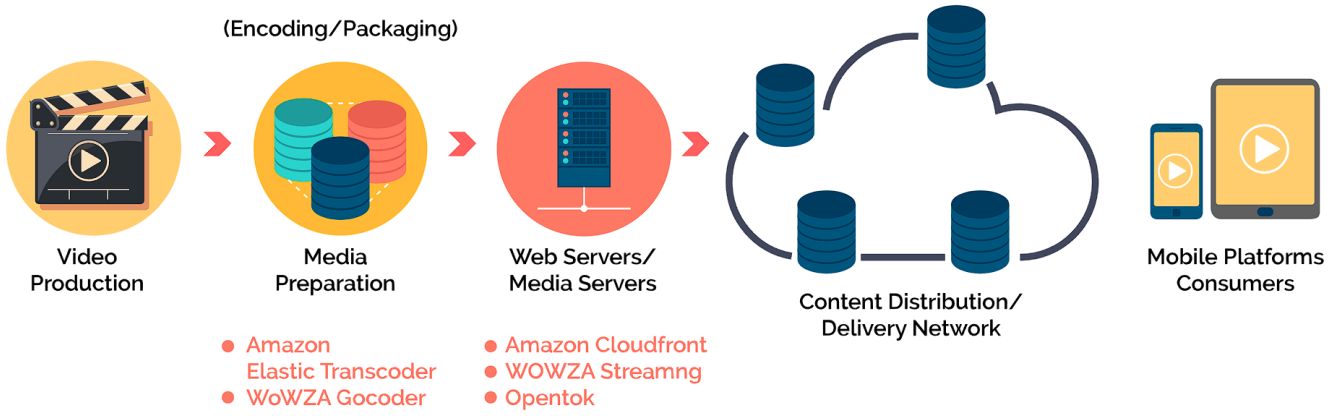

Рис. 1.1. Архітектура потокового мультимедіа

Різниця в методах доставки медіа безпосередньо впливає на телекомунікаційні мережі, оскільки більшість традиційних систем доставки мультимедіа є або потоковими (радіо, телебачення), або непотоковими (книги, відеокасети, аудіо компакт-диски). Проблеми з потоковою передачею контенту в Інтернеті включають в себе зупинки, затримки або погану буферизацію для користувачів з обмеженою пропускною здатністю мережі. Також існують виклики для користувачів, які не мають сумісного обладнання або програмних систем для потокового відтворення. Використання буферизації контенту перед відтворенням на декілька секунд може суттєво покращити якість відтворення.

Потокова передача представляє собою альтернативу завантаженню файлів, де кінцевий користувач отримує доступ до усього файлу із вмістом перед тим, як

розпочати його перегляд чи прослуховування. Завдяки цьому методу користувач може почати відтворення цифрового відео чи аудіо даних за допомогою свого медіа плеєра, навіть якщо весь файл ще не був повністю переданий. Термін «потокове мультимедіа» також охоплює медіа, що відрізняється від відео та аудіо, такі як прямі субтитри, бігуча стрічка та текст в реальному часі, які усі вважаються «потоковим текстом».

Відображення мультимедійних даних на комп'ютерах було досліджено ще з самого початку розвитку комп'ютерів в середині 20 століття. Проте протягом численних десятиліть прогрес був обмежений, в основному через високі витрати та обмежені можливості комп'ютерного обладнання. В кінці 1990-х років персональні комп'ютери для звичайних користувачів стали достатньо потужними для відтворення різноманітних медіа файлів. Технічні виклики, пов'язані з потоковою передачею, включають в себе необхідність наявності достатньої пропускної здатності процесора і шини для підтримки необхідних швидкостей передачі даних, досягнення обчислювальної продуктивності в реальному часі для запобігання недоповненого буфера і забезпечення плавної передачі вмісту.

Аналогічно до процесу завантаження файлів, потокове медіа розділяється на різні пакети даних. Потім ці пакети даних знову об'єднуються в оригінальний файл за допомогою програми, яку користувач використовує як потоковий медіаплеєр. Потокове відтворення медіафайлів є більш ефективним порівняно зі завантаженням медіафайлів, оскільки не потрібно зберігати копію цілого файлу на пристрої користувача перед початком відтворення. Замість цього копіюється лише невелика частина файлу, і ця частина видаляється для звільнення простору для наступної частини після її відтворення.

Незважаючи на це, для ефективного потокового відтворення високоякісних медіа-файлів потрібна велика пропускна здатність, або б мала велика пропускна здатність, якби не було кодеків. Кодеки складаються з двох ключових компонентів: кодера, який стискає дані перед відправленням їх з сервера на комп'ютер користувача, і декодера, який відновлює дані при їхньому отриманні. Кодеки поділяються на два типи: з втратами та без втрат. Кодеки з втратами видаляють зайву

інформацію для зменшення розміру передаваного файлу. Також вони можуть зменшувати кількість кольорів, роздільну здатність дисплея, частоту кадрів і видаляти звукові частоти поза діапазоном, який може сприймати більшість людей. Прикладами кодеків з втратами є *MP3* і *AAC* для аудіофайлів, а також *MPEG-2* для відеофайлів. Кодеки без втрат зберігають більше оригінального формату файлу, але при цьому розмір файлу збільшується. Декілька прикладів аудіокодеків без втрат включають *Apple Lossless Audio Codec* і *Dolby TrueHD*, а *MSU Lossless Video Codec* є прикладом відеокодека без втрат.

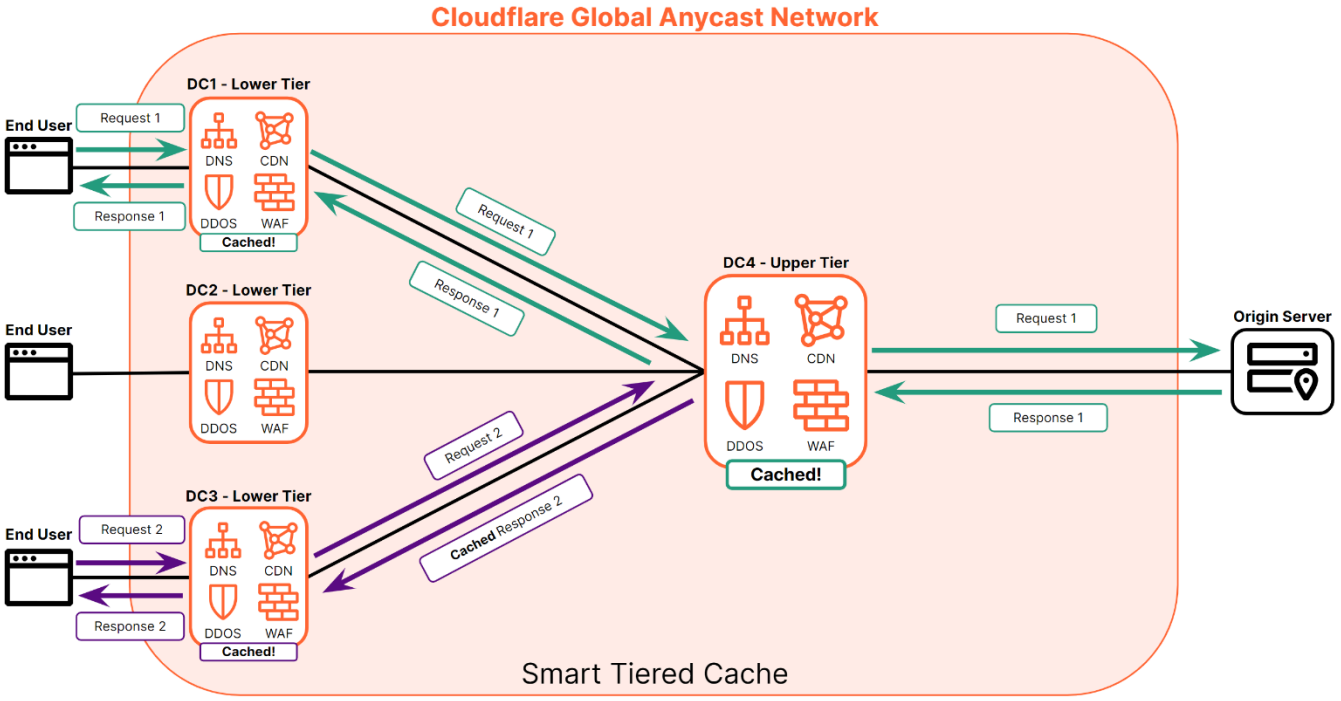

Рис. 1.2. *HTTP* запити та потік трафіку через *CDN*

Щоб уникнути переривань у відтворенні аудіо або відео через нестабільне підключення до Інтернету, потокові медіаплеєри завантажують файл заздалегідь на кілька секунд, що називається буферизацією. Великі компанії для потокового медіа також використовують мережі доставки контенту *Content Delivery Network* (рис. 1.2), де найпопулярніший контент зберігається на серверах, розташованих географічно ближче до місця трансляції. Це дозволяє подальше зменшення затримки мережі і зниження витрат на пропускну здатність. Крім того, компанії використовують *Adaptive Bitrate Streaming* (рис. 1.3), що базується на протоколі передачі гіпертексту *HTTP* і постійно адаптує швидкість потокового відтворення медіа до якості

Інтернет-з'єднання користувача та продуктивності комп'ютера. Ця функція стає особливо корисною під час подорожей, коли мобільна мережа глядача може перемикатися між високошвидкісними (*4G* або *5G*) та повільними (*3G*) мережами.

Adaptive Bitrate Streaming працює через *HTTP* та включає в себе кодування вихідного вмісту з різними швидкостями передачі даних [11]. Кожен потік бітової швидкості розбивається на невеликі частини тривалістю кілька секунд. Розмір цих сегментів може змінюватися в залежності від конкретної реалізації, але частіше всього становить від двох до десяти секунд. Якщо пропускна здатність мережі перевищує швидкість передачі даних, клієнт запитає сегмент із більшою швидкістю.

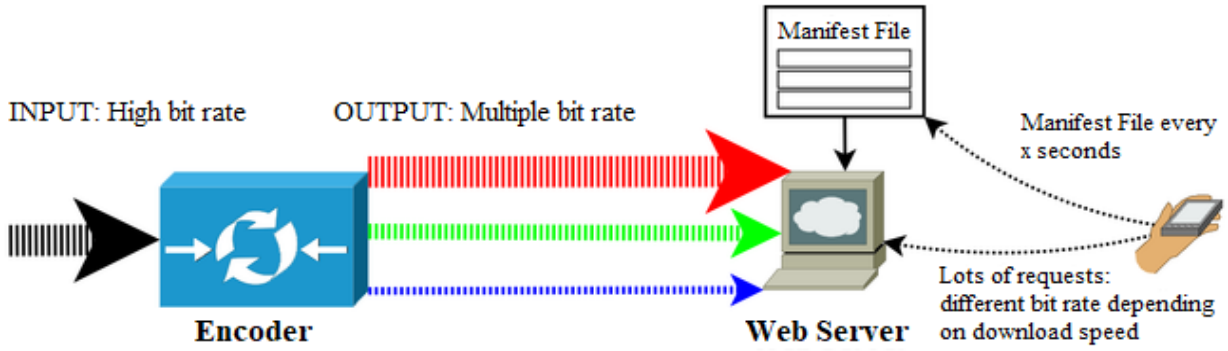

Рис. 1.3. Передача даних за технологією *Adaptive Bitrate Streaming*

У 2019 році розпочалися «війни потоків» – період високої конкуренції між потоковими відеокомпаніями на насиченому ринку. Ключовим подією став запуск *Disney+*, у листопаді того самого року, що викликав виклик *Netflix* та підштовхнув інші компанії приєднатися до боротьби. Конкуренти боролися не лише за право транслювати популярні телевізійні і кінематографічні твори, але й за ексклюзивний контент власного виробництва. Пандемія *COVID-19* ще більше прискорила «війни потоків», оскільки примусила людей залишатися вдома, де телевізійний перегляд став одним із небагатьох варіантів розваги. Сервіси боролися за свою частку в стрімко зростаючому числі абонентів.

Після того, як були скасовані обмеження, пов'язані з *COVID*, і кількість передплатників на кожну послугу стала приблизно однаковою, компанії, що надають потокове відео, перенаправили свою увагу з завоювання ринкової частки на максимізацію вартості вже наявних клієнтів. Стратегії для збільшення прибутку

включали об’єднання послуг, впровадження реклами та обмеження передачі паролів. Однак навіть з такими заходами ймовірно, що компанії ще більше консолідуватимуть або об’єднуватимуть свої потокові сервіси, оскільки деяким із них не вдається отримати достатній фінансовий потік від існуючих клієнтів. Наприклад, у 2023 році було оголошено, що *HBO Max* і *Discovery+* об’єднаються в один сервіс *Max*.

1.2. Сфери використання потокового мультимедіа

Поява потокового мультимедіа змінило уявлення про освіту, розваги та інформаційний обмін. Ця інноваційна форма передачі контенту відкрила безліч можливостей у різних сферах життя сучасного суспільства.

Послуги прямих трансляцій охоплюють широкий спектр тематики – від соціальних мереж до відеоігор, професійного спорту та освіти. Платформи, такі як *Facebook Live*, *Periscope*, *Kuaishou*, *Douyu* та *Bilibili* використовуються для трансляції різноманітних подій, включаючи заплановані рекламні акції та заходи знаменитостей, а також взаємні трансляції між користувачами, схожі на відеоконференції. Платформи, такі як *Twitch*, стали популярними для перегляду трансляцій ігор, таких як кіберспорт. Прямі трансляції спортивних подій є однією з найпоширеніших застосувань цих сервісів.

Останні роки відзначаються значним зростанням аудиторії та інвестицій у прямих трансляцій спортивних подій. У 2019 році цифровий потік на платформах, таких як *Prime Video*, *NFL Digital*, *Fox Sports Digital* та *Verizon Media Mobile*, зріс до понад 1 мільйона глядачів, що на 43% перевищило показники попереднього року (729000). Крім того, багато досліджень та прогнозів свідчать про зниження витрат споживачів на традиційні послуги платного телебачення до 74,5 мільярдів доларів у 2023 році. Прогнозується, що витрати на послуги передплати у США вперше перевищать витрати на платне телебачення у 2024 році. Корпорації великого масштабу, такі як *Amazon*, активно розширюють свою присутність у сфері прямих трансляцій спортивних подій. У 2021 році *Amazon* уклала 11-річну угоду на суму 113

мільярдів доларів щодо трансляції ігор Національної футбольної ліги на своїй платформі потокового відео *Amazon Prime*.

У медицині та науці потокове мультимедіа використовується для обміну дослідженнями та навчанням новим методам лікування. Відео операцій, онлайн-лекції від провідних фахівців та віртуальні лабораторії дозволяють розширити горизонти знань у цих важливих галузях.

У сфері освіти потокове мультимедіа дозволяє студентам та учням отримувати якісні знання відразу, не обмежуючи себе межами аудиторії чи розкладу. Відео-лекції, вебінари та інтерактивні уроки роблять навчання більш доступним і ефективним. Онлайн-платформи, такі як *Coursera* чи *Khan Academy*, забезпечують великий вибір курсів та матеріалів для самостійного вивчення.

Потокове медіа дає компаніям можливість залучити аудиторію ширше, ніж будь-коли раніше. У порівнянні з традиційними ЗМІ, компанії раніше стикалися з обмеженнями щодо географічних меж та обсягу своїх маркетингових бюджетів. Однак за допомогою потокового мультимедіа компанії можуть звертатися до мільйонів потенційних клієнтів по всьому світу, незалежно від їхнього місцезнаходження. Це особливо важливо для малих підприємств, які прагнуть розширити свою клієнтську базу, уникаючи зайвих витрат.

Гнучкість є однією з ключових переваг потокового медіа для бізнесу. Ви маєте можливість адаптувати свій контент відповідно до потреб вашої аудиторії, будь то створення спеціальних навчальних модулів чи адаптація повідомлень для різних відділів. Крім того, з можливістю потокової передачі на різні пристрої - від ноутбуків до смартфонів, ви можете забезпечити доступність вашого вмісту для всіх користувачів.

Пряма трансляція надає можливість підприємствам взаємодіяти з клієнтами в режимі реального часу, що створює більш особистий та взаємодіючий підхід. Цей вид трансляції можна використовувати для запуску продуктів, проведення сесій запитань і відповідей, або навіть для огляду внутрішніх процесів компанії. Такий спосіб взаємодії може сприяти створенню лояльної аудиторії та відновленню конкурентних позицій підприємства на ринку.

Класичні способи реклами, такі як телевізійна та друкована, можуть виявитися витратними, і часто складно визначити їхню ефективність. У відміну від цього, реклама в потокових медіа є більш доступною та спрямованою, що дозволяє компаніям досягати своїх цільових клієнтів за менші витрати.

Завдяки потоковому медіа компанії можуть підвищити визнання свого бренду і визначити себе як експертів у власній області. Розробка та розповсюдження високоякісного контенту дозволяє компаніям ставати провідними у своїх сферах і здобувати довіру від потенційних клієнтів. Це може призвести до зростання лояльності до бренду та збільшення кількості повторних угод.

Однією з ключових переваг потокового медіа є здатність відслідковувати дані та здійснювати аналітику. Ці дані дозволяють компаніям вимірювати ефективність своїх кампаній, отримувати інформацію про свою цільову аудиторію та приймати рішення на підставі об’єктивних даних. З цією інформацією компанії можуть налаштовувати свій контент і рекламні стратегії для ефективнішого привертання уваги аудиторії та досягнення своїх мет.

Отже, потокове медіа стає потужним інструментом, який компанії можуть використовувати для взаємодії зі своїми клієнтами, навчання персоналу, розповсюдження новин про компанію та проведення віртуальних заходів. З використанням можливостей відео та аудіо контенту ви можете підняти свій бізнес на новий рівень і випередити конкурентів.

1.3. Стрімінгові сервіси для одночасного проведення прямих трансляцій

Одночасне проведення прямих трансляцій представляє собою особливу форму передачі відео-потоку «один до багатьох», що включає паралельне розповсюдження прямого відео-потоку на різні цільові платформи. Під час багатопотоковості сигнал одночасно відправляється до кількох точок призначення. Ці точки можуть включати соціальні мережі, такі як *Facebook*, *Twitter*, *Twitch* або *YouTube*, а також веб-сайти, які підтримують протокол *RTMP*.

RTMP (*Real Time Messaging Protocol*) – це протокол передачі даних у реальному часі через мережу Інтернет (рис. 1.4). Він був розроблений компанією *Adobe Systems* для використання у системах потокового відео та аудіо. *RTMP* часто використовується для трансляції відео та аудіо в режимі реального часу через Інтернет. Протокол дозволяє вам передавати дані в режимі реального часу з високою швидкістю передачі та низькою затримкою. Основною областю застосування *RTMP* є стрімінгове відео, відео чати, онлайн-трансляції та інші сценарії, де необхідно миттєве відтворення медіа даних.

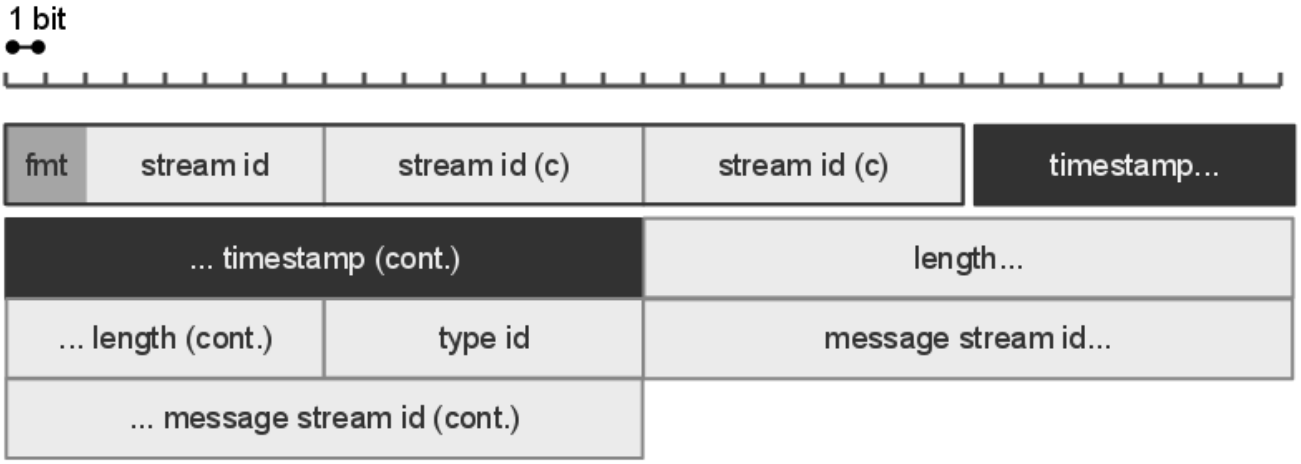

Рис. 1.4. Діаграма пакета протоколу *RTMP*

Можливість проведення потокової передачі залежить від програмного забезпечення, яке ви використовуєте для багатопотокової передачі. Деякі програми пропонують обмежену кількість напрямків, тоді як інші мають необмежені можливості. В більшості випадків провайдери, які підтримують обмежену кількість точок призначення, є більш економічними, ніж ті, що дозволяють досягати безлічі кінцевих точок призначення.

Для багатьох ведучих вибір платформи для трансляцій не є проблемою. Однак те, що обмежує їх, – це здатність транслювати в багато місць одночасно. Якщо користувач має можливість транслювати відео в режимі реального часу лише на одній платформі одночасно, йому треба зробити вибір: або витрачати значний час, або не мати змоги захопити всю аудиторію максимально привабливим способом.

Існує три шляхи для одночасного проведення прямих трансляцій:

1) Апаратний кодувальник із функцією багатопотокової передачі. Для завантаження відео має бути достатня пропускна спроможність інтернет з'єднання. І кожному потоку потрібен кодувальник, яких може бути досить багато.

2) Програмне забезпечення для настільних комп'ютерів. Тут також має бути доступна відповідна смуга пропускання та апаратна потужність. Користувач повинен мати певний досвід роботи з технологією для підключення відповідного обладнання до програмного забезпечення настільного комп'ютера.

3) Хмарне програмне забезпечення. Хмарне ПЗ надає потрібну інфраструктуру. Дані програми зазвичай дуже інтуїтивно зрозумілі, і потоки можна розподіляти за допомогою кількох операцій.

Використання кодувальника може бути простим або складним в залежності від його типу. Апаратні кодувальники вважаються більш надійними, але вони також можуть бути вартісними та складними в управлінні. Вартість та складність є компромісом у випадку багатопотокових апаратних кодувальників, які можуть коштувати від кількох тисяч гривень до десятків тисяч.

Програмні кодувальники, такі як *OBS*, *Wirecast* або *vMix*, часто підтримують багатопотоковість, але вони можуть вимагати додаткових налаштувань або складних процесів установки. Якщо користувач вже користуєтеся кодувальником для прямої трансляції і прагне до багатопотокового відтворення, важливо мати на увазі проблеми, з якими він можете стикнутися.

