МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра програмування
(повна назва кафедри)

Магістерська робота

Виконав: студент групи ПМІм-21
спеціальності
122-"Комп'ютерні науки"
(шифр і назва спеціальності)

Садовський Д.В.
(підпис) (прізвище та ініціали)

Керівник Музичук А. О.
(підпис) (прізвище та ініціали)

Консультант Нобіс В.В.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

ДЕКАН
факультету прикладної математики та інформатики
ЛНУ імені Івана Франка

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *програмування*

Освітньо-кваліфікаційний рівень *магістр*

Спеціальність *122 – комп'ютерні науки*

«ЗАТВЕРДЖУЮ»

Зав. кафедрою *доц. Ярошко С.А.*

« *13* » *вересня* 2022 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Садовський Даніїл Віталійович
(прізвище, ім'я, по батькові)

1. Тема роботи
Розробка браузерного розширення для контролю перевірки знань студентів

керівник роботи *доц. Музичук А.О.*
затверджена Вченою радою факультету від « *13* » *вересня* 20 *22* р., № *15*

2. Строк подання студентом роботи *12.12.2022 р.*

3. Вихідні дані до роботи
Література та інтернет-ресурси за тематикою роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Проаналізувати сучасний стан проблеми. Вибрати технології, а саме: базу данних, бібліотеку для створення UI інтерфейсу, фреймворк для створення Backend частини, протокол для трансляції медіа потоку. Створити UI для роботи викладача. Реалізувати браузерне розширення за допомогою Chrome API. Імплементувати сервер для обробки запитів. Встановити зв'язок між двома абонентами використовуючи вибраний протокол. Дати змогу

викладачу обмежити і контролювати доступ студентів до відповідних ресурсів.
Проаналізувати та перевірити виконання роботи відповідно до очікуваного результату.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Схеми, діаграми
Презентація дипломної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *05.09.22 р.*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Постановка задачі*	*Вересень*	
2.	*Огляд існуючих систем та технологій*	*Жовтень*	
3.	*Програмна реалізація*	*Листопад*	
4.	*Оформлення роботи*	*Грудень*	

Студент _____ *Садовський Д.В.*
підпис

Керівник роботи _____ доц. Музичук А.О.
підпис

ЗМІСТ

ВСТУП

На сучасному етапі економічного розвитку висуваються особливі вимоги до підготовки спеціалістів на всіх рівнях професійної освіти. Вони мають відповідати вимогам міжнародних стандартів якості, задовольняти запити всіх споживачів відповідних послуг.

Якість молодих фахівців, що випускаються, визначається, перш за все, глибиною і міцністю професійних знань, умінь застосовувати їх на практиці, а також набуттям конкретних навичок, які характеризують практичну цінність працівника, який вміє виконувати функції, визначені кваліфікаційною характеристикою спеціаліста. Відповідність випускників даним вимогам стала особливо актуальним нашого часу у ситуації жорсткої конкуренції ринку праці. У молодих людей, які не мають досвіду роботи за спеціальністю, виникає необхідність довести свою компетентність, сформованість професійних умінь та навичок, здатність працювати в даній сфері, вирішувати виробничі завдання.

Однією з основних умов міцних та глибоких знань, стійких умінь та навичок є постійний та цілеспрямований контроль діяльності студентів, у процесі якого вони привчаються працювати систематично, а це вже виробляє звичку вдосконалюватись і вчитися далі, сприяє вихованню дисциплінованості.

Питання контролю, оцінювання знань, умінь завжди стояли перед педагогічною наукою. Нині ця проблема не втрачає своєї актуальності. Усі останні документи, спрямовані на модернізацію освіти, покликані забезпечити підвищення якості освіти загалом, і, зокрема, підвищення якості підготовки спеціалістів. Повсюдно усвідомлюється необхідність удосконалення методів та засобів оцінки, як освітніх програм, так і результатів їх освоєння.

Метою моєї роботи є розробка системи на базі браузерного розширення для контролю та перевірки справедливих знань студента під час роботи онлайн. Буде створено “клієнт-сервіс” архітектуру для взаємодії і контролю роботи студентів викладачем, а також ядром нашої системи буде браузерне розширення, яке дозволятиме контролювати статус взаємодії студента з певним списком сервісів, який налаштовується безпосередньо викладачем до початку тестування знань.

Також має зміст транслювати робочий стіл студентів під час виконання робіт з метою перевірки і контролю знань, оскільки це являється додатковим прошарком для впевненості в справедливості знань студента і обмежує його від застосування інтернет ресурсів для швидкого пошуку відповідей та отримання фальшивих знань, які лише розширюють дисперсію під час аналізу результатів виконання робіт. В кінці ми зможемо проаналізувати переваги і недоліки данної реалізації і зробити певні висновки.