Перш за все, користувачу слід переконатися, що мережа має достатньо пропускної здатності для одночасного відсилання декількох потоків. В іншому випадку відео для аудиторії може бути низької якості або трансляція може перериватися. Якщо користувач використовує програмний кодувальник, також важливо перевірити, чи здатний комп'ютер обробляти кодування кількох потоків без втрати продуктивності.

Завдяки можливості використання багатопотокового режиму ви економите час і не хвилюєтеся про необхідну пропускну здатність для завантаження відео в

реальному часі. Інші важливі аспекти використання багатопотокового режиму включають:

- Не потрібно примушувати аудиторію обирати конкретну платформу. Замість того, щоб переконувати глядачів обрати один конкретний ресурс, ведучий можете зустрітися зі своєю аудиторією там, де вона знаходиться. Це підвищує відкритість і дає можливість транслювати одночасно на різних платформах. Це привертаєте свою основну аудиторію, розширюючи при цьому охоплення нових слухачів.
- Трансляції стають більш надійними. Якщо трансляція з якихось причин припиняється на одній платформі, вона продовжує транслювати на інших. Таким чином, ведучий не втрачає всю працю, яку вкладає у свої потоки.
- Розширюється охоплення без значних витрат. Багатопотоковий режим не вимагає додаткового часу або грошей. Ведучий може використовувати ту саму камеру, мікрофон і підключення до інтернету для одного потоку, але її одночасно зможуть спостерігати на різних платформах.

1.4. Аналіз існуючих стрімінгових сервісів

1.4.1. *YouTube*

YouTube – це великий відеохостинг та соціальна мережа, яка надає користувачам можливість завантажувати, переглядати та обмінюватися відео контентом (рис. 1.5). Платформа була створена трьома колишніми співробітниками компанії *PayPal* – Чедом Харлі та Стівом Ченом, які запустили сайт у лютому 2005 року. У жовтні 2006 року компанія була придбана *Google* за 1,65 мільярда доларів.

YouTube став не тільки однією з найпопулярніших платформ для спільного перегляду та завантаження відео, а й значущим культурним явищем. Платформа пропонує широкий спектр відео контенту, включаючи від розважального та освітнього до новин та спорту. Архітектуру платформи *YouTube* показано на рисунку 1.6.

Користувачі можуть створювати власні канали, де вони публікують відео на різні теми. Це дозволяє створити унікальні спільноти та залучити аудиторію з усього світу. Велика частина відео-контенту на *YouTube* є безкоштовною для перегляду, але є також можливість підписки на платний контент або пожертви від глядачів через систему «*Super Chat*».

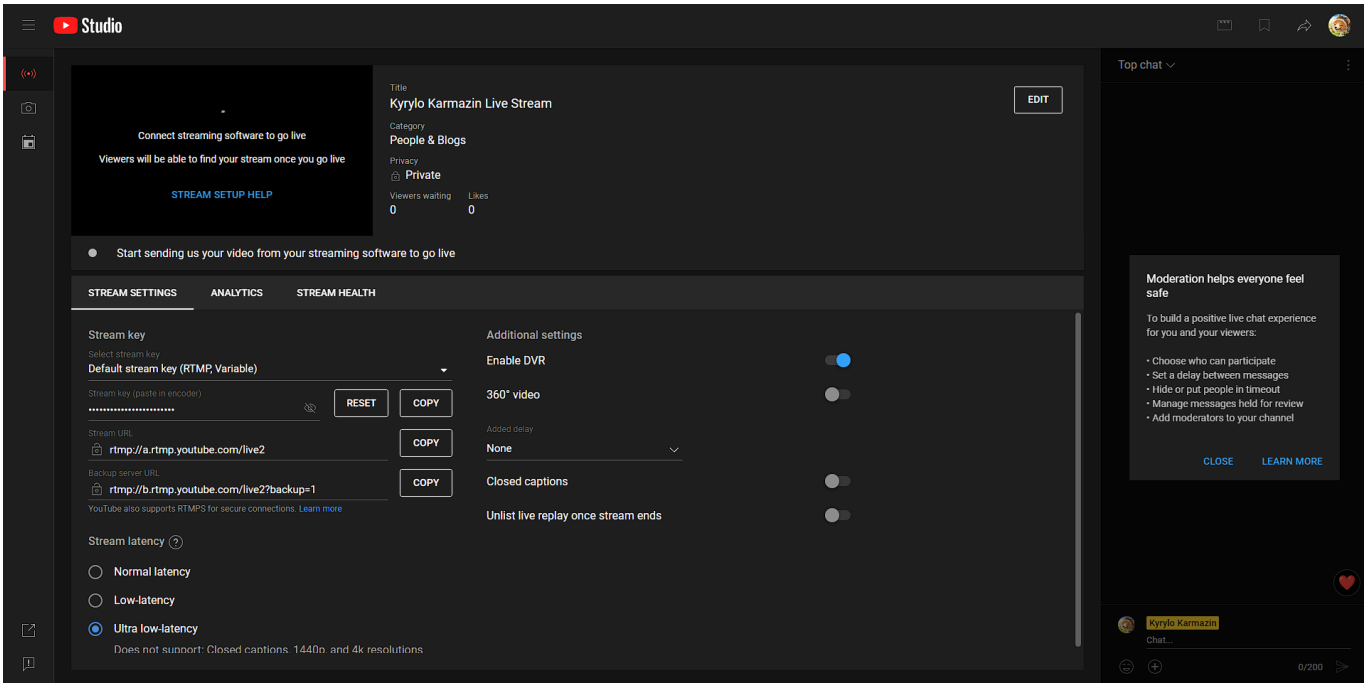

Рис. 1.5. Вікно налаштувань трансляції *YouTube*

Крім того, *YouTube* надає можливість монетизації для творчих людей, які можуть заробляти гроші на своїх відео через рекламу, партнерські програми та інші способи. Це зробило *YouTube* важливим засобом для заробітку для багатьох відео-блогерів та творчих осіб.

Основні переваги *YouTube*:

1) Доступність та зручність використання. Платформа доступна на різних пристроях, включаючи комп’ютери, смартфони та телевізори, що робить її легкою у використанні та доступною для широкого кола користувачів.

2) Можливість монетизації. *YouTube* дозволяє творцям заробляти гроші на своїх відео через різні програми монетизації, такі як реклама, партнерські програми, пожертви глядачів і платний контент.

3) Великий обсяг контенту. *YouTube* має вражаючий обсяг відео контенту на різні теми, від розважального до освітнього та професійного. Користувачі можуть знайти відео практично на будь-яку тему, що робить платформу дуже різноманітною та цікавою для широкого кола аудиторії.

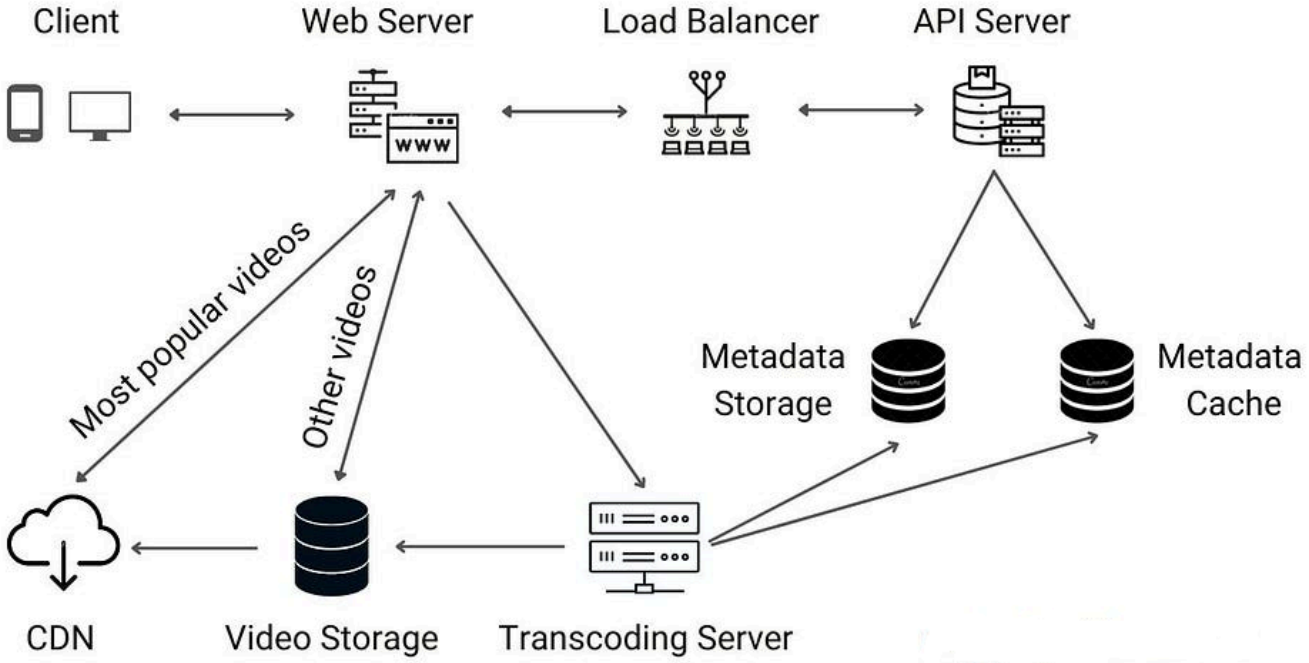

Рис. 1.6. Архітектура платформи *YouTube*

Основні недоліки:

1) Проблеми з авторськими правами. Платформа часто стикається з проблемами щодо порушень авторських прав, особливо в контексті завантаження та використання музики та інших матеріалів без відповідних дозволів.

2) Алгоритмічний ефект «фільтру бульбашки». Алгоритми *YouTube*, спрямовані на підвищення залученості глядачів, можуть призводити до того, що користувачі отримують обмежений спектр контенту, що підтримує їхні власні переконання, утворюючи так званий «фільтр бульбашки».

3) Залежність від алгоритмів. Зміни в алгоритмах *YouTube* можуть впливати на видимість та доходи творців контенту, що робить їх вразливими перед можливими змінами в політиці платформи.

1.4.2. *Twitch*

Twitch – це онлайн-платформа для стрімінгу відео-контенту, спеціалізована на геймінгу, але також включає в себе широкий спектр тематики, таких як музика, мистецтво, спорт, техніка та інше (рис. 1.7). Заснований у 2011 році, *Twitch* здобув величезну популярність і став однією з провідних платформ для стрімінгу у світі.

Однією з головних особливостей *Twitch* є можливість живого взаємодії між ведучими та глядачами через чат. Глядачі можуть надсилати повідомлення, задавати питання, а також взаємодіяти з іншими учасниками спільноти. Це створює враження участі та спільноти навколо вмісту.

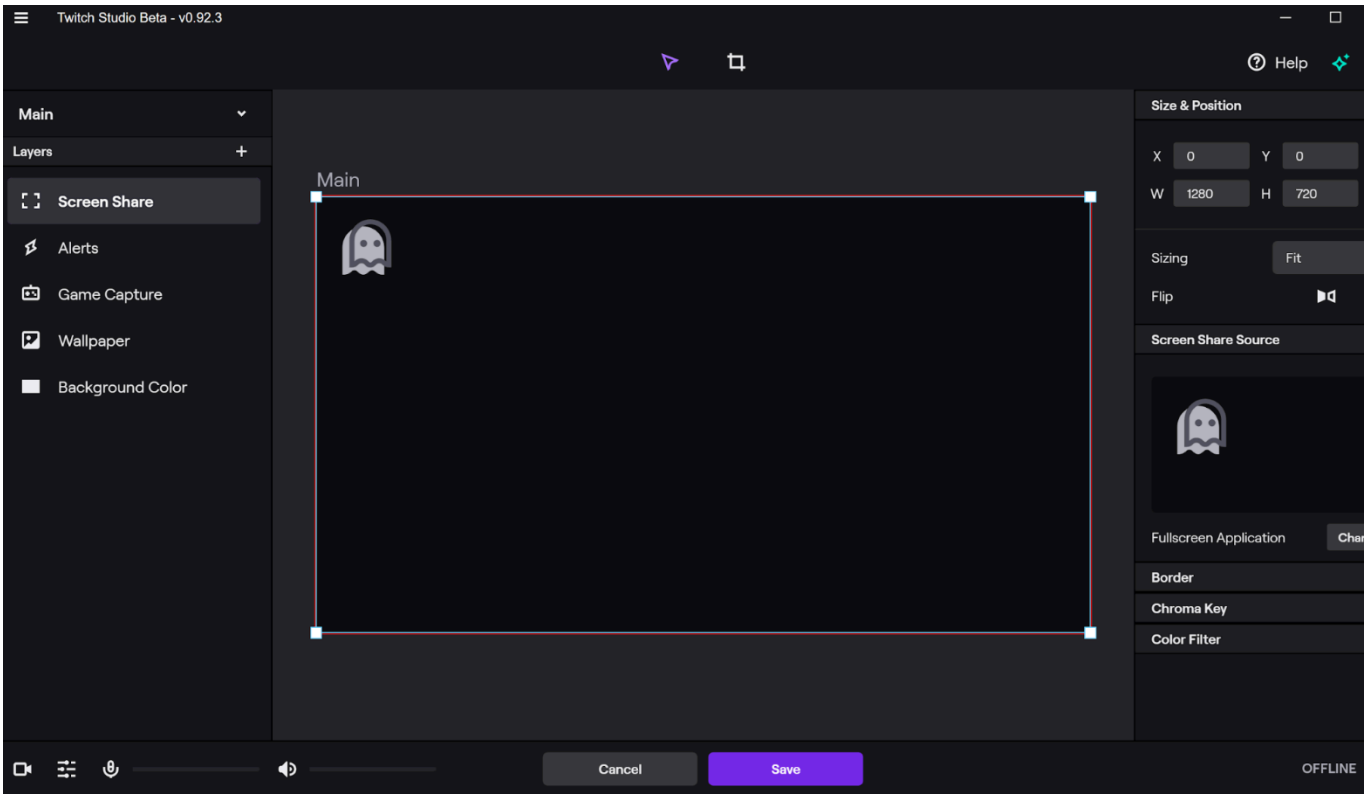

Рис. 1.7. Панель керування трансляцією *Twitch*

Платформа також надає можливість ведучим отримувати доход через різні шляхи, такі як пожертвування глядачів, рекламні партнерства та підписки. Глядачі можуть підтримувати своїх улюблених ведучих, вносячи фінансові внески або підписуючись на платний контент.

Головною аудиторією *Twitch* є геймери, які транслюють свою геймплей-сесію або проводять розважальні стріми. Зокрема, платформа дозволяє транслювати гру в реальному часі, коментувати ігрові моменти, а також спілкуватися з глядачами. Крім того, на *Twitch* можна знайти велику кількість контенту, пов'язаного із спортом, музикою, кулінарією, науково-популярними відео та іншими темами.

Основні переваги *Twitch*:

1) Широкий спектр контенту. *Twitch* пропонує різноманітний контент, починаючи від геймінгу і закінчуючи музикою, творчістю, спортом та іншими темами. Це робить платформу привабливою для різних аудиторій.

2) Інтерактивність. Однією з ключових особливостей *Twitch* є можливість живого взаємодії між ведучими та глядачами через чат. Це створює враження спільноти та реальної участі.

3) Монетизація. Ведучі можуть заробляти гроші на платформі через пожертвування глядачів, рекламні партнерства та платні підписки. Це робить *Twitch* привабливим для тих, хто хоче отримувати дохід на власному контенті.

Основні недоліки:

1) Конкуренція і насиченість ринку. З великою кількістю ведучих на платформі конкуренція може бути дуже високою, що ускладнює завоювання уваги та аудиторії.

2) Залежність від алгоритмів. Політика алгоритмів рекомендацій може впливати на те, який контент отримує більше уваги. Це може бути викликом для новачків або тих, хто не вписується в стандартні шаблони популярності.

3) Проблеми з цензурою. Платформа часто стикається з критикою у зв'язку з рішеннями про цензуру та перевіркою контенту, що може викликати суперечки щодо свободи слова.

1.4.3. *Restream*

Restream – це популярний стрімінговий сервіс, який надає користувачам можливість одночасно транслювати відео на кілька платформ, таких як *YouTube*, *Facebook*, *Twitch*, *Twitter* і багато інших (рис. 1.8). Цей сервіс став важливим

інструментом для ведучих, брендів, компаній та інших творців, які хочуть максимізувати своє покриття та взаємодію з аудиторією.

Однією з ключових переваг *Restream* є можливість трансляції відео на кілька платформ одночасно. Це дозволяє ведучим ефективно залучати аудиторію з різних джерел, забезпечуючи більше можливостей для зростання популярності та отримання нових підписників. Крім того, *Restream* надає користувачам зручний інтерфейс для керування та моніторингу трансляцій на різних платформах в реальному часі.

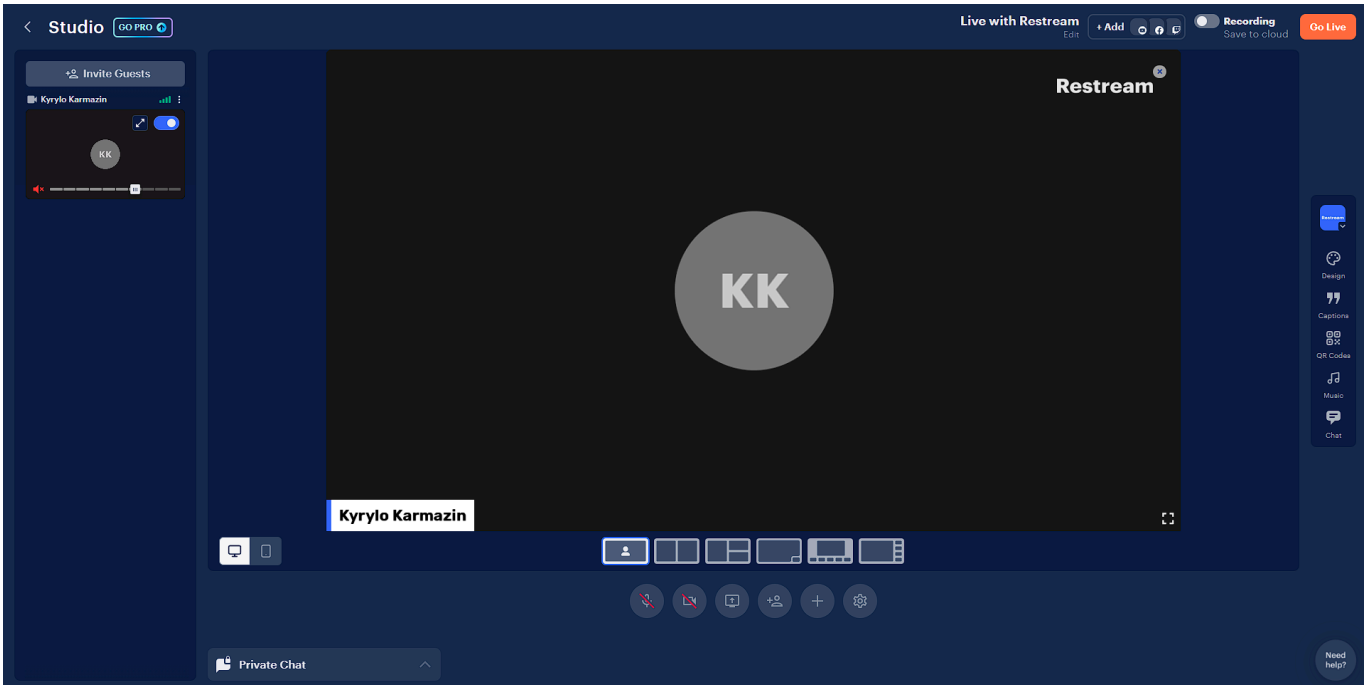

Рис. 1.8. Вікно конфігурацій трансляції *Restream*

Іншою важливою функцією є можливість налаштовувати відео та аудіо параметри для кожної платформи окремо. Це дозволяє користувачам оптимізувати якість стріму для кожної аудиторії та платформи, що призводить до кращого враження для глядачів і збільшення репутації ведучого або бренду.

Також слід зазначити, що *Restream* надає ряд аналітичних інструментів, які допомагають користувачам відстежувати показники ефективності трансляцій на різних платформах. Це включає в себе інформацію про кількість переглядів, вподобань, коментарів і підписників для кожного стріму, що робить процес аналізу результатів більш простим та ефективним.

Крім того, *Restream* підтримує інтеграції з різними сторонніми сервісами, такими як *OBS Studio*, *XSplit*, а також з можливість використання власного відео плеєра для трансляцій. Це робить його універсальним і зручним використовуваним інструментом для широкого спектру користувачів.

Основні переваги *Restream*:

1) Статистика трансляцій. *Restream* надає користувачам статистику по кожній з платформ, що може бути корисно для аналізу ефективності.

2) Індивідуальне налаштування параметрів для кожної платформи. Користувачі можуть оптимізувати якість відео та аудіо для кожної аудиторії, що робить *Restream* гнучким інструментом для різних цільових груп.

3) Інтеграції з іншими сервісами. Підтримка сторонніх інструментів, таких як *OBS Studio* та *XSplit*, робить використання *Restream* зручним і сумісним з різними програмами.

Основні недоліки:

1) Вартість послуги: Хоча *Restream* пропонує безкоштовний план, для повноцінного використання всіх можливостей, користувачеві може знадобитися сплачувати місячні внески, що може вплинути на бюджет.

2) Якість залежить від інтернет-з'єднання. Як і в будь-якому іншому стрімінговому сервісі, якість трансляції *Restream* може залежати від стабільності та швидкості інтернет-з'єднання користувача.

3) Затримка в трансляції. *Restream* може призводити до деякої затримки в трансляції, що може бути проблемою для тих, хто веде взаємодію з глядачами у реальному часі.

1.4.4. *Streamlabs*

Streamlabs – це комплексне програмне забезпечення, спеціально розроблене для ведучих та творців контенту, щоб забезпечити їм широкі можливості у створенні та трансляції відео контенту в режимі реального часу (рис. 1.9). Це комплексний

інструмент, який об’єднує в собі різноманітні функції для покращення якості та ефективності онлайн трансляцій.

Однією з ключових особливостей *Streamlabs* є його легкість використання. Програма пропонує інтуїтивний інтерфейс, який дозволяє користувачам швидко орієнтуватися та налаштовувати свої трансляції без надмірних зусиль. Інтегровані інструменти для стрімінгу, такі як вікна для чату, сповіщення про пожертви та відеоекран, роблять процес взаємодії з аудиторією більш захоплюючим та ефективним.

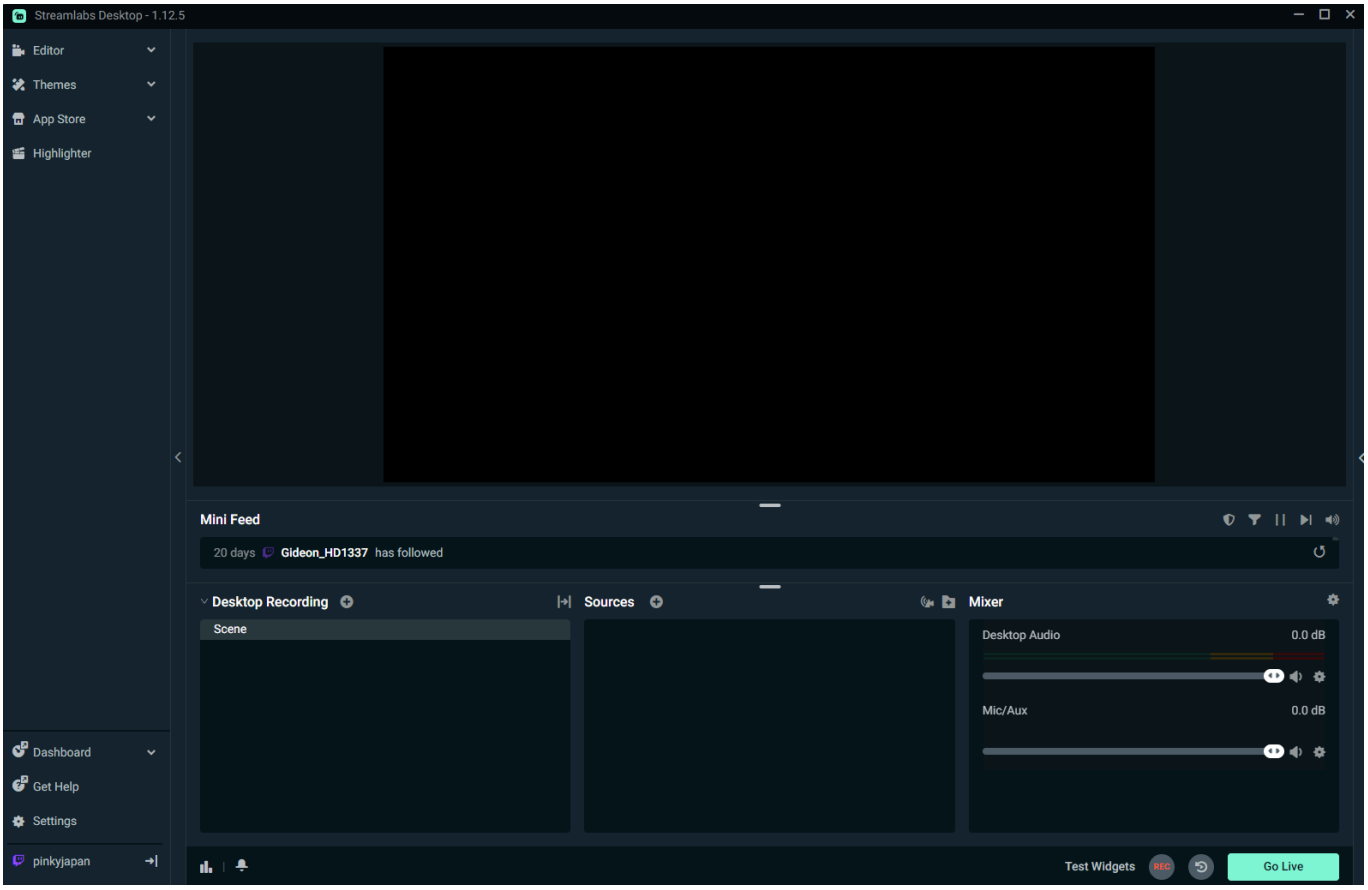

Рис. 1.9. Панель налаштувань трансляції *Streamlabs*

Крім того, *Streamlabs* надає розширені можливості для персоналізації контенту. Користувачі можуть створювати власні графічні ефекти, налаштовувати анімації та додавати унікальні елементи до своїх трансляцій, що допомагає їм виділятися серед інших ведучих.

Основні переваги *Streamlabs*:

1) Інтеграція з популярними платформами. *Streamlabs* легко інтегрується з такими популярними платформами для стрімінгу, як *Twitch*, *YouTube* і *Facebook*, надаючи користувачам можливість розповсюджувати свій контент на різних платформах одночасно або вибирати конкретні платформи за їх вибором.

2) Персоналізація контенту. *Streamlabs* надає розширені можливості для персоналізації трансляцій, дозволяючи користувачам створювати власні графічні ефекти, анімації та додавати унікальні елементи до своїх стрімів.

3) Широкий вибір додатків і розширень. Платформа підтримує різноманітні додатки та розширення, які дозволяють ведучим розширити функціональність своїх трансляцій.

Основні недоліки:

1) Великі системні вимоги. *Streamlabs* вимагає значних ресурсів комп'ютера, що може вплинути на продуктивність, особливо на менш потужних пристроях.

2) Конфлікти з іншими програмами. Іноді можуть виникати проблеми з сумісністю *Streamlabs* з іншими програмами або іграми, що може призвести до труднощів у використанні.

3) Платна підписка на більшість функції. Хоча основні функції *Streamlabs* доступні безкоштовно, деякі розширені можливості вимагають платної підписки, що впливає на доступність для деяких користувачів.

1.5. Висновки до розділу

У першому розділі було досліджено поняття та архітектуру потокового мультимедіа та його впливу на різноманітні сфери. Розглядаючи поняття мультимедіа, було визначено його ключові складові, такі як текст, зображення, звук і відео, які об'єднуються для створення комплексного інформаційного змісту. Ця інтеграція різноманітних медійних форматів розширює можливості сприйняття та сприйняття інформації користувачами. Розглядаючи сфери використання потокового мультимедіа, було виявлено його великий потенціал у різноманітних галузях, починаючи від розваг і завершуючи освітніми та корпоративними заходами.

Особливу увагу було приділено сферам використання потокового мультимедіа, що є невід'ємною складовою сучасного цифрового середовища. Зокрема, висвітлено значення стрімінгових сервісів у різних контекстах, починаючи від розваг та ігор закінчуючи освітою та бізнесом. Використання стрімінгових сервісів стає все більш популярним, надаючи можливість не тільки отримувати інформацію, але й активно взаємодіяти з нею у режимі реального часу.

Було досліджено, що стрімінгові сервіси стали не просто джерелом розваг та інформації, а й потужним інструментом для взаємодії та спілкування у віртуальному просторі. Вони відкривають нові можливості для творчості, освіти та бізнесу, одночасно постійно еволюціонуючи та вдосконалюючи свої можливості.

Суттєвим аспектом вивчення стрімінгових сервісів стало розглядання їхнього впливу на формування спільнот та культури онлайн-спілкування. Розмаїття інструментів та можливостей цих сервісів дозволяє користувачам активно взаємодіяти, обмінюватися думками та створювати унікальний досвід. Однак, разом з тим, існують проблеми, такі як відсутність ефективного контролю над трансляціями та питання конфіденційності.

Одним з ключових аспектів потокового мультимедіа є ефективне кодування та стиснення даних. Це дозволяє зменшити обсяг даних, які передаються, що є критичним для забезпечення плавної передачі контенту при обмежених швидкостях інтернет-з'єднання. Також ключовим аспектом є використання відповідних протоколів. *HTTP Live Streaming* та *Dynamic Adaptive Streaming over HTTP* є популярними протоколами, які дозволяють адаптувати якість трансляції в залежності від швидкості інтернет-з'єднання користувача.

Було виявлено, що ефективність передачі потокового мультимедіа визначається серверною інфраструктурою. Великі платформи використовують *Content Delivery Network* для розподілу контенту по всьому світу, забезпечуючи швидкий доступ для користувачів у різних регіонах.

Також було проаналізовано поняття стрімінгових сервісів, ідентифіковані ключові принципи їх реалізації та визначено різноманітні шляхи забезпечення високоякісних трансляцій. Особлива увага була приділена таким сервісам, як

YouTube, *Twitch*, *Restream* та *Streamlabs*, які є найбільш популярними платформами у сфері потокового мультимедіа. *YouTube* відзначається великою аудиторією та інтеграцією з іншими сервісами *Google*, *Twitch* спеціалізується на розважальному контенті, *Restream* дозволяє транслювати на кілька платформ одночасно, а *Streamlabs* надає розширені інструменти для ведучих. На основі проаналізованої інформації можна зробити висновок, що вони надають користувачам широкий спектр інструментів для одночасного проведення прямих трансляцій.

Під час аналізу існуючих стрімінгових сервісів було ретельно розглянуто їх можливості, переваги та недоліки. Це надало можливість визначити оптимальний вибір функцій та технологій для проведення трансляцій на декілька платформ, враховуючи потреби та вимоги користувачів.

РОЗДІЛ 2
ПРОЕКТУВАННЯ БРАУЗЕРНОГО СЕРВІСУ

2.1. Технології створення браузерних сервісів

2.1.1. *React.js*

React.js є відкритою бібліотекою для створення користувацьких інтерфейсів. Розробляється і підтримується компанією *Facebook*. *React.js* використовується для побудови високопродуктивних веб-додатків з динамічною інтерактивністю [7].

Основними властивостями *React.js* є:

1) Компоненти. *React.js* базується на концепції компонентів – невеликих, незалежних блоків, які можуть бути легко перевикористані. Кожен компонент відповідає за свою власну логіку та візуальний вигляд, спрощуючи розробку та підтримку коду.

2) Віртуальний *DOM* (*Document Object Model*). *React.js* використовує віртуальний *DOM* для ефективного оновлення інтерфейсу. Замість безпосередньої маніпуляції реальним *DOM*, *React.js* спочатку оновлює віртуальний *DOM*, а потім визначає мінімальні зміни, які потрібно зробити в реальному *DOM* (рис. 2.1).

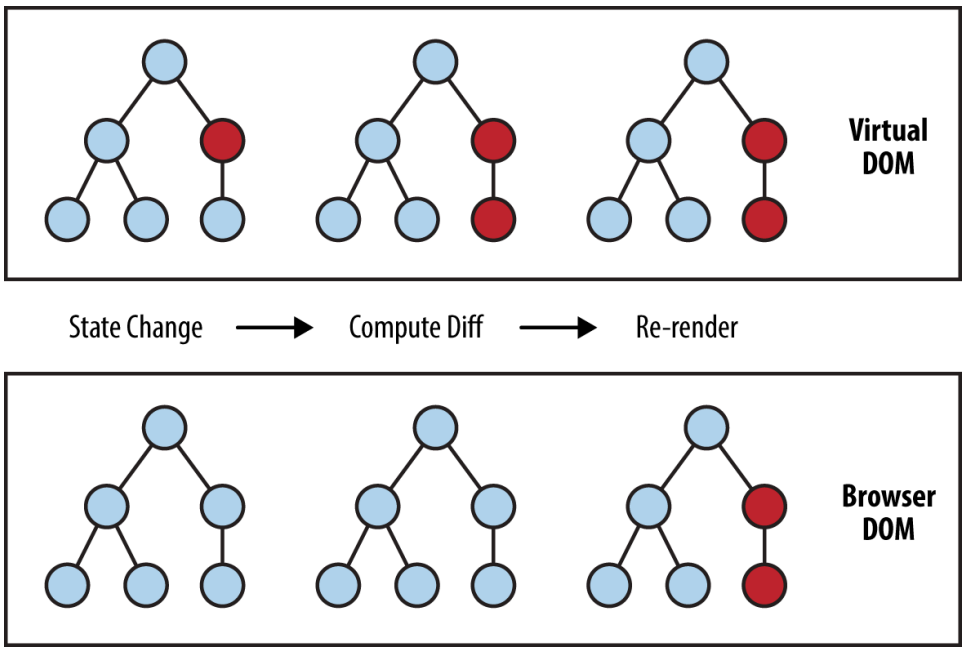

Рис. 2.1. Процес оновлення *DOM* в *React.js*

3) *JSX* (*JavaScript Syntax Extension*). Ще однією ключовою особливістю *React.js* є *JSX*, який дозволяє писати *HTML*-подібний код прямо в *JavaScript* файлі (рис. 2.2). Це полегшує розробку та читання коду, роблячи його більш ефективним та зрозумілим [14].

4) Реакції на події. *React.js* дозволяє обробляти події в інтерфейсі, такі як натискання кнопок чи введення тексту. Обробники подій можуть бути додані до компонентів, забезпечуючи динамічну взаємодію з користувачем.

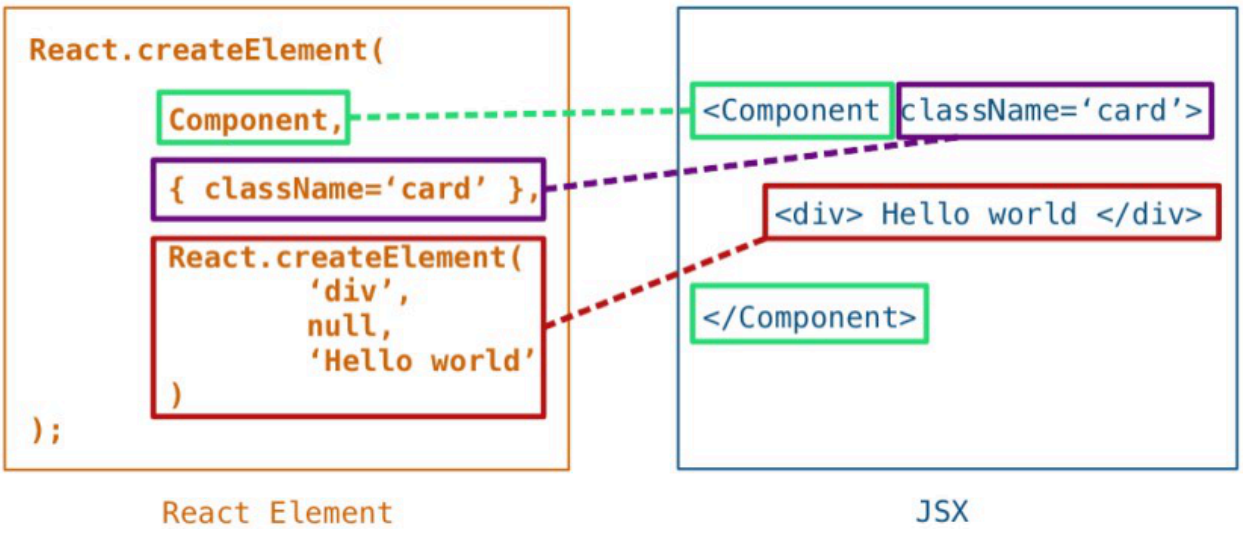

Рис. 2.2. Порівняння елементу *React* і *JSX*

Завдяки віртуальному *DOM* та іншим оптимізаціям, *React.js* дозволяє побудувати додатки, які працюють ефективно та швидко, навіть при великій кількості даних та складних інтерфейсах.

React.js ідеально підходить для створення односторінкових додатків (*SPA*), де весь контент завантажується один раз, і подальша навігація відбувається без повторних запитів на сервер. Також *React.js* легко інтегрується з іншими бібліотеками та фреймворками, що робить його гнучким і пристосованим до різних технологій.

Компоненти у *React* виступають як будівельні блоки програми, подібні до функцій *JavaScript*. Вони отримують різні параметри (*Props*) і повертають *React*-елементи, які описують відображення на екрані.

Важливо розуміти, що усе на екрані *React*-програми складається з компонентів. Додаток *React* можна уявити як сукупність компонентів, що вкладаються один в одного. Таким чином, розробники не створюють сторінки у *React*, а складають їх з компонентів.

Використання компонентів дозволяє розбити інтерфейс користувача на самостійні частини, які можна використовувати повторно. Спрощений спосіб визначення компонента – це написати функцію на *JavaScript*:

```
function Component(props) {
  return <h1>This is {props.name}</h1>;
}
```

Ця функція представляє собою дійсний компонент *React*, оскільки вона отримує єдиний аргумент у вигляді об'єкта *prop* з даними і повертає елемент *React*. Ці компоненти часто називаються «функціональними компонентами», оскільки вони фактично є *JavaScript*-функціями.

Окрім функціональних компонентів, існує інший тип компонентів – «класові компоненти». Класовий компонент відрізняється від функціонального тим, що визначається як клас *ES6*, як показано нижче:

```
class Component extends React.Component {
  render() {
    return <h1>This is {this.props.name}</h1>;
}}
```

Props представляють собою скорочену форму властивостей і служать для посилання на внутрішні дані компонента в *React*. Їх визначають всередині викликів компонентів та передають у компоненти (рис. 2.3). Синтаксис використання *props* також відповідає синтаксису атрибутів *HTML*. Є дві важливі риси в *props*: 1) визначення та використання їх значення відбувається до створення компонента; 2) значення *props* залишаються незмінними, тобто їх можна лише читати після передачі в компоненти.

Розробник може отримати доступ до властивості, посилаючись на неї через властивість «*this.props*», яка є доступною для кожного компонента.

Стан – це об’єкт, який містить певну інформацію, що може зазнавати змін під час життя компонента. Це означає, що він представляє лише поточний стан даних, збережених у *props* компонента. Оскільки дані можуть еволюціонувати з часом, необхідно використовувати методи управління для зміни даних, забезпечуючи тим самим відображення компонента у вигляді, який визначає розробник у конкретний момент [16].

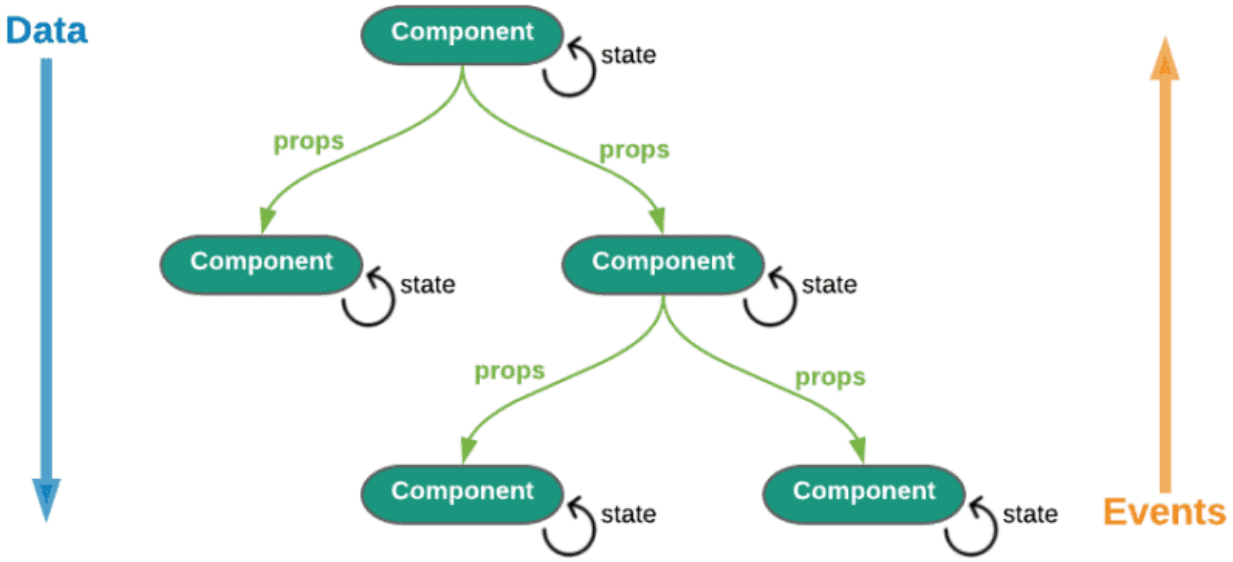

Рис. 2.3. Напрямок руху даних в компонентах *React*

У *React* стан можна також відслідковувати глобально, і дані можна обмінювати між компонентами за необхідності. В основі цього підходу лежить ідея того, що завантаження даних у нових місцях у програмах *React* не є таким витратним, як з іншими технологіями. Додатки *React* виявляються найефективнішими порівняно з аналогічними, особливо в контексті того, як вони зберігають і обробляють дані. Це відкриває нові можливості для створення інтерфейсів, які інноваційно використовують дані.

Ефективніше розробляти всю систему, уникаючи використання стану, і замість цього користуватися атрибутами та подіями. Це полегшує процес обслуговування, тестування та розуміння компонентів. Якщо потрібно використовувати стан, його слід додавати через контейнери стану, такі як *Redux* і *MobX*, або через компоненти контейнера (обгортки). *Redux* є популярною системою управління станом для інших

реактивних фреймворків, яка реалізує централізовану станову машину, керовану діями.

Кожен компонент *React* пройде три етапи: монтаж, візуалізація та демонтаж. Послідовність подій, які відбуваються на цих етапах, називається життєвим циклом компонента. Хоча ці події частково пов'язані із станом компонента (його внутрішніми даними), життєвий цикл має свої особливості. *React* володіє внутрішнім кодом, який завантажує та вивантажує компоненти за необхідності, і компонент може існувати на кількох етапах використання в цьому внутрішньому коді.

2.1.2. *Node.js*

Node.js – це відкрите середовище виконання *JavaScript*, побудоване на двигуні *V8* від компанії *Google Chrome*. Вперше випущений в 2009 році Райаном Далем та іншими розробниками. *Node.js* став популярним інструментом для розробки серверних застосунків та веб-додатків.

Node.js працює подійно-орієнтованим та неблокуючим способом, що означає, що він ефективно обробляє багато операцій одночасно без блокування виконання коду [8]. Його асинхронність робить його ідеальним для обробки багатьох одночасних підключень. Архітектуру *Node.js* зображено на рисунку 2.4.

Основною мовою програмування в *Node.js* є *JavaScript*. Використовуючи *Node.js*, розробники можуть використовувати *JavaScript* для створення серверної частини веб-додатків, що раніше було можливим лише на стороні клієнта.

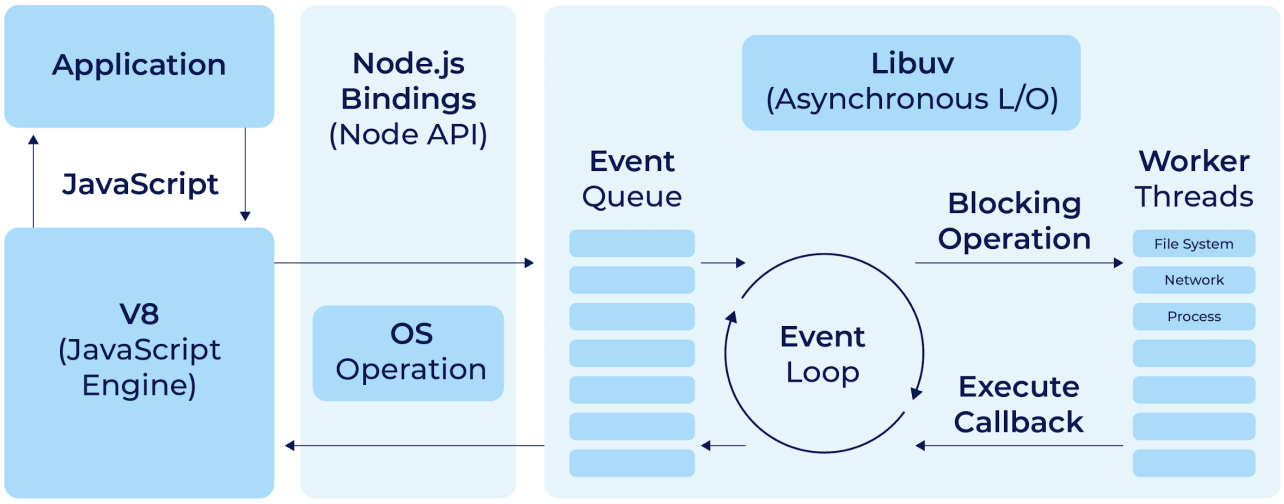

Рис. 2.4. Архітектура *Node.js*

Node.js поставляється з великою кількістю вбудованих модулів, таких як «*http*» для створення серверів чи «*fs*» для роботи з файловою системою. Крім того, використовується пакетний менеджер «*npm*», який дозволяє розробникам легко встановлювати, оновлювати та видаляти залежності для їхніх проектів.

Node.js славиться своєю високою швидкістю виконання завдяки використанню двигуна *V8*. Він також дозволяє створювати масштабовані додатки за допомогою одночасного оброблення багатьох запитів, що дозволяє побудувати ефективні веб-системи. Взаємодія компонентів двигуна *V8* показано на рисунку 2.5.

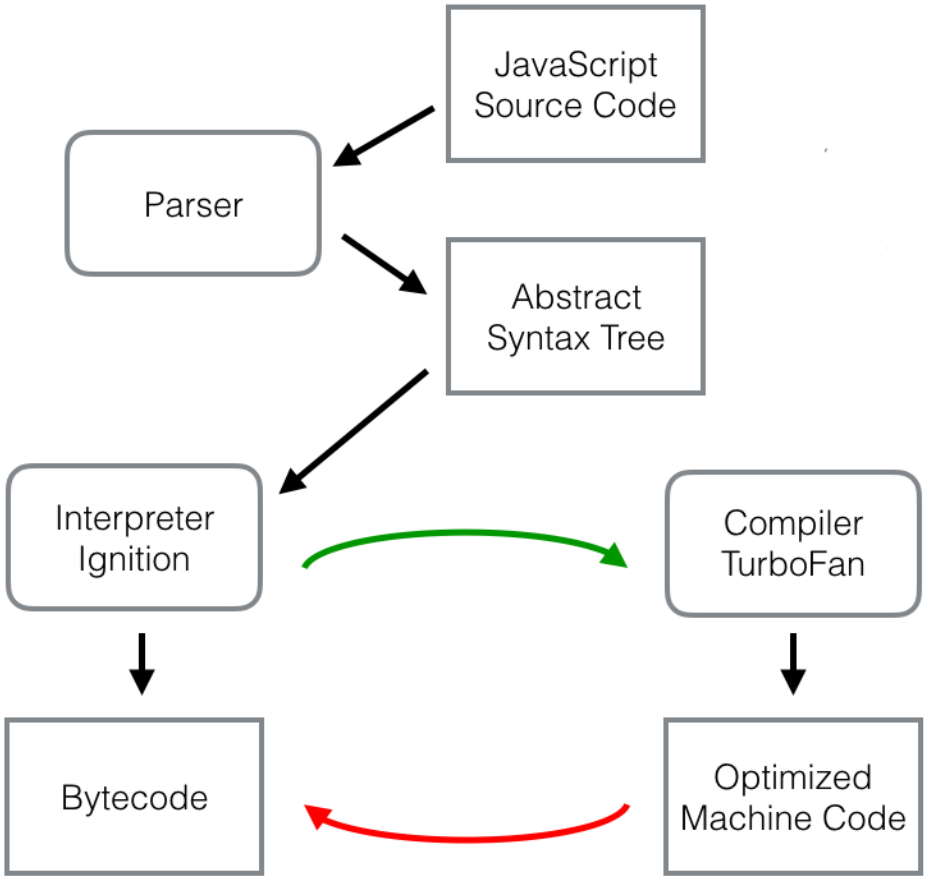

Рис. 2.5. Взаємодія компонентів *V8*

Node.js використовує архітектуру з циклом подій в однопотоковому режимі, що відрізняє його від інших серверів і робить його більш масштабованим. У порівнянні з іншими серверами, які використовують обмежені потоки для обробки запитів, сервери *Node.js* відповідають неблокуючим чи асинхронним способом за допомогою механізму, керованого подіями. Це робить *Node.js* більш масштабованим, дозволяючи йому ефективно обробляти велику кількість запитів. У порівнянні з традиційними серверами, такими як *HTTP*-сервери *Apache*, *Node.js* володіє властивістю обробки більшого обсягу запитів завдяки своєму однопотоковому підходу [12].

Сучасним компаніям необхідне масштабоване програмне забезпечення, і однією з ключових відповідей на цю потребу є *Node.js*. Цей інструмент розв'язує одну з найактуальніших проблем в розробці програмного забезпечення - масштабованість. *Node.js* ефективно обробляє одночасні запити, особливо завдяки використанню кластерного модуля, який керує балансуванням навантаження між

активними ядрами центрального процесора. Однак найбільш вражаючою особливістю *Node.js* є його здатність розділяти програми горизонтально, що в основному досягається за допомогою використання дочірніх процесів.

Використовуючи цей механізм, різні версії додатків можуть бути адаптовані для різних цільових аудиторій, а також легко налаштовуватися для відповіді на вподобання клієнтів.

Node.js використовує двигун виконання *JavaScript V8*, що також використовується у *Google Chrome*. Користуючись концентратором, який надає обгортку для *JavaScript*, двигун виконання стає більш ефективним, що призводить до підвищення швидкодії внутрішнього процесу обробки запитів у *Node.js*.

Одержання та обробка даних у різних потоках зазвичай забирають значний час. У випадку *Node.js* цей процес відбувається дуже швидко, що дозволяє значно економити час. *Node.js* забезпечує ефективну обробку файлів, які обробляються та завантажуються одночасно, що призводить до поліпшення загальної швидкості передачі даних і потокового відео.

Node.js може бути використаний на різних типах операційних систем, таких як *Windows*, *UNIX*, *LINUX*, *MacOS*, а також на інших мобільних пристроях. Для створення автономного виконання його можна поєднати з будь-яким відповідним пакетом.

Node.js базується на принципі однопотокового програмування. Кластер – це модуль, який реалізує багатопотоковість шляхом створення дочірніх процесів, які спільно використовують той самий порт сервера та працюють одночасно.

Також *Node.js* включає в себе інструмент для налагодження, який можна використовувати через вбудований клієнт налагодження. Налагоджувач *Node.js* має обмежені функції, проте він підтримує просту перевірку коду. Для запуску налагоджувача в терміналі, ви можете використовувати ключове слово «*inspect*», передаючи назву файлу *JavaScript*. Наприклад, для перевірки файлу з назвою *script.js*, можна скористатися наступним методом: *$ node inspect script.js* [17].

2.1.3. *Express*

Express.js є легковаговим і гнучким веб-фреймворком для побудови веб-додатків на платформі *Node.js*. Розроблений в 2010 році, *Express* став одним з найпопулярніших інструментів у *Node.js* екосистемі завдяки своїй простоті та продуктивності. Архітектуру фреймворку *Express* зображено на рисунку 2.6.

Головна ідея *Express* полягає в тому, щоб надати зручний і мінімалістичний набір функцій для обробки *HTTP*-запитів та відповідей. Завдяки цьому, розробники можуть ефективно будувати веб-додатки без надмірної складності. Варто відзначити, що *Express* не включає в себе базовий шаблонізатор чи *ORM* (*Object-Relational Mapping*), але забезпечує свободу вибору інших бібліотек для цих завдань [21].

Однією з ключових особливостей *Express* є *Middleware*. Це функції, які обробляють запити перед тим, як вони дістаються до обраних маршрутів. Вони можуть виконувати різноманітні завдання, такі як реєстрація запиту, перевірка аутентифікації, обробка даних тощо. Це надає великий контроль над обробкою *HTTP*-запитів і відкриває простір для реалізації різноманітних функціональностей.

Маршрутизація також є суттєвою частиною *Express*. Вона дозволяє визначити, як програма відповідає на різні *HTTP*-запити і визначати обробники для конкретних шляхів. Маршрутизація спрощує організацію проекту, роблячи його більш модульним і підтримуючи чистоту коду.

Express надає також можливість використовувати шаблонізатори для відтворення *HTML*-сторінок. Завдяки цьому, розробники можуть легко вбудовувати динамічний контент в статичні *HTML*-файли. Популярними шаблонізаторами для *Express* є *EJS*, *Pug* та *Handlebars*.

Забезпечуючи інтеграцію з різноманітними іншими бібліотеками та модулями, *Express* робить його дуже гнучким. Розробник може легко інтегрувати бази даних, обробники форм, аутентифікацію та авторизацію. Це дозволяє розробникам будувати повноцінні веб-додатки з урахуванням потреб конкретного проекту.

Багато фреймворків забезпечують лише базовий функціонал, залишаючи велику частину роботи на розробнику. З іншого боку, *Express* пропонує екосистему

модулів та пакетів, які спрощують багато рутинних завдань, що дозволяє розробникам зосередитися на створенні функціоналу їхнього додатку.

Основні переваги *Express*:

1) Легкість використання. *Express* надає простий та інтуїтивний інтерфейс, що спрощує побудову веб-додатків. Він не нав'язує велику кількість конвенцій, дозволяючи розробникам вибирати підходи, які найкраще відповідають їх потребам.

2) Маршрутизація. *Express* дозволяє легко визначати шляхи (*routes*), які оброблятимуть конкретні *URL*-адреси. Це спрощує структуру коду та дозволяє розподіляти функціональність вашого додатка.

3) Швидкодія. За рахунок використання середовища виконання *Node.js*, *Express* може швидко оброблювати багато одночасних запитів за рахунок неблокуючого вводу/виводу.

Основні недоліки *Express*:

1) Нестача стандартів. *Express.js* не нав'язує жорстких стандартів архітектури, і це може призвести до того, що розробники реалізовують рішення по-різному, що ускладнює розуміння та підтримку коду великого проекту.

2) *Callback Hell*. З використанням колбеків для обробки асинхронних операцій може виникати так званий «*Callback Hell*» або «*Pyramid of Doom*», коли вкладені колбеки стають важкими для читання та розуміння.

3) Мала кількість вбудованих можливостей. У порівнянні з іншими сучасними фреймворками, *Express.js* має менше вбудованих можливостей. Це може означати, що вам потрібно використовувати сторонні бібліотеки для багатьох завдань, що може призвести до додаткового завдання управління залежностями.

На традиційному веб-сайті, що управляється даними, веб-програма очікує *HTTP*-запитів від веб-браузера (або іншого клієнта). Після отримання запиту програма визначає, яку дію слід виконати, використовуючи шаблон *URL*-адреси та, можливо, інформацію, яка міститься в даних *POST* або *GET*. Залежно від потреби, вона може читати або записувати інформацію в базу даних або виконувати інші завдання для задоволення запиту. Потім програма повертає відповідь веб-браузеру,

часто динамічно створюючи *HTML*-сторінку для відображення браузером та вставляючи отримані дані в відповідні місця в шаблоні *HTML* [13].

Express надає методи для вказівки, яка функція викликається для конкретного *HTTP*-запиту (*GET*, *POST*, *SET* і т.д.) та шаблону *URL*-адреси. Також *Express* пропонує методи для вказівки механізму шаблонів, де знаходяться файли шаблонів та який саме шаблон використовувати для створення відповіді. З використанням проміжного програмного забезпечення *Express* розробник може додати підтримку файлів *cookie*, сесій, користувачів та отримувати параметри з *POST*/*GET*-запитів і т.д. *Express* не визначає конкретну поведінку, пов'язану з базою даних, тому розробники можуть використовувати будь-який механізм бази даних, який підтримує *Node*.

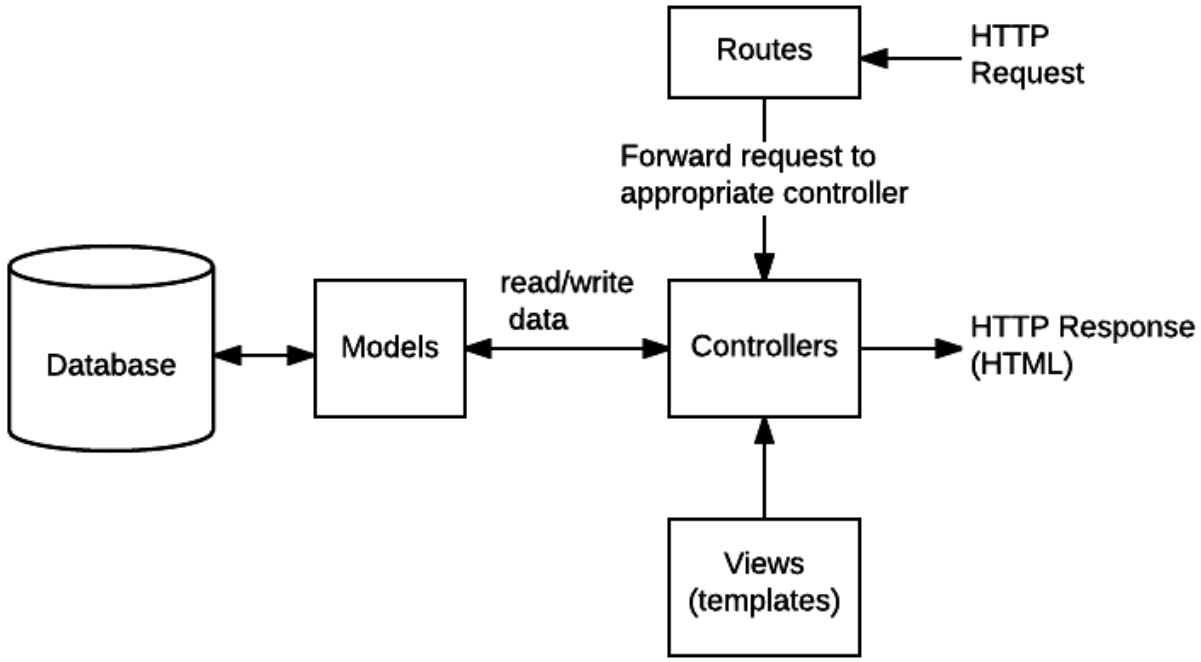

Рис. 2.6. Архітектура фреймворку *Express*

Використання неблокуючих асинхронних *API* набуває ще більшої важливості в середовищі *Node*, ніж у веб-переглядачі, оскільки *Node* працює у виконанні з одним потоком та керується подіями в навколишньому середовищі. Термін «однопотоковий» означає, що всі запити до сервера обробляються у єдиному потоці, а не створюються в окремих процесах. Ця модель виявляється надзвичайно ефективною з точки зору швидкості та ресурсів сервера. Проте це також означає, що

якщо будь-яка з ваших функцій викликає синхронні методи, які вимагають значного часу для завершення, вони будуть блокувати не тільки поточний запит, але й всі інші запити, які обробляються вашою веб-програмою.

Основним фактором, що обмежує продуктивність *Express*, є те, що за замовчуванням *Node.js* використовує лише одне ядро ЦП. Це може викликати труднощі при виконанні завдань, які вимагають інтенсивного використання процесора, оскільки це може швидко насичити ресурси основної системи.

Нижче наведено кілька найрозповсюдженіших сценаріїв використання *Express.js*:

1) Розробка *API*. *Express.js* є чудовим вибором для створення мінімальних, гнучких та масштабованих структур для розробки мікросервісів і серверних програм *RESTful API*. Це є, можливо, основним використанням, яке спричиняє популярність Express.

2) Впровадження веб-сервера. *Express.js* ідеально підходить для обслуговування різних видів веб-програм, будь то статичні сайти, динамічні веб-сторінки або односторінкові додатки. Його можливості включають як статичну маршрутизацію, так і використання шаблонів.

3) Програми в режимі реального часу. Завдяки інтеграції з фреймворками, такими як *Socket.io*, *Express* підтримує розробку програм у режимі реального часу, таких як чатові програми, платформи для онлайн-ігор та інструменти для спільної роботи.

4) Інтеграція в широко використовувані технологічні стеки. *Express.js* є ключовим компонентом у популярних технологічних стеках, таких як *MEN*, *MERN*, *MEAN* і *MEVN*, що забезпечує міцну основу для створення сучасних, масштабованих та ефективних веб-програм.

2.1.4. *Redux*

Redux – це бібліотека управління станом для веб-додатків, яка допомагає зберігати стан застосунку в одному центральному об’єкті, названому «*store*». Ця

бібліотека часто використовується разом із бібліотеками і фреймворками, такими як *React.js*, для покращення управління станом великих і складних додатків. Архітектуру *Redux* зображено на рисунку 2.7.

Основні концепції *Redux*:

1) Сховище (*Store*). Центральний об'єкт, який містить стан додатку. Структура стану повинна бути незмінною і змінюватися тільки за допомогою чистих функцій, таких як «*reducers*».

2) Дії (*Actions*). Об'єкти, які описують зміни стану. Вони є єдиним джерелом інформації для «*store*». Дії викликаються компонентами або іншими частинами додатка.

3) Редуктори (*Reducers*). Чисті функції, які приймають поточний стан і дію, і повертають новий стан. Вони визначають, як саме стан додатка має змінюватися відповідно до викликаної дії.

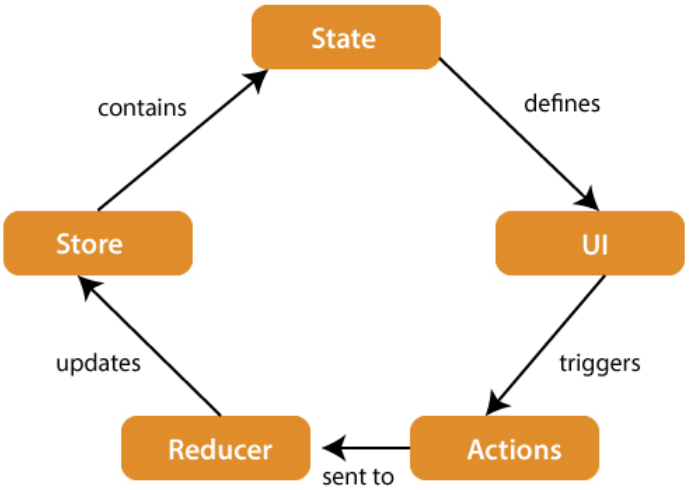

Рис. 2.7. Архітектура *Redux*

Якщо розроблювальний додаток вимагає асинхронних операцій, *Redux* дозволяє використовувати *middleware*, такий як *Redux Thunk* або *Redux Saga*. Це розширює базовий цикл роботи, дозволяючи виконувати асинхронні запити та інші несинхронні операції. Передачу даних в *Redux* разом з *Middleware* показано на рисунку 2.8.

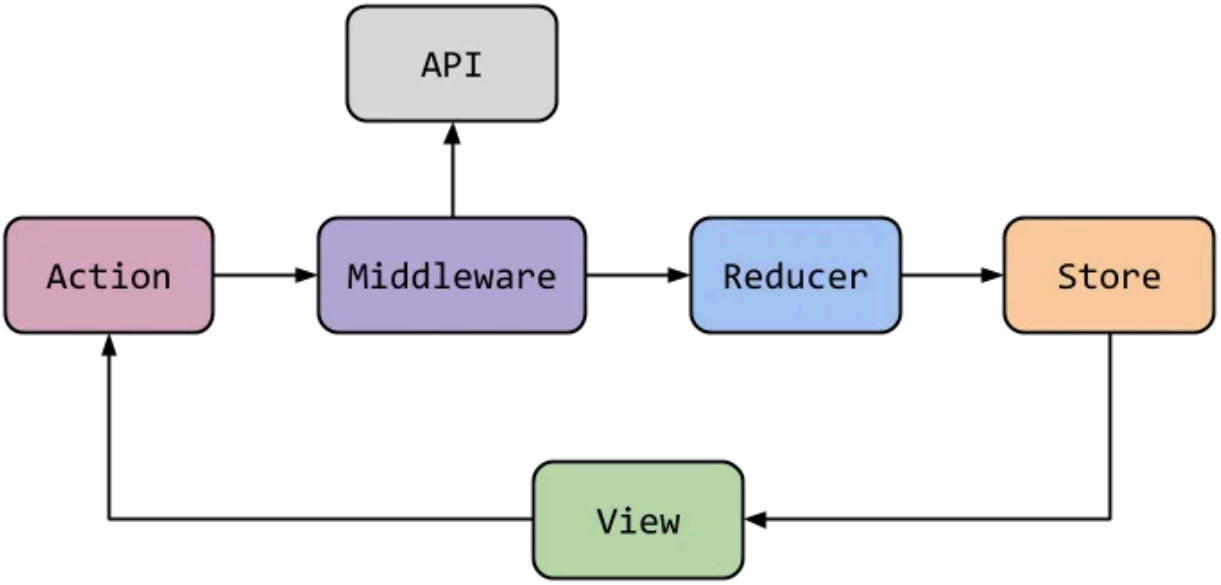

Рис. 2.8. Передача даних в *Redux* разом з *Middleware*

Одразу після випуску, *Redux* став однією з головних тем обговорень у світі веб-програмування. Цей інструмент дозволяє ефективно керувати станом програми в одному місці, зробити зміни більш передбачуваними та відслідкованими, що полегшує розуміння змін, які відбуваються в вашій програмі. Проте, із цими перевагами пов’язані певні труднощі. Деякі розробники вважають, що *Redux* може стати непотрібним шаблоном, який потенційно ускладнює те, що інакше може бути простою задачею. Втім, це залежить від архітектурних рішень, взятих у проекті.

Redux використовується для управління та оновлення даних у програмах, щоб забезпечити їхнє спільне використання кількома компонентами, залишаючи ці компоненти незалежними один від одного. У великому додатку важливо зберігати стан в центральному місці та ділитися ним між різними компонентами.

Якщо необхідно передати дані від батьківського компонента до дочірнього у глибині дерева, це все ще можна зробити за допомогою утиліт *React*, таких як *Context*. Але, коли мова йде про спільне використання стану між компонентами на одному рівні, *Redux* є необхідним.

Такі бібліотеки, як *React* і *Angular*, спроектовані так, щоб компоненти могли внутрішньо керувати своїм станом, не потребуючи зовнішніх бібліотек або інструментів. Це ефективно працює для додатків з невеликою кількістю

компонентів, але зі зростанням додатка управління спільним станом між компонентами стає складнішим завданням.

У додатку, де дані обмінюються між компонентами, може бути важко визначити, де повинен знаходитися стан. В ідеалі дані повинні зберігатися лише в одному компоненті, щоб уникнути заплутування, тому обмін даними між однорідними компонентами стає проблематичним.

Наприклад, для обміну даними між компонентами на одному рівні в *React*, стан має бути розміщений у батьківському компоненті. Метод для оновлення цього стану надається батьківським компонентом і передається як реквізит цим компонентам-рівню.

Redux дозволяє розробникам перехоплювати всі дії, що надсилаються компонентами, до того, як вони будуть передані функції *reducer*. Це перехоплення здійснюється за допомогою проміжного програмного забезпечення, яке представляє собою функції, викликаючи наступний метод, який передається у якості аргументу після обробки поточної дії [25].

У *Redux* стан завжди є передбачуваним. При передачі одного і того ж стану та дії до редуктора завжди отримується однаковий результат, оскільки редуктори є чистими функціями. Також, стан є незмінним, що дозволяє виконувати складні завдання, такі як нескінченне скасування та повторення. Це також відкриває можливість реалізації подорожі у часі, що означає здатність переміщатися вперед і назад між попередніми станами та спостерігати за результатами в реальному часі.

Redux строго дотримується принципів організації коду, що сприяє зрозумінню структури будь-якої програми *Redux* для тих, хто ознайомлений із цим інструментом. Це зазвичай полегшує обслуговування та допомагає відокремити бізнес-логіку від структури компонентів. Для великих програм особливо важливо мати передбачувану та зручну для обслуговування структуру коду.

Redux спрощує налагодження програм. Записуючи дії та стан, можна легко виявити помилки кодування, мережеві помилки та інші види неполадок, що можуть виникнути під час експлуатації. Крім журналювання, в ньому вбудовані відмінні *DevTools*, які дозволяють виконувати подорожі у часі, зберігати дії під час оновлення

сторінки і таке інше. Для середніх і великих додатків налагодження займає більше часу, ніж розробка функціоналу, і *Redux DevTools* робить доступ до всіх переваг *Redux* легким і зручним.

Redux також може бути використаний для виконання візуалізації на стороні сервера. За його допомогою можна виконувати початкову візуалізацію програми, включаючи передачу стану програми на сервер разом із відповіддю на запит сервера. Необхідні компоненти потім відображаються в *HTML* і відправляються клієнтам.

Отже, однією з основних переваг *Redux* є можливість відстежувати історію стану, що дозволяє розробникам аналізувати, як стан змінювався протягом життєвого циклу програми. Проте важливо впроваджувати *Redux* лише тоді, коли він відповідає вашим потребам, і коли вашому проекту необхідний інструмент для управління станом.

2.2. Вибір технологій для розробки браузерного стрімінгового сервісу

2.2.1. *Socket.IO*

Socket.IO є потужною бібліотекою для реалізації двостороннього (дуплексного) зв’язку між клієнтом і сервером у веб-додатках (рис. 2.9). Вона побудована на основі протоколу *WebSocket* та надає простий та ефективний механізм обміну даними в реальному часі. З його допомогою можна легко реалізувати різноманітні сценарії, починаючи від простих чатів і закінчуючи складними системами реального часу.

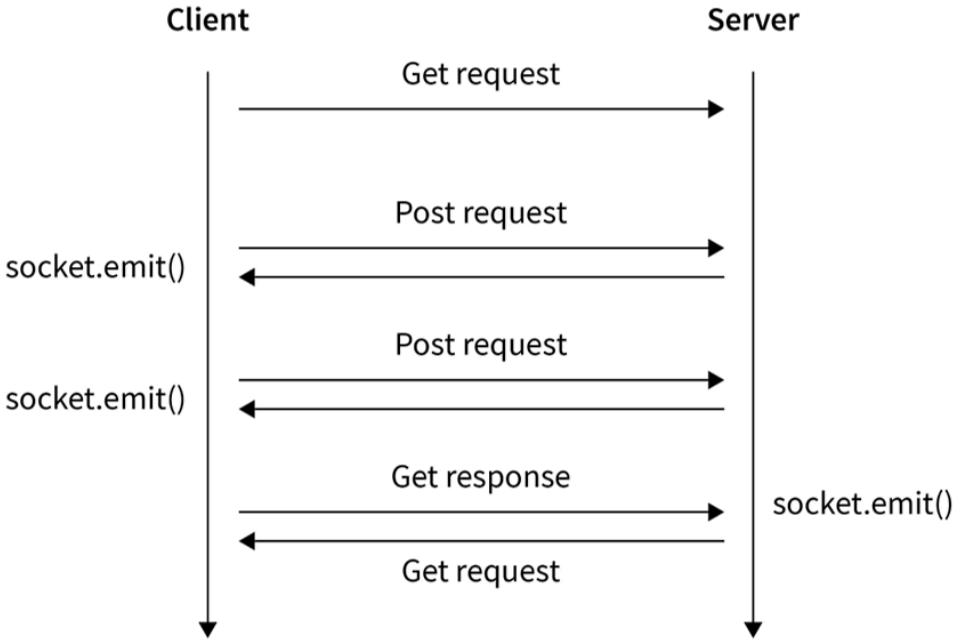

Рис. 2.9. Послідовні *HTTP*-запити в *Socket.IO*

Socket.IO складається з серверної та клієнтської частин, які можуть взаємодіяти між собою за допомогою різних транспортних протоколів, таких як *WebSocket*, *AJAX*, і т.д. Однак основною принциповою можливістю є здатність автоматичного вибору оптимального транспортного протоколу в залежності від умов мережі та технічних можливостей. Діаграму пакета протоколу *WebSocket* показано на рисунку 2.10.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FIN	RSV1	RSV2	RSV3	Opcode				Mask	Payload length						
Extended payload length (optional)															
Masking key (optional)															
Payload data															

Рис. 2.10. Діаграма пакета протоколу *WebSocket*

Socket.IO надає зручний механізм обміну подіями між клієнтом і сервером. Це дозволяє створювати додатки, які реагують на події в реальному часі, такі як чати,

онлайн-гри, або моніторинг систем. За допомогою подій, інформація може передаватися взамно, що робить взаємодію більш динамічною та гнучкою.

Socket.IO розроблено з урахуванням масштабованості. Він може впоратися з великими навантаженнями та забезпечує надійний обмін даними. Також врахована можливість використання балансувальників навантаження для розподілу трафіку між багатьма серверами *Socket.IO*.

Створення програм у режимі реального часу з використанням популярних стеків веб-програмування, таких як *LAMP* (*PHP*), традиційно було завданням, що викликало значні труднощі. Це включає в себе опитування сервера для виявлення змін, відстеження часових міток і виявляється далеко від оптимальної швидкості.

Традиційно сокети виступали як рішення, навколо якого було побудовано більшість систем реального часу, забезпечуючи двосторонній канал зв'язку між клієнтом і сервером. Це означає, що сервер може ініціювати передачу повідомлень клієнтам. Кожного разу, коли відбувається подія, концепція полягає в тому, що сервер отримує її і передає відповідним підключеним клієнтам.

Socket.IO, який широко використовується, використовується такими організаціями, як *Microsoft Office*, *Yammer*, *Zendesk*, *Trello* та інші, для побудови надійних систем реального часу. Це один з найпотужніших фреймворків *JavaScript* на *GitHub* і найбільш залежний модуль *NPM* (*Node Package Manager*). *Socket.IO* також має величезну спільноту, що спрощує отримання допомоги.

В *Socket.IO* існує декілька зарезервованих подій, до яких можна отримати доступ за допомогою об'єкта сокета на стороні сервера:

- *connect*: виникає, коли клієнт успішно підключається до сервера. Ви можете використовувати цю подію для виконання дій після успішного встановлення з'єднання;
- *message*: використовується для обміну даними між клієнтом і сервером. Клієнт і сервер можуть відправляти один одному повідомлення з використанням цієї події;

– *disconnect*: спрацьовує, коли з'єднання між клієнтом і сервером обривається. Вона може виникнути через помилку, закриття вікна браузера або інші причини;
– *reconnect*: відбувається, коли клієнт успішно відновлює з'єднання після втрати зв'язку;
– *ping*: виникає, коли сервер надсилає *ping*-запит клієнту для визначення його доступності;
– *join*/*leave*: використовуватися для відстеження приєднання та від'єднання користувачів до/від кімнат чи каналів в системі *Socket.IO*. Вони корисні в розподілених системах чи в сценаріях, де необхідно контролювати присутність користувачів в реальному часі.

Також *Socket.IO* дозволяє створювати «простори імен» для сокетів, що фактично означає визначення різних кінцевих точок або шляхів. Ця корисна можливість допомагає зменшити кількість ресурсів (*TCP*-з'єднань) та одночасно розв'язувати проблеми у вашій програмі, розділяючи канали зв'язку. Декілька просторів імен фактично використовують одне з'єднання *WebSocket*, що дозволяє зберігати порти сокетів на сервері [19].

Простори імен створюються на боці сервера, проте клієнти приєднуються до них, висилаючи запит на сервер.

2.2.2. *WebRTC*

WebRTC (*Web Real Time Communication*) – це технологія, яка впроваджує обмін даними в режиму реального часу у веб-додатках, дозволяючи безпосередню взаємодію між користувачами через браузер (рис. 2.11). Розроблений спільними зусиллями великих інтернет-компаній, таких як *Google*, *Mozilla* і *Opera*.

Однією з ключових особливостей *WebRTC* є можливість передачі аудіо та відео безпосередньо в браузері, не використовуючи додаткових розширень чи плагінів. Це робить технологію дуже зручною та доступною для веб-розробників, які можуть легко інтегрувати функції відеоконференцій, відео-дзвінків, чатів та інших реального

часу застосунків без глобальних труднощів. Також *API WebRTC* включає захоплення медіа файлів, кодування та декодування аудіо та відео, рівень транспортування та керування сеансами [9].

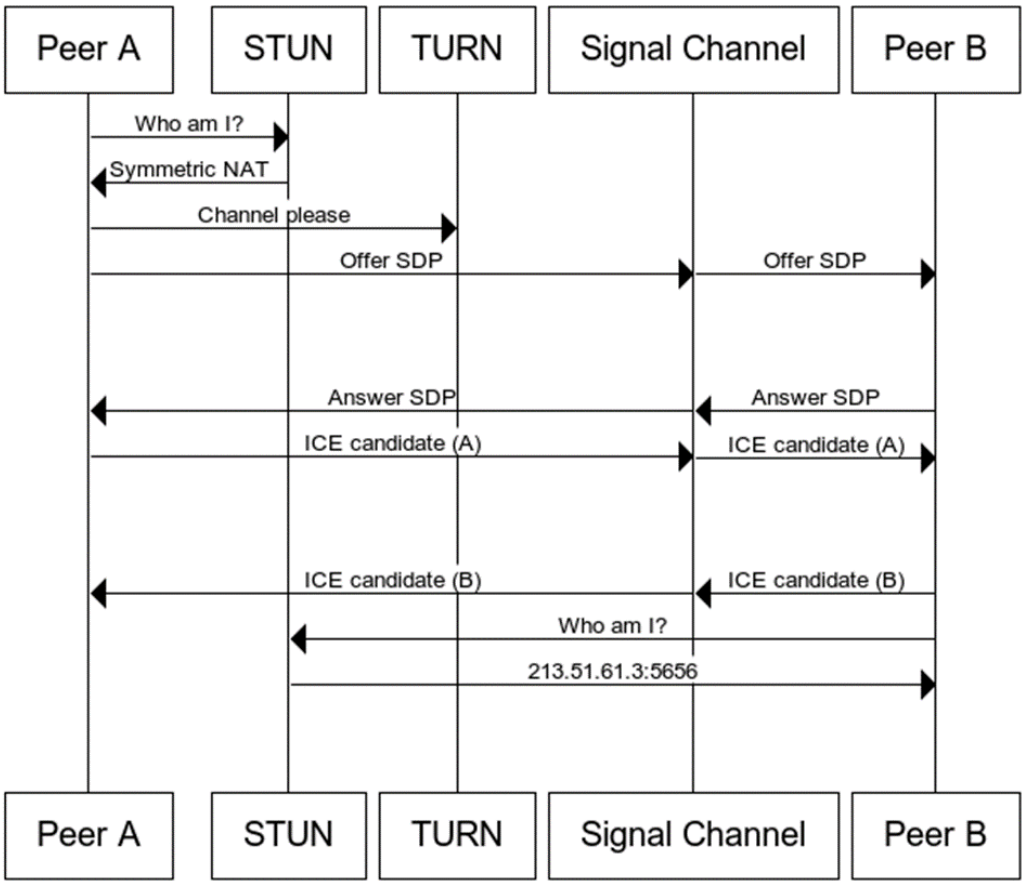

Рис. 2.11. Створення однорангового з’єднання в *WebRTC*

WebRTC також використовується для реалізації веб-додатків для віддаленого навчання, медичних консультацій, віртуальних конференцій та інших відмінних застосувань. Це робить технологію ключовою для сучасного веб-розвитку, забезпечуючи нові рівні взаємодії та спілкування в онлайн-середовищі.

До переваг *WebRTC* відносяться:

1) *WebRTC* вбудований безпосередньо в браузери, що робить його легко доступним. Розробники можуть використовувати *JavaScript API* для доступу до функціональності *WebRTC*.

2) *WebRTC* реалізований як відкритий стандарт, розроблений консорціумом *W3C* і *IETF*. Це забезпечує високий рівень сумісності між різними браузерами і дозволяє створювати одноманітні веб-додатки для взаємодії в реальному часі.

3) *WebRTC* надає вбудовану безпеку через шифрування даних з точки походження до пункту призначення, що є важливим для забезпечення конфіденційності інформації в реальному часі.

Під час використання *WebRTC* спочатку потрібно отримати доступ до камери та мікрофона користувача. Визначаємо наявні пристрої, отримуємо згоду від користувача на їх використання і керуємо процесом передачі даних.

Передача аудіо та відео даних через інтернет є складним завданням, яке включає в себе використання кодування та декодування. Цей процес включає поділ відеокадрів і аудіохвиль на менші фрагменти та їх подальше стиснення. Цей алгоритм відомий як кодек, і існує велика кількість різних кодеків, які використовуються різними компаніями для вирішення різних бізнес-завдань. У *WebRTC* також доступно багато кодеків, таких як *H.264*, *iSAC*, *Opus* і *VP8*. При з'єднанні двох браузерів вони автоматично вибирають найбільш оптимальний підтримуваний кодек для ефективної взаємодії між користувачами. Слід зазначити, що *WebRTC* виконує значну частину роботи з кодуванням на рівні системи, що полегшує процес для користувачів.

Транспортний рівень відповідає за управлінням порядком передачі пакетів, боротьбою з втратами пакетів і встановленням з'єднань з іншими користувачами. Крім того, *WebRTC API* надає зручний доступ до подій, які повідомляють про можливі проблеми зі з'єднанням.

Функції керування сеансом включають управління, встановлення та організацію з'єднань, які зазвичай відомі як сигналізація. Якщо ви передаєте аудіо та відео потоки користувачеві, також має сенс передавати супутні дані, що здійснюється за допомогою *API RTCDataChannel*.

APIMediaStream представляє синхронізовані потоки мультимедіа, такі як потік, який об'єднує відео та аудіо доріжки з вхідних джерел, таких як камера та мікрофон.

Кожен об'єкт *MediaStream* має вхід, який може бути створений за допомогою методу *getUserMedia()*, та вихід, який може бути направлений на елемент відео або *RTCPeerConnection*.

Метод *getUserMedia()* приймає параметр у вигляді об’єкта *MediaStreamConstraints* і повертає об’єкт *Promise*, який конвертується в об’єкт *MediaStream*. Кожен об’єкт *MediaStream* має унікальний ідентифікатор «*label*». Методи *getAudioTracks()* та *getVideoTracks()* повертають масив доріжок *MediaStreamTrack*.

WebRTC використовує *RTCPeerConnection* для передачі потокових даних між браузерами, які також відомі як однорангові вузли. Проте для координації з’єднань та відправки управляючих повідомлень, що відомо як сигналізація, також необхідний. Важливо відзначити, що методи та протоколи сигналізації не є частиною *WebRTC API* та не визначені у ньому [18].

Замість цього розробники додатків, які використовують *WebRTC*, можуть вибрати будь-який протокол обміну повідомленнями за їхнім вибором, наприклад, *SIP* або *XMPP*, і використовувати будь-який відповідний дуплексний (двосторонній) канал зв’язку.

Сигналізація використовується для обміну трьома видами інформації:

– Повідомлення для управління сеансом: це включає ініціацію або завершення зв’язку та повідомлення про можливі помилки;
– Конфігурація мережі: для надання зовнішньому середовищу інформації про *IP*-адресу та порт вашого комп’ютера;
– Медіа-можливості: визначення кодеків та дозволів, які підтримує ваш браузер, і взаємодію з якими він бажає встановити.

Успішне завершення обміну інформацією через сигналізацію необхідно перед тим, як розпочати одноранговий потік передачі даних. Після успішного завершення процесу сигналізації дані можуть передаватись безпосередньо між викликаючим і викликаним абонентом у режимі однорангової взаємодії. В іншому випадку, у разі невдачі, дані можуть бути передані через проміжний сервер ретрансляції.

Кодеки та протоколи, які використовуються в рамках *WebRTC*, виконують значний обсяг роботи для забезпечення можливості ведення реального часу обміну повідомленнями, навіть в умовах ненадійних мереж:

– приховання втрати пакетів;

– усунення відлуння;
– адаптивність до пропускної здатності;
– динамічна буферизація джиттера;
– автоматичне регулювання підсилення;
– зниження та приглушення шуму;
– очищення зображень.

2.2.3. *FFmpeg*

FFmpeg є потужним інструментом для обробки мультимедіа, який надає засоби для запису, конвертації та потокової обробки аудіо та відео даних. Цей відкритий програмний продукт має широкий спектр функціональних можливостей і є незамінним інструментом для вирішення різноманітних завдань у галузі обробки мультимедіа [10].

Однією з ключових особливостей *FFmpeg* є його спроможність працювати з різними форматами аудіо та відео. Програма підтримує безліч кодеків, що дозволяє використовувати її для конвертації файлів з одного формату в інший без втрати якості (рис. 2.12). Багато користувачів визнають *FFmpeg* за надійний інструмент для роботи з великою кількістю мультимедійних форматів.

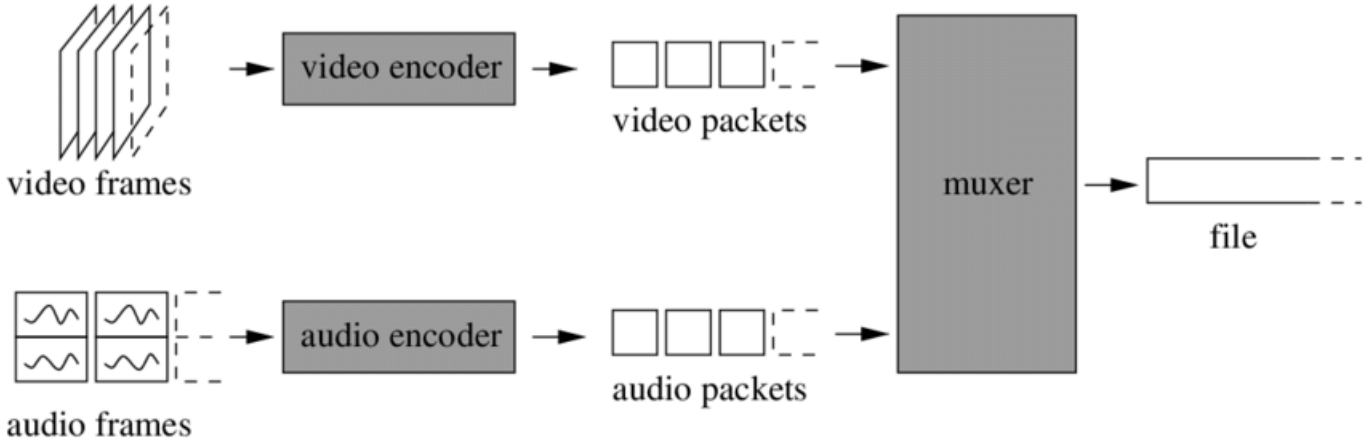

Рис. 2.12. Процес кодування відео в *FFmpeg*

Окрім конвертації, *FFmpeg* дозволяє здійснювати різноманітні операції обробки, такі як вирізання та з’єднання відео, додавання аудіо доріжок, покращення

якості зображення, обрізка та зміна роздільної здатності. Ці можливості роблять його незамінним інструментом для професіоналів у сфері мультимедіа-продукції.

Ще однією важливою характеристикою *FFmpeg* є його можливість працювати в режимі пакетної обробки, що дозволяє автоматизувати рутинні завдання. Це робить програму ефективним інструментом для автоматизації процесів обробки мультимедіа на великих обсягах даних.

FFmpeg також широко використовується у веб-розробці, оскільки вона дозволяє вставляти аудіо та відео у веб-сайти з різних джерел. Із застосуванням *FFmpeg*, веб-розробники можуть створювати динамічні та високоякісні мультимедійні відтворення для своїх проектів [20].

Головна мета цього програмного пакету полягає в забезпеченні швидкого, ефективного та гнучкого рішення для редагування мультимедійних файлів. З плином часу *FFmpeg* набув великої популярності і став улюбленим інструментом для багатьох фахівців і ентузіастів у галузі відеопродукції, трансляції та обробки мультимедіа.

FFmpeg включає в себе декілька компонентів, кожен з яких має свою власну функціональність для обробки мультимедійних файлів (рис. 2.13). Серед цих компонентів можна виділити:

– *ffmpeg*. Це основний інструмент командного рядка, призначений для виконання різноманітних операцій, таких як конвертація, редагування, фільтрація та потокова обробка відео та аудіо файлів;
– *ffprobe*. Командний рядок, що збирає інформацію про мультимедійні файли, включаючи дані про кодек, формат, бітрейт та інші метадані;
– *ffplay*. Простий мультимедійний програвач, призначений для попереднього перегляду відео та аудіо файлів під час їх обробки;
– *libavcodec*. Бібліотека, яка містить набір аудіо- та відеокодеків, дозволяючи *FFmpeg* декодувати та кодувати різноманітні мультимедійні формати;
– *libavformat*. Бібліотека, що обробляє введення та виведення мультимедійних контейнерів, дозволяючи *FFmpeg* читати та записувати різноманітні формати файлів;

– *libavfilter*. Бібліотека, яка пропонує різноманітні фільтри для обробки відео та аудіо, такі як зміна розміру, обрізка, корекція кольорів і мікшування аудіо;
– *libavdevice*. Бібліотека, яка забезпечує підтримку різних пристроїв введення та виведення, таких як камери, мікрофони та дисплеї;
– *libavutil*. Службова бібліотека, яка включає різноманітні допоміжні функції та структури даних, використовувані іншими компонентами *FFmpeg*.

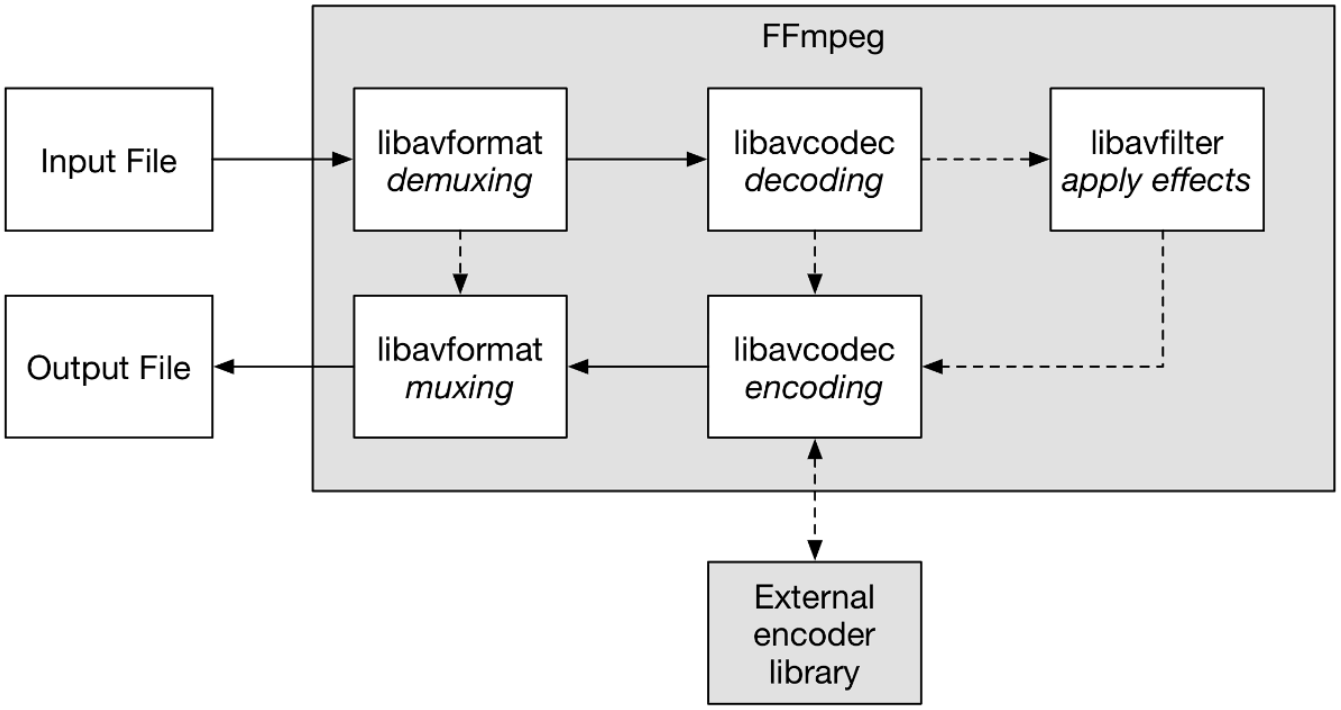

Рис. 2.13. Обробка вхідного файлу в фреймворку *FFmpeg*

FFmpeg представляє собою універсальний інструмент, призначений для використання у різних завданнях, пов'язаних з обробкою мультимедіа. Декілька найбільш поширених типів використання *FFmpeg* включають в себе:

1) Конвертування форматів: *FFmpeg* володіє широким спектром підтримки відео та аудіо кодеків, що дозволяє користувачам здійснювати легку конвертацію файлів між різними форматами. Ця можливість має значення з точки зору сумісності, оскільки різні пристрої та програми можуть вимагати конкретних форматів файлів.
2) Редагування відео та аудіо: *FFmpeg* надає різноманітні можливості редагування, такі як вирізання, об'єднання та обрізання відео та аудіо

файлів. Користувачі можуть також додавати чи видаляти аудіо доріжки, регулювати гучність та змінювати інші параметри, такі як частота кадрів і роздільна здатність.

3) Фільтрація та ефекти: За допомогою різноманітних фільтрів *FFmpeg* дає можливість застосовувати різні ефекти та трансформації до мультимедійних файлів. Приклади фільтрів включають зміну розміру, обрізання, обертання, корекцію кольорів, зменшення шуму та вирівнювання звуку (рис. 2.14).
4) Потокова передача: *FFmpeg* дозволяє реалізовувати потокову передачу відео та аудіо контенту в реальному часі через Інтернет або місцеві мережі. Користувачі можуть налаштовувати програмне забезпечення для роботи в якості сервера або клієнта для потокової передачі і підтримки різних потокових протоколів, таких як *HLS* і *DASH*.
5) Запис і захоплення екрана: *FFmpeg* надає можливість запису екрану комп'ютера, відео з веб-камер та аудіо з мікрофонів. Ця функція є важливою для створення навчальних посібників, презентацій та інших записів.
6) Операції з метаданими: *FFmpeg* може зчитувати, записувати та редагувати метадані, пов'язані з мультимедійними файлами, такі як заголовки, описи та інформація про авторські права.

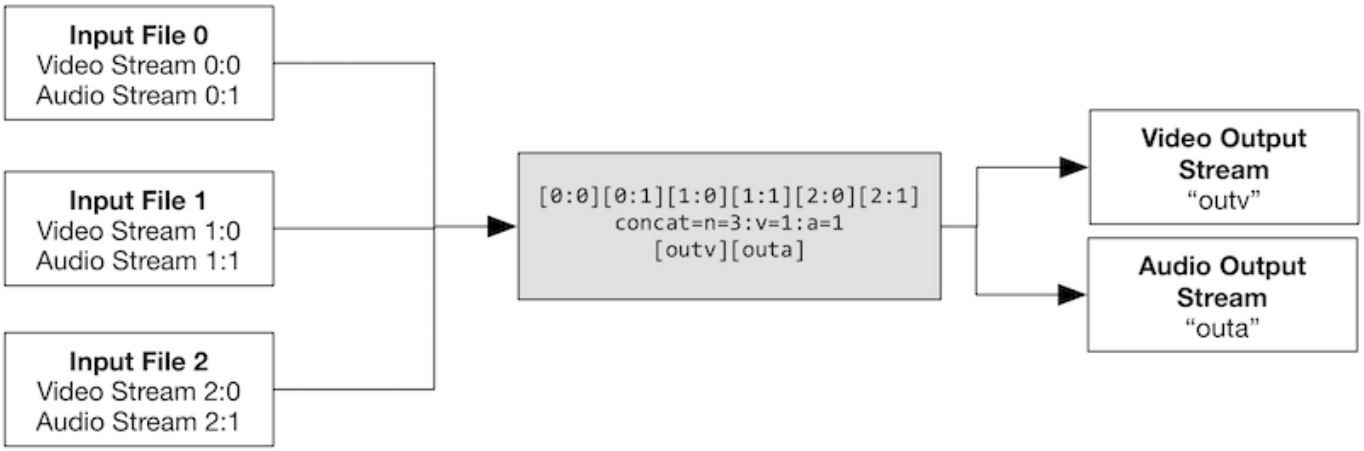

Рис. 2.14. Конкатенація декількох потоків в *FFmpeg*

2.3. Висновки до розділу

У другому розділі було ретельно розглянуто та проаналізовано важливі технології для реалізації браузерного відео стрімінгового сервісу. Цей аналіз включав такі технології: *React.js*, *Node.js*, *Redux*, *Socket.IO*, *WebRTC* та *FFmpeg*.

React.js був обраний як основна бібліотека для розробки інтерфейсу користувача. Його можливості у створенні ефективних та легко підтримуваних компонентів забезпечили високу продуктивність та зручність у взаємодії з користувачем. *React.js* відзначається декларативним підходом до створення інтерфейсів, що спрощує розробку великих та складних додатків. *Virtual DOM* забезпечує ефективне оновлення інтерфейсу, що є важливим для забезпечення відмінного користувацького досвіду.

Node.js використовується як серверна платформа, що дозволяє використовувати *JavaScript* для розробки серверної частини додатка. Його асинхронний підхід і використання одного потоку для багатьох одночасних з'єднань дозволяє створювати швидкі та ефективні серверні додатки, а також сприяє швидкості обробки запитів і реалізації масштабованих додатків.

Redux був використаний для ефективного керуванням станом додатка. Він допомагає уникнути проблем, пов'язаних із складністю стану великих додатків. Його однонапрямлений потік даних і централізоване збереження стану робить код більш передбачуваним і легше супроводжуваним. *Middleware* в *Redux* дозволяє розширювати його функціональність, наприклад, для обробки асинхронних дій.

Socket.IO є бібліотекою для реалізації багаторівневих з'єднань між клієнтом та сервером через протокол *WebSocket*. Він був впроваджений для забезпечення двостороннього зв'язку між клієнтом та сервером у режимі реального часу. Його можливості забезпечують миттєву передачу даних та ефективну обробку подій, що є важливим для багатьох сучасних додатків, таких як чати, групові трансляції та конференції.

WebRTC був використаний для забезпечення підтримки передачі аудіо та відео даних у режимі реального часу. Його масштабованість та висока якість забезпечують зручність та ефективність комунікації через браузер. Також *WebRTC* забезпечує

низьку затримку та високу якість трансляцій, тому він є ключовим для веб-додатків, що використовують режим реального часу та медіа дані. *API WebRTC* має можливість отримувати доступ до медіа пристроям користувача, таких як камера і мікрофон. Для успішного встановлення з'єднання *WebRTC* може використовувати сервера *Session Traversal Utilities for NAT* та *Traversal Using Relays around NAT*. В свою чергу *STUN* допомагає виявляти публічні *IP*-адреси та порти пристрою, а *TURN* використовується у випадку, якщо пряме з'єднання між пристроями неможливе.

FFmpeg є набором бібліотек та інструментів для обробки аудіо і відео даних. Даний фреймворк був використаний для обробки, конвертації та передачі мультимедійних даних на платформу. Його багатий функціонал дозволяє забезпечити високий рівень обробки відео та аудіо в сервісі. *FFmpeg* може приймати різні формати файлів, а також може виконувати різні операції, такі як зміна розміру, зміна частоти кадрів, накладання фільтрів, кодування та декодування аудіо і відео та інші. *FFmpeg* підтримує безліч кодувальників і форматів, що робить його потужним інструментом для роботи з мультимедійними даними.

Загалом в даному розділі було розглянуто технології, які, працюючи разом, надають змогу створювати сучасні браузерні сервіси з високою продуктивністю та розширюваністю. Використання *React.js* та *Node.js* дозволяє ефективно поєднувати клієнтську та серверну розробку, а використання *Redux* забезпечує ефективне управління станом додатку. *Socket.IO* та *WebRTC* вносять можливість реалізації реального часу та багатокористувацької взаємодії. Підтримка звукової та відео інформації надається завдяки *FFmpeg*, що робить цей набір технологій ідеальним для розробки браузерного відео стрімінгового сервісу для одночасного проведення прямих трансляцій на кілька платформ.

РОЗДІЛ 3
СТВОРЕННЯ БРАУЗЕРНОГО СЕРВІСУ

3.1. Структура браузерного сервісу

Для розробки браузерного сервісу було використано наступні інструменти:

– редактор вихідного коду *Visual Studio Code*;
– браузер *Google Chrome*;
– мова розмітки документів у браузері *HTML*;
– каскадну таблицю стилів *CSS*;
– *CSS*-фреймворк *Tailwind*;
– мова програмування *TypeScript*;
– бібліотека для розробки інтерфейсу *React.js*;
– програмна платформа *Node.js*;
– бібліотека для керування станом додатку *Redux*;
– бібліотека для обміну даними в реальному часі *Socket.IO*;
– технологія обміну медіа даними через браузер *WebRTC*;
– фреймворк для обробки медіа даних *FFmpeg*.

Браузерний сервіс побудовано на основі архітектури клієнт-сервер. Спершу клієнт та сервер створюють між собою двонаправлене з'єднання, після чого обмінюються один з одним описом сесії за допомогою протоколу *SDP*. Далі сервер отримує від клієнта медіа потік, котрий він перекодовує в *RTMP*-потік та передає платформі. Схема взаємодії модулів браузерного сервісу зображено на рис. 3.1.

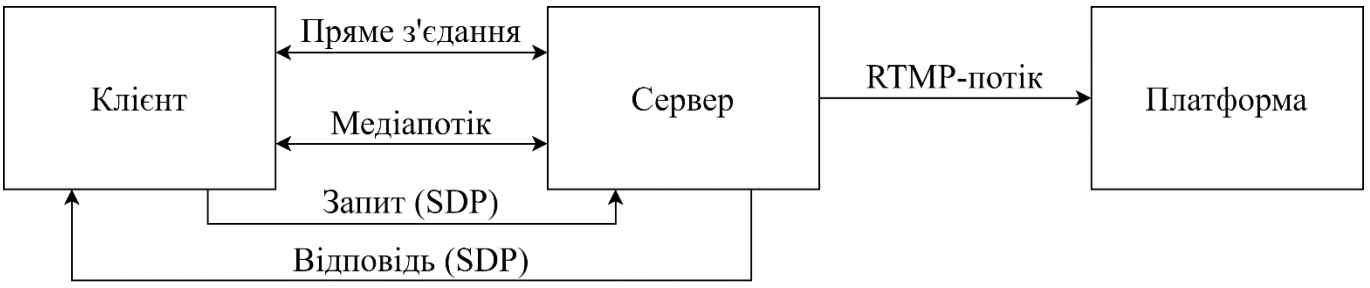

Рис. 3.1. Взаємодія модулів браузерного сервісу

Структура клієнтської частини браузерного сервісу:

– папка *node_modules*;
– папка *public*;
– папка *store*;
– *Input.tsx*;
– *ModalWindow.tsx*;
– *MainPage.tsx*;
– *StreamPage.tsx*;
– *socket.ts*;
– *index.css*.

Структура серверної частини браузерного сервісу:

– папка *node_modules*;
– *server.ts*.

Для створення двонаправленого з’єднання між клієнтом та сервером використовується бібліотека *Socket.IO*, для передачі медіа потоку – *WebRTC*, для передачі *RTMP*-потоку – фреймворк *FFmpeg*. Діаграму пакетів браузерного сервісу зображено на рис. 3.2.

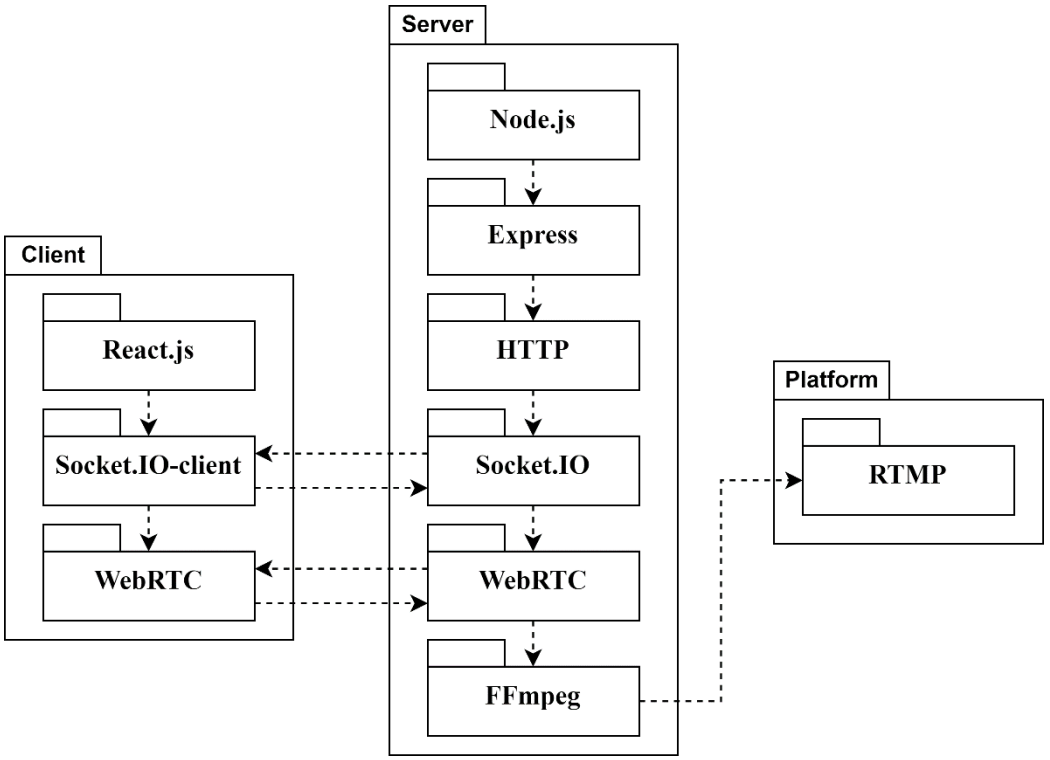

Рис. 3.2. Діаграма пакетів браузерного сервісу

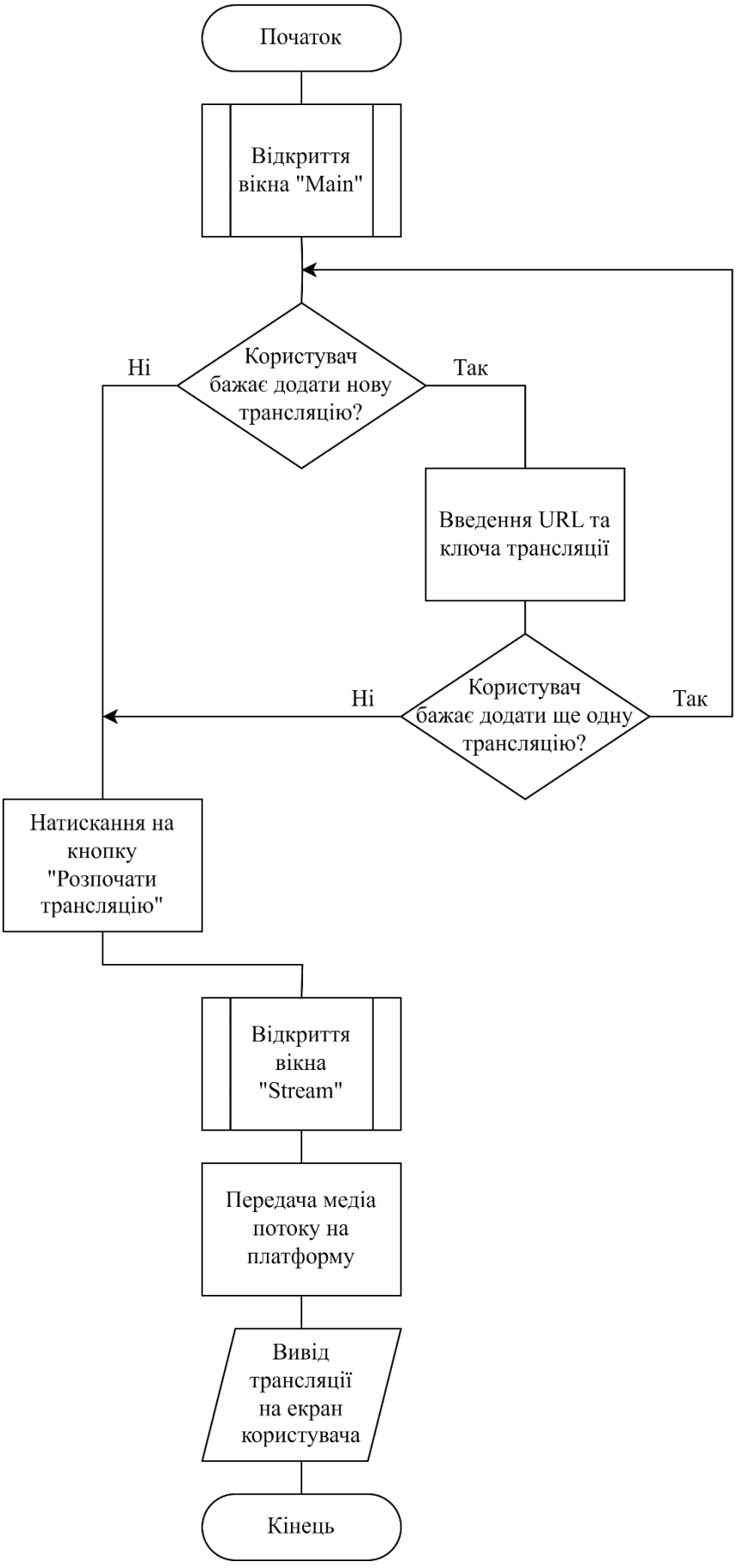

Рис. 3.3. Схема алгоритму роботи браузерного сервісу

Схема алгоритму роботи браузерного сервісу зображено на рисунку 3.3.

Папка *node_modules* містить залежності, які є необхідними для роботи сервісу. Папка *public* містить файл *index.html* з основним *HTML*-документом, файл конфігурації *manifest.json* та інші статичні файли. Папка *store* містить файл *store.ts* для конфігурації сховища *Redux* та файл *streamSlice.ts* для налаштування стану сервісу, який зберігає дані про трансляції.

Файли *Input.tsx* та *ModalWindow.tsx* відповідають за зовнішній вигляд та логіку роботи компонентів: поле для вводу даних користувача, модальне вікно налаштування трансляції.

MainPage.tsx та *StreamPage.tsx* містять сценарії поведінки веб-сторінок сервісу, які написані мовою *TypeScript* [24].

Файл *socket.ts* використовується для створення глобальної змінної в якій проводиться ініціалізація з'єднання між клієнтом та сервером та визначаються параметри даного з'єднання.

Таблиця стилів, яка використовується для стилізації вмісту веб-сторінок сервісу та конфігурація *CSS*-фреймворку *Tailwind* міститься у файлі *index.css*.

Файл *server.js* є файлом налаштування та визначення сценаріїв сервера. В цьому файлі створюється з'єднання з клієнтом за допомогою технологій *Socket.IO* і *WebRTC*, а також виконується перекодування вхідного медіа потоку і передача його на платформу.

3.2. Організація з'єднання між клієнтом і сервером

Для реалізації з'єднання між клієнтом і сервером спершу створюється сам сервер. Для його визначення та налаштування необхідно: створити екземпляр об'єкта *Express*, який використовується для налаштування та обробки *HTTP*-запитів; використати *middleware*, який дозволяє використовувати статичні файли з каталогу «*public*» через шлях «/*static*»; створити *HTTP*-сервер; створити екземпляр *Socket.IO Server* з налаштуванням *CORS* (*Cross Origin Resource Sharing*), що дозволяє обмінюватися даними між різними джерелами. У цьому випадку, екземпляр *Server*

налаштовано так, щоб дозволяти доступ з будь-якого джерела з використанням методів *GET* та *POST*. Вказуються порт «3001» та ім'я хоста «*localhost*», на якому буде сервер:

```
const app = express()
app.use('/static', express.static(path.join(__dirname, 'public')))
const server = createServer(app)
const PORT = 3001
const HOSTNAME = 'localhost'
const io = new Server(server, {
  cors: {
    origin: '*',
    methods: ['GET', 'POST'],
},})
```

Для встановлення і керування підключенням між двома одноранговими вузлами створюється об'єкт *RTCPeerConnection*. В об'єкт передається конфігурація, яка містить сервери *ICE* (*Interactive Connectivity Establishment*) для встановлення з'єднання.

```
const peerConnection = new wrtc.RTCPeerConnection({
  iceServers: [{
      urls: [
        'stun:stun.l.google.com:19302',
        'stun:stun1.l.google.com:19302',
        'stun:stun2.l.google.com:19302',
        'stun:stun3.l.google.com:19302',
        'stun:stun4.l.google.com:19302',
],},],})
```

Для збереження кандидатів *ICE*, отриманих від сервера, створюється пустий масив:

```
const serverIceCandidates: any = []
```

Діаграма послідовності створення з’єднання між модулями браузерного сервісу зображено на рисунку 3.4.

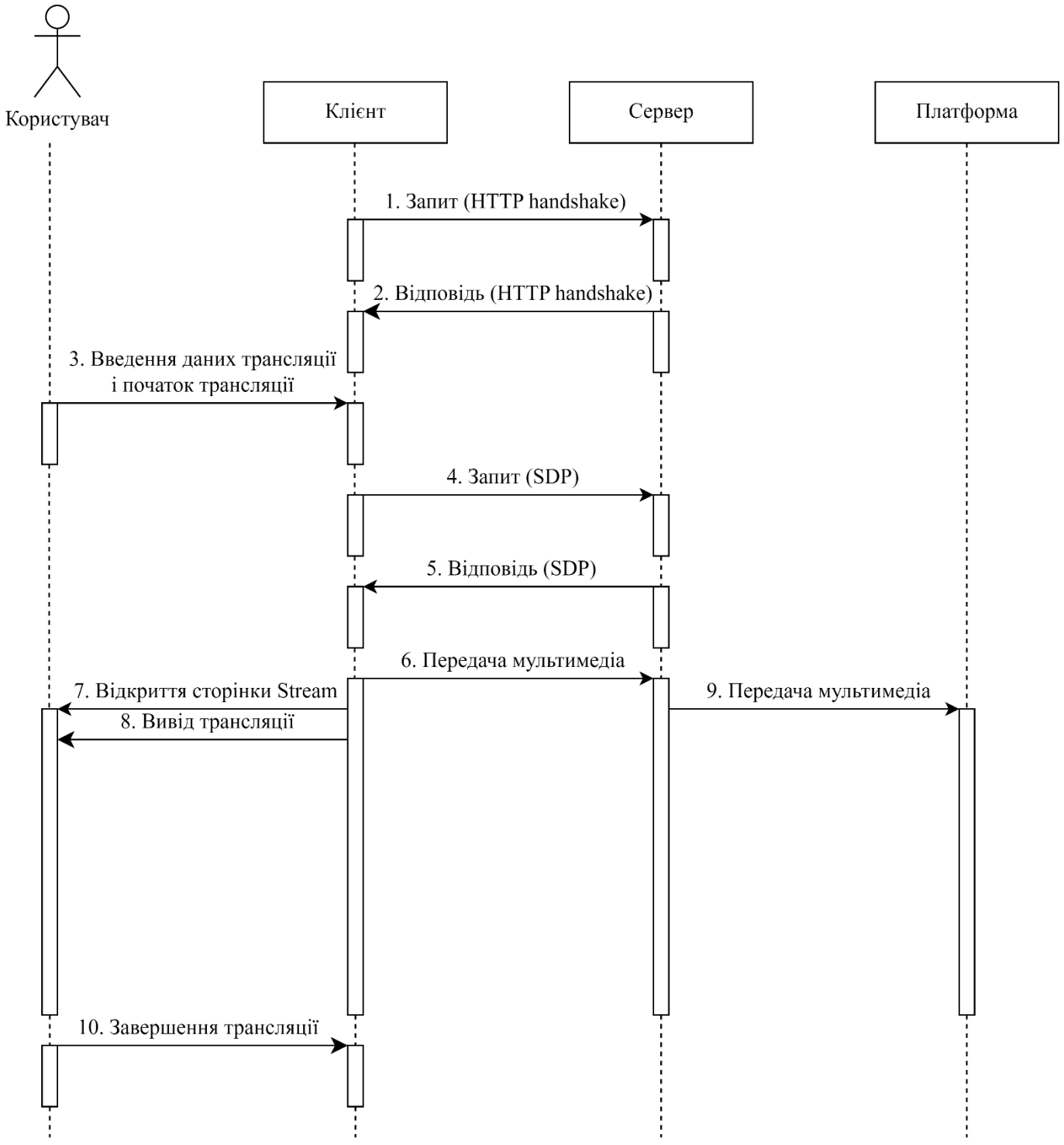

Рис. 3.4. Діаграма послідовності створення з’єднання

Далі встановлюється обробник подій для події «*onicecandidate*», який спрацьовує, коли одноранговий вузол створює кандидата *ICE*. Кандидати *ICE* необхідні для визначення оптимального маршруту для передачі даних між

одноранговими вузлами. В обробнику подій відбувається перевірка, чи існує кандидат, якщо так, то цей кандидат додається до масиву *serverIceCandidates*:

```
peerConnection.onicecandidate = (e: any) => {
  if (e.candidate) {
    serverIceCandidates.push(e.candidate)
}}
```

Для обробки події «conneection» встановлюється відповідний обробник, який спрацьовує кожен раз, коли новий клієнт підключається до сервера. Функція з параметром *socket* представляє об'єкт сокету для взаємодії з підключеним клієнтом. Також встановлюється обробник подій для події «*offer*», яка відбувається, коли клієнт відправляє пропозицію «*offer*» і список кандидатів *ICE* на сервер. Параметри *offer* і *iceCandidates* представляють дані, які надходять від клієнта. Сервер використовує отриману пропозицію «*offer*» і встановлює її як віддалений опис для об'єкту *peerConnection*. Сервер створює відповідь «*answer*» на отриману пропозицію за допомогою методу *createAnswer()*. Локальний опис для об'єкту *peerConnection* оновлюється з відповіддю «*answer*». Сервер відправляє клієнту відповідь «*answer*» разом із кандидатами *ICE*, які він може використовувати для встановлення з'єднання. Ця інформація обробляється на клієнтській стороні для подальшої взаємодії між клієнтом і сервером через *WebRTC*:

```
io.on('connection', (socket: any) => {
  console.log('Client connected with ID: ' + socket.id)
  socket.emit('status', 'Socket was connected')
  socket.on('offer', async (offer: any, iceCandidates: any) => {
    await peerConnection.setRemoteDescription(offer)
    const answer = await peerConnection.createAnswer()
    await peerConnection.setLocalDescription(answer)
    socket.emit('answer', { answer, serverIceCandidates })})
```

В свою чергу клієнт створює з'єднання з сервером за допомогою функції *io()*, яку надає бібліотека *Socket.IO*. Дана функція приймає в якості параметрів адресу сервера з яким відбуватиметься з'єднання та об'єкт конфігурації. Об'єкт

конфігурації містить параметр «*transports: [‘websocket’]*», який вказує, що спроби з’єднатися з сервером мають використовувати протокол *WebSocket*, котрий дозволяє встановлювати двостороннє з’єднання в реальному часі між клієнтом та сервером; параметр «*upgrade: false*», який вказує що не слід пробувати автоматично оновлювати з’єднання до *WebSocket*:

```
const socket = io('http://localhost:3001', {
  transports: ['websocket'],
  upgrade: false,
})
```

Коли користувач, після того як ввів дані трансляції, натискає на кнопку «Додати трансляцію» спрацьовує обробник подій «*handleClickSendStream*». Даний обробник викликає функцію *createOffer()*, який створює пропозицію для встановлення з’єднання і повертає об’єкт *RTCSessionDescription*, який містить цю пропозицію. Далі встановлюється локальний опис для з’єднання за допомогою функції *setLocalDescription()*. Клієнт відправляє серверу подію «*offer*», яка містить об’єкт *RTCSessionDescription* та об’єкт *iceCandidates*:

```
const handleClickStream = async () => {
    const offer = await peerConnection.createOffer()
    await peerConnection.setLocalDescription(offer)
    socket.emit('offer', { offer, iceCandidates })
}
```

Після того, як сервер створить та відправить відповідь клієнту, спрацює обробник подій «*answer*», який отримає об’єкти *answer* та *serverIceCandidates*. Далі встановлюється віддалений опис *SDP* для віддаленого вузла. Потім код проходить через масив *serverIceCandidates* та додає кожен кандидат до об’єкта *peerConnection* за допомогою методу *addIceCandidate()*:

```
socket.on('answer', async ({ answer, serverIceCandidates }) => {
    await peerConnection.setRemoteDescription(answer)
    iceCandidates.forEach((c) => peerConnection.addIceCandidate(c))})
```

3.3. Організація передачі медіа потоку

Після створення двонаправленого з'єднання між клієнтом і сервером, клієнтська частина сервісу підключає медіа пристрої (дисплей, камера, мікрофон). Для цього використовується асинхронна функція *enableStream()*, яка викликається в *React* функції *useEffect()*. Спочатку вона викликає метод *getDisplayMedia()* з екземпляру класу *navigator.mediaDevices* для отримання відео і аудіо потоку з екрану користувача, або метод *getUserMedia()* для отримання відповідних потоків з камери і мікрофону. Якщо отримання потоку вдалося, то він встановлюється за допомогою функції *setLocalStream()*. Далі, всі треки з отриманого потоку додаються до об'єкту *peerConnection*, для передачі їх на сервер. Також виконується перевірка, чи локальний потік ще не був створений. Якщо умова виконується, то викликається функція *enableStream()* для створення потоку. Якщо умова не виконується, то викликається функція *cleanup()*, яка отримує всі треки потоку функцією *getTracks()* та зупиняє кожного з них функцією *stop()*. Це є важливим для вивільнення ресурсів і уникнення проблем з пам'яттю:

```
React.useEffect(() => {
  async function enableStream() {
    try {
      const stream = await navigator.mediaDevices.getDisplayMedia({
        video: true,
        audio: true,
      })
      setLocalStream(stream)
      stream
        .getTracks()
        .forEach((track) => peerConnection.addTrack(track, stream))
    } catch (error) {
      console.warn(error)
    }}
```

```
  if (!localStream) {
    enableStream()
  } else {
    return function cleanup() {
      localStream.getTracks().forEach((track: any) => {
        track.stop()
})}}})
```

У свою чергу у серверній частині сервісу створюються нові трансивери для аудіо і відео даних у межах *peerConnection*. Також створюються об’єкти класів *RTCAudioSink* і *RTCVideoSink* для обробки аудіо і відео даних, які були отримані через трансивери. Всі отримані потоки зберігаються в масиві *stream*:

```
const audioTransceiver = peerConnection.addTransceiver('audio')
const videoTransceiver = peerConnection.addTransceiver('video')
const audioSink = new RTCAudioSink(audioTransceiver.receiver.track)
const videoSink = new RTCVideoSink(videoTransceiver.receiver.track)
const streams: any = []
```

Після збереження отриманих потоків виконується їх обробка за допомогою об’єкта *videoSink* класу *RTCVideoSink*. Для цього встановлюється обробник подій для події frame, котрий викликає функцію, що містить властивості: *width*, *height* і *data*. Далі визначається розмір кадру, який об’єднує ширину та висоту, і записується у змінну *size*. Перевіряється, чи існує перший потік в масиві *streams*, або чи розмір поточного кадру відрізняється від розміру першого потоку. Якщо умова вище виконується, тоді створюється новий об’єкт *stream*. Властивість *size* містить розмір кадру, а властивості *video* та *audio* представляють потокові об’єкти *PassThrough*. *PassThrough* є реалізацією потокового інтерфейсу у *Node.js*, що дозволяє передавати дані через потік без будь-якої обробки:

```
videoSink.addEventListener(
  'frame',
  ({ frame: { width, height, data } }) => {
    const size = width + 'x' + height
```

```
if (!streams[0] || (streams[0] && streams[0].size !== size)) {
  const stream = {
    size,
    video: new PassThrough(),
    audio: new PassThrough(),
  }
```

Після цього визначається функція *onAudioData*, яка викликається при отриманні аудіо даних. Параметр функції містить об’єкт з властивістю *samples*, яка в свою чергу містить масив аудіо зразків (*samples*). У даному випадку, масив аудіо-зразків містить властивість *buffer*, яка представляє собою буфер аудіо даних. Перевіряється, чи аудіо-потік ще не завершено, і якщо умова виконується, то дані з буфера аудіо додаються до потоку аудіо *stream.audio* за допомогою методу *push*. Додається слухач подій для об’єкта *audioSink* на подію *data*. Коли відбувається ця подія, буде викликана раніше визначена функція *onAudioData*. Додається обробник події *end* до аудіо потоку *stream.audio*. Цей обробник видаляє слухача подій *onAudioData* з об’єкта *audioSink*, коли аудіо потік завершується. Додається новий потік на початок масиву *streams*:

```
const onAudioData = ({ samples: { buffer } }) => {
  if (!stream.audio.end) {
    stream.audio.push(Buffer.from(buffer))
}}
audioSink.addEventListener('data', onAudioData)
stream.audio.on('end', () => {
  audioSink.removeEventListener('data', onAudioData)
})
streams.unshift(stream)
```

Далі виконується ітерація по масиву *streams* за допомогою функції *forEach()*, який перебирає кожен елемент масиву. Для кожного такого елементу виконується перевірка, чи поточний елемент не є тим самим потоком, який вже був оброблений, і чи цей потік не є завершеним. Якщо умова виконується, то встановлюється флаг *end*

для позначення того, що потік був завершений. Якщо у поточного елемента існує аудіо потік, то для цього аудіо-потоку викликається метод *end()*, котрий завершує аудіо потік. Також даний метод викликається для відео потоку поточного елемента:

```
streams.forEach((item: any) => {
  if (item !== stream && !item.end) {
    item.end = true
    if (item.audio) {
      item.audio.end()
    }
    item.video.end()
}})
```

Для передачі медіа даних на платформу створюється новий об’єкт класу *FFmpeg*. До нього додається вхідна інформація для відео потоку, яка міститься в об’єкті *StreamInput*. Далі додаються параметри для вхідного потоку відео, такі як формат вхідного потоку, піксельний формат, розмір і частота кадрів. Аналогічно до відео, додається вхідна інформація для аудіо-потоку, використовуючи об’єкт *StreamInput*. Додаються параметри для вхідного потоку аудіо, включаючи формат аудіо, частоту дискретизації та кількість каналів. Після цього зазначається *URL* адреса, куди будуть надсилатися оброблені медіа дані:

```
stream.proc = ffmpeg()
  .addInput(new StreamInput(stream.video))
  .addInputOptions([
    '-f', 'rawvideo', '-pix_fmt', 'yuv420p', '-s', stream.size, '-r', '30'])
  .addInput(new StreamInput(stream.audio).url)
  .addInputOptions(['-f s16le', '-ar 48k', '-ac 1'])
  .output(url + '/' + key)
stream.proc.run()
```

Наприкінці, використовується метод *push()*, який призначений для додавання нового елемента до відео-потоку. У цьому випадку, новим елементом є буфер даних,

який отриманий з параметру *data*. Функція *Buffer.from(data)* створює новий буфер, використовуючи передані дані:

streams[0].video.push(Buffer.from(data))

Коли користувач завершує трансляції спрацьовує обробник події *disconnect*. В даному обробнику викликається метод *leave()* для об'єкта *socket*. Цей метод призначений для клієнта з групи сокетів до якої він є приєднаним:

```
socket.on('disconnect', () => {
  socket.leave()
})
```

3.4. Робота браузерного сервісу

Для запуску браузерного сервісу необхідно відкрити веб-браузер, і в адресному рядку ввести *URL*-адресу: *http://localhost:3000*.

Після запуску браузерного сервісу необхідно надати дозвіл веб-браузеру на використання медіа пристроїв користувача. Після цього відобразиться головна сторінка сервісу (рис. 3.5).

На даній сторінці користувач може попередньо обрати пристрій для запису трансляції. Далі він може додати платформи на яких будуть проводитись трансляції. Для цього необхідно натиснути на кнопку «Додати трансляцію» і в новому модальному вікні ввести: назву, *URL* і ключ трансляції. Після натискання на кнопку «Розпочати трансляцію» відкриється вікно налаштувань трансляції (рис. 3.6).

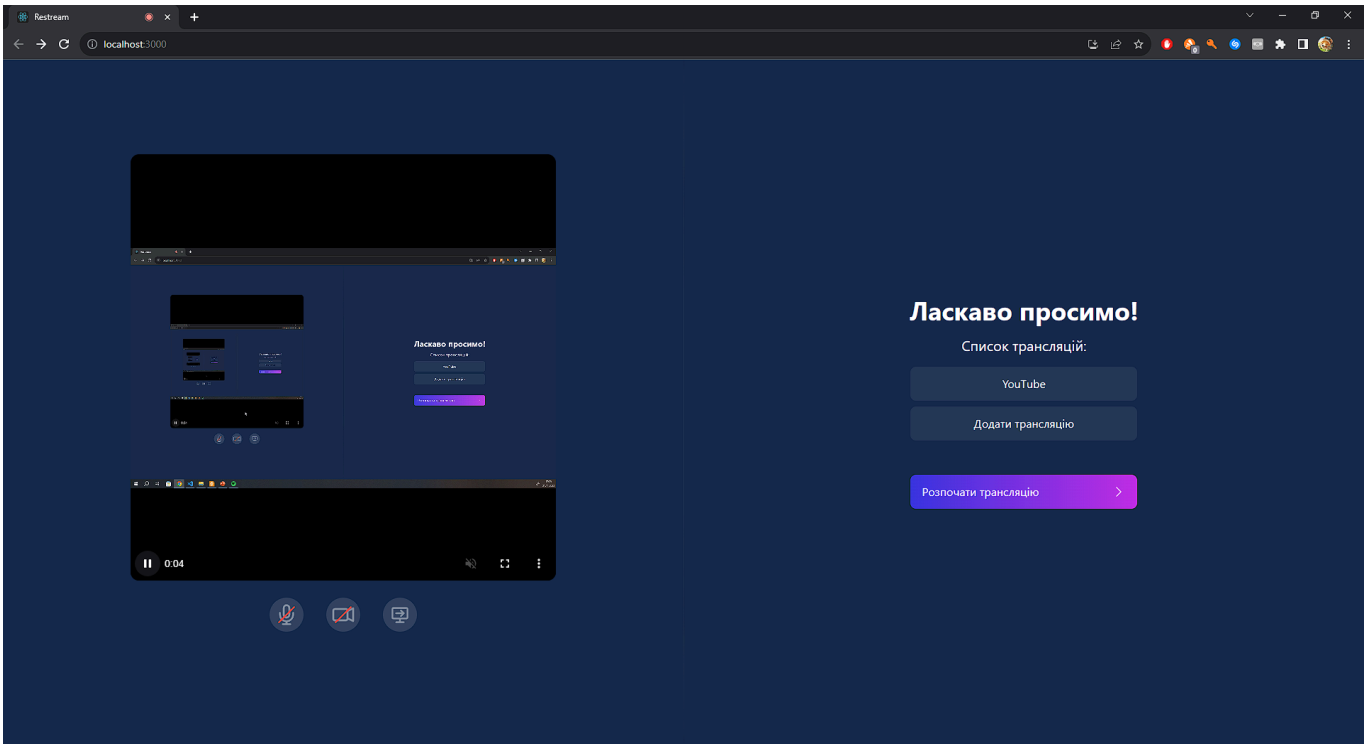

Рис. 3.5. Головне вікно браузерного сервісу

На даній сторінці відображається трансляція, яка буде передана на платформи. В нижній частині вікна користувач може змінити пристрій для запису трансляції, або завершити її. В правій частині вікна можна додати, змінити чи видалити пункт призначення трансляції.

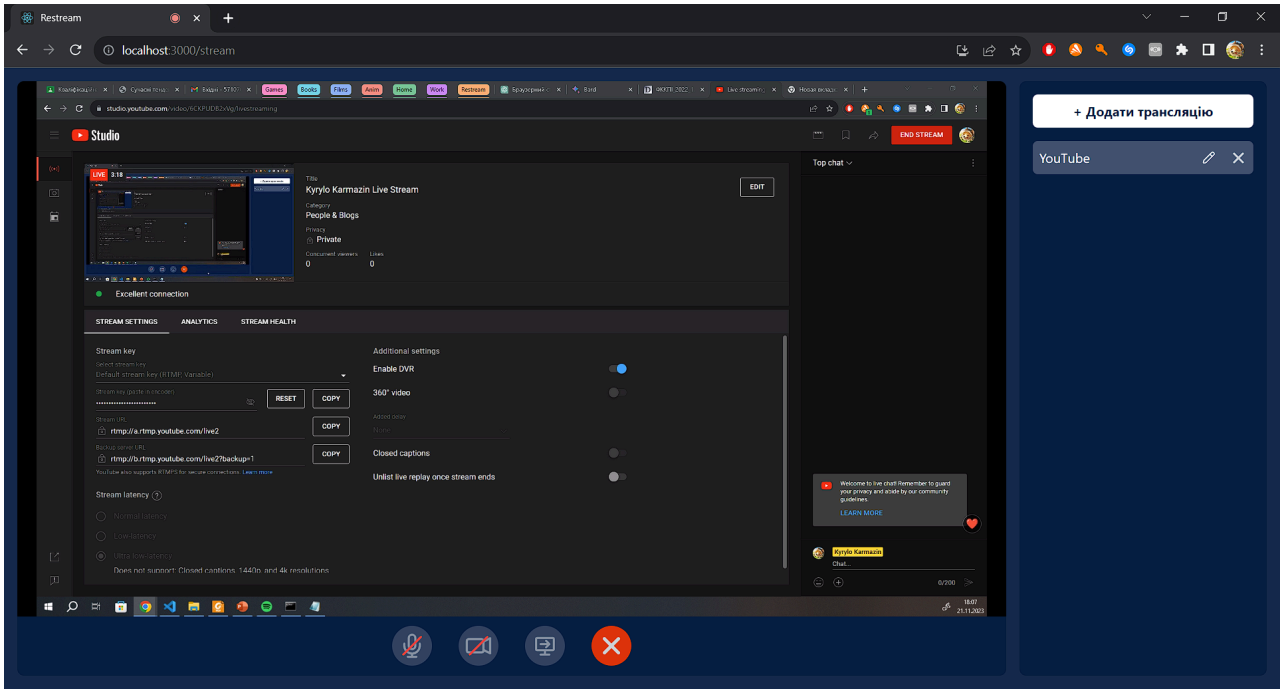

Рис. 3.6. Вікно налаштувань трансляції

Для того, щоб додати новий пункт призначення необхідно натиснути на кнопку «Додати трансляцію», після чого в модальному вікні, що з’явилося, ввести дані платформи (рис. 3.7).

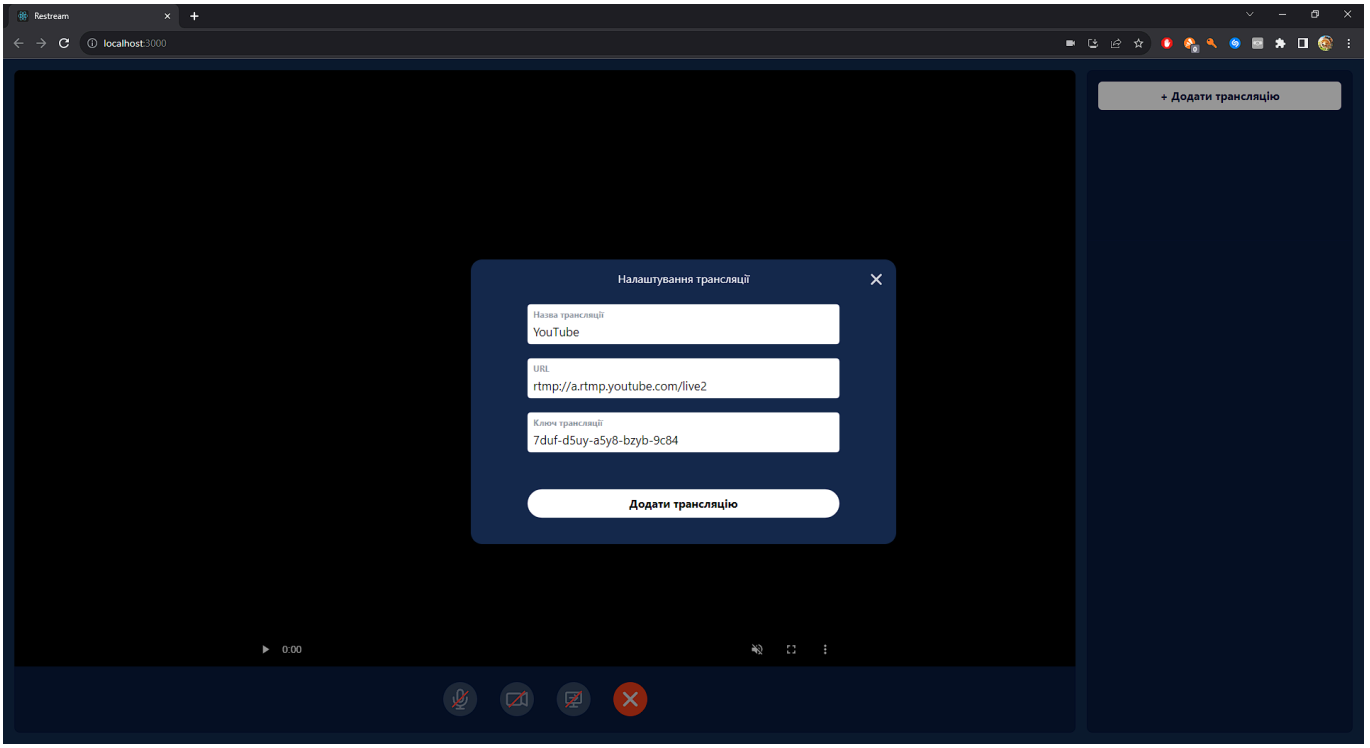

Рис. 3.7. Вікно створення пункту призначення

Для того, щоб отримати трансляцію на певній платформі, потрібно попередньо зареєструватися на ній та обрати в якості джерела трансляції зовнішнє джерело. Наприклад, на платформі *YouTube* необхідно спершу натиснути кнопку «*Create – Go live*» після чого в новому вікні обрати пункт «*External source*». Далі відкриється вікно налаштувань, де можна отримати *URL*-адресу і ключ трансляції. При додаванні нової трансляції в браузерному сервісі вводимо отримані дані у відповідні поля. Після цього платформа сповістить користувача про отримання потоку медіа даних від користувача і розпочне миттєво пряму трансляцію (рис. 3.8).

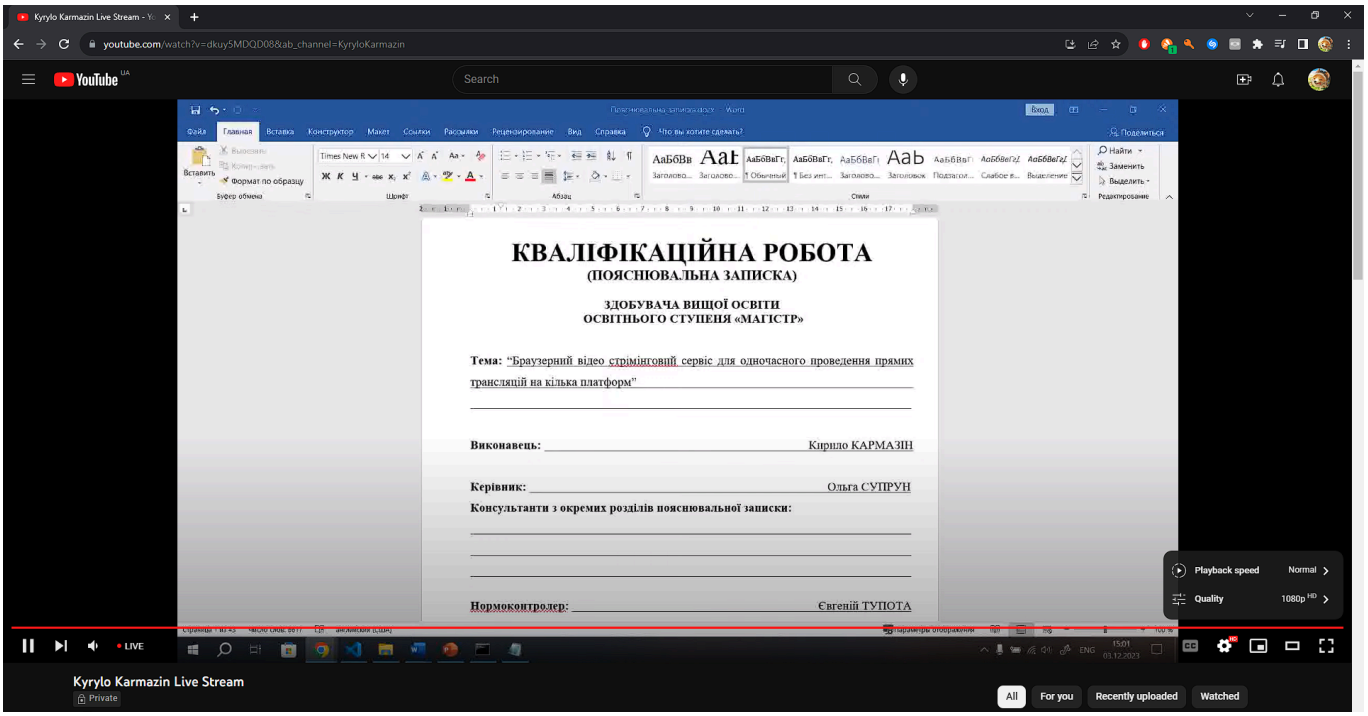

Рис. 3.8. Вікно трансляції на платформі *YouTube*

Для завершення трансляції в браузерному сервісі необхідно натиснути червону кнопку «Завершити трансляцію».

3.5. Висновки до розділу

У третьому розділі було описано структуру браузерного сервісу для одночасного проведення трансляцій на кілька платформ. В даному сервісі було використано такі технології для передачі медіа даних, як: *WebRTC*, *Socket.IO* і *FFmpeg*. Для розробки сервісу було використано середовище *Visual Studio Code*.

В межах сервісу було створено головну сторінку та сторінку для налаштувань трансляції. На головній сторінці користувач має можливість попередньо обрати пристрій для запису трансляції, а також обрати платформи куди будуть передаватися медіа дані. На сторінці налаштувань є можливість змінити пристрій для запису, а також додати, редагувати чи видалити пункт призначення трансляції.

Однією з важливих технологій, використаних у розробці, є *WebRTC*. Його використання дозволило забезпечити низьку затримку та високу якість відео- та аудіопередачі між клієнтом та сервером. Завдяки механізмам шифрування та

розвиненим алгоритмам управління потоками, *WebRTC* гарантує безпеку та стабільність трансляцій. За допомогою *WebRTC*, під час розробки було створено об'єкти *RTCPeerConnection* для забезпечення *P2P*-з'єднань між клієнтами. Також було визначено обробники для обміну медіа потоками та керуванням відправкою і прийманням даних.

Socket.IO відіграв важливу роль у забезпеченні двостороннього зв'язку між клієнтом та сервером. Його застосування дозволяє в реальному часі передавати інформацію про події та стан системи, що є ключовим для ефективної координації між учасниками та моніторингу стану трансляцій. При розробці на стороні сервера було створено сокетний сервер, який слухатиме підключення від клієнтів. Також було створено події, які клієнти можуть викликати та обробляти на сервері, і навпаки. Наприклад, події «*offer*» і «*answer*», щоб обмінюватися кандидатами *ICE* та описами з'єднання. *Socket.IO* автоматично обробляє ретрансляцію подій між всіма підключеними клієнтами.

Для забезпечення безпеки з'єднань при використанні цих технологій було використано *HyperText Transfer Protocol Secure*, який дозволяє забезпечувати шифрування даних та автентифікацію користувачів.

FFmpeg використовується для інтеграції з *RTMP*-платформою та забезпечення потокового відео на різні платформи. Його висока продуктивність та розширювані можливості конвертації форматів дозволяють легко взаємодіяти з різними системами і забезпечувати оптимальну якість трансляцій.

Для створення інтерфейсу користувача було використано бібліотеку *Tailwind*. Дана бібліотека дозволяє швидко створювати інтерфейс, оскільки розробнику не потрібно постійно переключатися між файлами *CSS* і *HTML*. Є можливість визначати стилі прямо в *HTML*-коді за допомогою класів, що робить процес розробки ефективнішим. *Tailwind* пропонує консистентні ім'я класів, які відображають саму сутність стилю [22]. Також *Tailwind* пропонує зручні засоби для роботи з адаптивним дизайном, що дозволяє використовувати класи для приховування, зміни розмірів або зміни стилів на різних екранах та пристроях [15].

Середовище розробки *Visual Studio Code* виявилося ідеальним інструментом для створення браузерного сервісу. Відмінна підтримка *JavaScript*, зручний інтерфейс та багатий функціонал забезпечили ефективний процес розробки та тестування [23].

Також був описаний алгоритм встановлення зв’язку між клієнтом і сервером, та алгоритм передачі медіа даних від сервера на платформу. Алгоритм встановлення зв’язку між клієнтом і сервером використовує технології, такі як *WebSocket*, для стабільного та ефективного забезпечення взаємодії. Наприкінці, було надано результат роботи браузерного сервісу.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено браузерний відео стрімінговий сервіс для одночасного проведення прямих трансляцій на кілька платформ, який використовує фреймворк *FFmpeg* для обробки та передачі медіа потоку на платформу.

По-перше, було проаналізовано предметну область «потокове мультимедіа», включаючи порівняльний аналіз і огляд аналогічних стрімінгових сервісів, таких як *YouTube*, *Twitch*, *Streamlabs* та *Restream*. Для кожного було надано опис їх функціональності.

Аналізуючи концепцію мультимедіа, були визначені основні компоненти, такі як текст, зображення, звук і відео, які комбінуються для формування комплексного інформаційного контенту. Ця інтеграція різних медійних форматів розширює можливості сприйняття і розуміння інформації користувачами. При розгляді областей використання потокового мультимедіа було встановлено його значний потенціал у різних сферах, від розваг до освітніх та корпоративних заходів.

Особлива увага була приділена областям використання потокового мультимедіа, що є необхідною частиною сучасного цифрового оточення. Зокрема, висвітлено важливість стрімінгових сервісів у різних сценаріях, від розваг та ігор до освіти та бізнесу. Використання стрімінгових сервісів стає все більш популярним, дозволяючи не лише отримувати інформацію, але і активно взаємодіяти з нею у режимі реального часу.

Однією з ключових аспектів потокового мультимедіа є ефективне кодування та стиснення даних. Це призводить до зменшення обсягу даних, які передаються, що має важливе значення для забезпечення безперебійної передачі контенту при обмежених швидкостях Інтернет-з’єднання. Також важливим аспектом є використання відповідних протоколів. *HTTP Live Streaming* та *Dynamic Adaptive Streaming over HTTP* є популярними протоколами, які дозволяють автоматично адаптувати якість трансляції відповідно до швидкості Інтернет-з’єднання користувача.

Було встановлено, що ефективність передачі потокового мультимедіа залежить від інфраструктури сервера. Великі платформи використовують мережу доставки контенту *Content Delivery Network* для глобального розподілу матеріалів, що забезпечує швидкий доступ користувачам у різних частинах світу.

Також проведено аналіз концепції стрімінгових сервісів, виокремлені основні принципи їх реалізації та визначено різноманітні методи забезпечення високоякісних трансляцій. Особлива увага була приділена таким платформам, як *YouTube*, *Twitch*, *Restream* та *Streamlabs*, які є найбільш популярними у сфері потокового мультимедіа. *YouTube* характеризується великою аудиторією та інтеграцією з іншими сервісами *Google*, *Twitch* спеціалізується на розважальному контенті, *Restream* дозволяє одночасно транслювати на кілька платформ, а *Streamlabs* забезпечує ведучих розширеними інструментами. За аналізом можна зробити висновок, що ці сервіси надають користувачам різноманітні інструменти для одночасного проведення прямих трансляцій.

Під час аналізу існуючих стрімінгових сервісів було детально розглянуто їх можливості, переваги та недоліки. Це дозволило визначити оптимальний вибір функцій та технологій для здійснення трансляцій на різні платформи, з урахуванням потреб та вимог користувачів.

По-друге, було детально розглянуто технології створення браузерного сервісу. Визначено функції та роль технологій потокової передачі мультимедіа в реальному часі, таких як *WebRTC* та *Socket.IO*. Обрано фреймворк *FFmpeg*, оскільки він є у вільному доступі і надає широкий спектр можливостей для обробки потокового мультимедіа.

Також визначено платформи для побудови клієнтської та серверної частини браузерного сервісу, а саме: *JavaScript*-бібліотеку *React.js* і платформу для створення серверів *Node.js*, котрі є популярними та ефективними серед інших у розробці сервісів даного типу.

Основною бібліотекою для розробки інтерфейсу користувача був обраний *React.js*. Його можливості у створенні ефективних та легко підтримуваних компонентів забезпечили високу продуктивність та зручність у взаємодії з

користувачем. Декларативний підхід *React.js* спрощує процес розробки великих та складних додатків. *Virtual DOM* дозволяє ефективно оновлювати інтерфейс, що є ключовим для забезпечення відмінного користувацького досвіду.

Node.js використовується як платформа для сервера, що надає можливість використовувати *JavaScript* для створення серверної частини додатка. Його асинхронний підхід та використання єдиного потоку для обробки багатьох одночасних з'єднань дозволяють створювати швидкі та ефективні серверні додатки, сприяють швидкості обробки запитів і реалізації масштабованих додатків.

Redux був використаний для ефективного управління станом додатка, допомагаючи уникнути проблем, що виникають при роботі зі складністю стану в громіздких додатках. Однонапрямлений потік даних та централізоване зберігання стану роблять код більш передбачуваним і легшим для супроводження. Можливість використання *Middleware* в *Redux* дозволяє розширювати його функціонал, наприклад, для обробки асинхронних операцій.

Бібліотеку *Socket.IO* призначена для створення багаторівневих з'єднань між клієнтом і сервером через протокол *WebSocket*. Його впроваджено для забезпечення взаємодії між клієнтом і сервером в режимі реального часу. Функціонал *Socket.IO* дозволяє миттєво передавати дані та ефективно обробляти події, що є критичним для багатьох сучасних додатків, таких як чати, групові трансляції та конференції.

В розробленому браузерному сервісі *WebRTC* використовувався для підтримки передачі аудіо та відеоданих в режимі реального часу. Його масштабованість і висока якість забезпечують комфорт та ефективність взаємодії через веб-браузер. Крім того, *WebRTC* забезпечує низьку затримку та високу якість трансляцій, що робить його ключовим для веб-додатків, що працюють у режимі реального часу з медіа даними. *API WebRTC* надає можливість отримання доступу до медіа пристроїв користувача, таких як камера і мікрофон. Для успішного встановлення з'єднання *WebRTC* може використовувати сервери *Session Traversal Utilities for NAT* та *Traversal Using Relays around NAT*. *STUN* допомагає виявляти публічні *IP*-адреси та порти пристрою, а *TURN* використовується у випадку, якщо пряме з'єднання між пристроями неможливе.

Фреймворк *FFmpeg* використовується для обробки, конвертації та передачі мультимедійних даних на платформу. Завдяки його розширеному функціоналу можливо забезпечити високий рівень обробки відео та аудіо в рамках сервісу. *FFmpeg* може працювати з різними форматами файлів і виконувати різноманітні операції, такі як зміна розміру, зміна частоти кадрів, застосування фільтрів, кодування та декодування аудіо та відео, і багато інших. Підтримка *FFmpeg* безлічі кодувальників і форматів робить його потужним інструментом для роботи з мультимедійними даними.

Загалом було розглянуто технології, які взаємодіючи один о одним, дозволяють створювати сучасні браузерні сервіси з високою продуктивністю та можливістю легкої розширюваності. Використання *React.js* та *Node.js* ефективно поєднує клієнтську та серверну розробку, а використання *Redux* забезпечує ефективне управління станом додатку. *Socket.IO* та *WebRTC* додають можливість реалізації режиму реального часу та багатокористувацької взаємодії. Підтримка звукової та відеоінформації надається за допомогою *FFmpeg*, що робить цей комплект технологій ідеальним для розробки браузерного відео стрімінгового сервісу для одночасного ведення прямих трансляцій на різні платформи.

По-третє, було досліджено алгоритми створення двонаправленого зв’язку між клієнтом і сервером, та алгоритм передачі медіа даних на платформи. При цьому було надано результат роботи сервісу, який містить детальний опис дій разом із знімками екрану користувача.

У межах сервісу було створено головну сторінку та сторінку налаштувань трансляції. На головній сторінці користувач має можливість перед трансляцією попередньо обрати пристрій для запису та визначити платформи, куди будуть передаватися медіа дані. Сторінка налаштувань надає можливість змінити пристрій для запису та додати, редагувати чи видалити призначення трансляції.

У ході розробки був створений браузерний сервіс, який виконує наступні завдання:

– створення двонаправленого зв’язку;
– створення з’єднання для потокової передачі мультимедіа;

- попередній вибір пристрою запису трансляції і платформ;
- редагування чи видалення вже існуючих пунктів призначення трансляції;
- можливість одночасної передачі потокового мультимедіа на кілька платформ;
- можливість проведення трансляції з камери і мікрофону користувача;
- можливість демонстрації дисплею користувача.

Максимальна кількість трансляцій, які здатен проводити розроблений браузерний сервіс, залежить від декілька факторів, а саме: швидкість і пропускна спроможність мережі; продуктивність і потужність сервера; наявність багатозадачності і оптимізованих алгоритмів обробки відео даних; наявність достатньої кількості процесорів і оперативної пам’яті.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойчено С. В., Іванченко О. В., «Положення про дипломні роботи (проекти) випускників Національного авіаційного університету», СМЯ НАУ П 03.01(10)-02-2017.

2. ДСТУ 3008–95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення».

3. ДСТУ *ISO* 5807:2016 «Оброблення інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів».

4. ГОСТ 7.11–2004 «Система стандартів по інформації, бібліотечній та видавничій справі. Бібліографічний запис. Скорочення слів та словосполучень на іноземних та європейських мовах».

5. ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання».

6. ДСТУ 3582: 2013 «Бібліографічний опис скорочення слів і словосполучень в українській мові».

7. *Vivian Walker*, *React JS*: *From Basics to Advanced - A Comprehensive* 3-*in*-1 *Guide to Effortless Web Development for Beginners, Intermediates*, *and Experts. – Publisher*: *Amazon Kinlde*, 2023. – 391 *pages.*

8. *Sebastian Springer*, *Node.js*: *The Comprehensive Guide to Server-Side JavaScript Programming*. *– Publisher*: *Rheinwerk Computing*, 2023. – 834 *pages*.

9. *Beny Rahman Hakim*, *Create, Deploy & Evaluate A WebRTC Video Conference*, 1*st edition*. *– Publisher*: *UD*. *Duta Sablon*, 2023. – 237 *pages*.

10. *Nick Ferrando*, *FFMPEG – From Zero to Hero*. *– Publisher*: *Wondermark Books*, 2023. – 164 *pages*.

11. *Josh Fischer*, *Ning Wang*, *Grokking Streaming Systems*: *Real-time event processing*. *– Publisher*: *Manning*, 2022. – 312 *pages*.

12. *Theophilus Siameh*, *Stream Processing with Apache Pulsar*: *Building real-time scalable data streaming applications*. – *Publisher*: *Independently published*, 2021. – 370 *pages*.

13. *David DuRocher*, *HTML and CSS QuickStart Guide*: *The Simplified Beginners Guide to Developing a Strong Coding Foundation*, *Building Responsive Websites*, *and Mastering Web Design*. – *Publisher*: *ClydeBank Media LLC*, 2021. – 352 *pages*.

14. *Philip Ackermann*, *JavaScript*: *The Comprehensive Guide to Learning Professional JavaScript Programming*, 1*st edition*. – *Publisher*: *Rheinwerk Computing*, 2022. – 982 *pages*.

15. *Jenifer Tidwell*, *Charles Brewer*, *Aynne Valencia*, *Designing Interfaces*: *Patterns for Effective Interaction Design*, 3*rd edition*. – *Publisher*: *O'Reilly Media*, 2020. – 599 *pages*.

16. Офіційна документація *React.js* [Електронний ресурс] 2023 *URL*: *https://uk.legacy.reactjs.org/docs/getting-started.html*

17. Офіційна документація *Node.js* [Електронний ресурс] 2023 *URL*: *https://nodejs.org/en/docs*

18. Офіційна документація *WebRTC* [Електронний ресурс] 2023 *URL*: *https://webrtc.org/getting-started/overview*

19. Офіційна документація *Socket.IO* [Електронний ресурс] 2023 *URL*: *https://socket.io/docs/v4*

20. Офіційна документація *FFmpeg* [Електронний ресурс] 2023 *URL*: *https://ffmpeg.org/documentation.html*

21. *Express – Node.js web application framework* [Електронний ресурс] 2023 *URL*: *https://expressjs.com*

22. *Installation – Tailwind CSS* [Електронний ресурс] 2023 *URL*: *https://tailwindcss.com/docs/installation*

23. *Documentation for Visual Studio Code* [Електронний ресурс] 2023 *URL*: *https://code.visualstudio.com/docs*

24. *TypeScript*: *The starting point for learning TypeScript* [Електронний ресурс] 2023 *URL*: *https://www.typescriptlang.org/docs*

25. *Redux Essentials*, *Part* 1: *Redux Overview and Concepts* [Електронний ресурс] 2023 *URL*: *https://redux.js.org/tutorials/essentials/part-1-overview-concepts*