РОЗДІЛ 1. ВИБІР БРАУЗЕРА ТА ВЕРСІЇ API

Було вирішено використовувати браузер Chrome, оскільки він являється одним з найпоширенніших та має свої переваги над іншими браузерами. Він вважається досить швидким, гнучким в контексті різних застосунків та розширень, має регулярні оновлення та вважається досить стабільним. Проаналізувавши документацію [1] версії від 28.09.2022 прийнято рішення щодо створення розширення на основі Manifest V3. Manifest V3 додає нові правила, які обмежують здатність розширення виконувати неперевірений JavaScript через поєднання змін платформи та обмежень політики. Також існує ряд нових функцій і функціональних змін для розширень, які використовують Manifest V3:

1) Сервісні працівники (service workers) замінюють фонові сторінки (background pages) .

2) Модифікація мережевого запиту (network request modification) тепер обробляється за допомогою нового API **declarativeNetRequest**.

3) Віддалено розміщений код (remotely hosted code) більше не дозволяється; розширення може виконувати лише JavaScript, який включено до його пакета.

4) До багатьох методів додано підтримку Promise, хоча зворотні виклики (callbacks) все ще підтримуються як альтернатива.

РОЗДІЛ 2. ВИБІР ПРОТОКОЛУ ПЕРЕДАЧІ ДАНИХ

2.1 WebSocket

WebSocket [2] — незалежний веб-протокол, який дозволяє створювати інтерактивне з'єднання між сервером і клієнтом (браузером) і обмінюватися повідомленнями в реальному часі. На відміну від HTTP, веб-сокети дозволяють працювати з двонаправленим потоком даних, тому технологія є унікальною.

2.1.1 Переваги

• Стандартизований Протокол, що означає можливість з допомогою організувати зв’язок між веб-серверами і клієнтами в режимі реального часу.

• Веб-сокети перетворюються в багатоплатформовий стандарт для зв’язку в реальному часі між клієнтом і сервером.

• Стандарт допускає новий вигляд додатків.

• За допомогою цієї технології підприємства, що працюють в режимі реального часу, можуть прискорити роботу.

• Найбільша перевага JavaScript WebSocket — це двосторонній зв’язок (повний дуплекс) по одному TCP-з’єднання. HTTP має свій власний набір схем, таких як http і https. Протокол веб-сокета також має аналогічну схему, визначену в його шаблоні URL. Остання специфікація протоколу WS визначається, як RFC 6455 – пропонований стандарт. RFC 6455 підтримується різними браузерами, такими як Internet Explorer, Mozilla Firefox, Google Chrome.

2.1.2 Як працює WebSocket

Для встановлення з'єднання веб-сокет застосовує метод відкриваючого рукостискання. Він полягає в тому, що клієнт випереджає відправлення/отримання повідомлень попереднім запитом, в якому клієнт і сервер «домовляються» використовувати веб-сокети. Це і є «рукостискання». Структура такого запиту

схожа на HTTP, але трохи відрізняється від нього. Потім клієнт та сервер обмінюються даними вже в рамках цього з’єднання.

2.1.3 Застосування WebSocket

WebSocket підходить, коли потрібні оновлення даних у реальному часі та можливість доставляти повідомлення клієнту. Декілька прикладів для WebSocket:

- торговельні програми з мінливістю котирувань, цін у реальному часі: платформи продажів, біржі;
- ігрові програми;
- чати, зокрема на сайтах підтримки;
- push-сповіщення;
- соціальні мережі;
- керування пристроями в IoT (використовується підпротокол WAMP).

Коли потрібно отримати незмінні дані, які витягуються лише один раз, щоб обробити їх додатком, краще використовувати протокол HTTP, а не WebSocket. Це може бути, наприклад, сторінка зі статтею. Після публікації стаття практично не змінюється, тому немає сенсу використовувати постійне з'єднання для її відображення.

Також HTTP-протокол краще, якщо ми не хочемо зберігати з'єднання протягом певного часу або повторно використовувати одне з'єднання для передачі даних. Це, наприклад, ситуація, коли сервер повинен віддати всі дані для форми однією відповіддю.

2.2 WebRTC

WebRTC [3] — це браузерна технологія, призначена для передачі потокових даних між браузерами або додатками з використанням технології двохточкової передачі (передача точка-точка). Це проект з відкритим вихідним кодом на основі

HTML-5 для зв'язку в режимі реального часу на основі браузера, що означає, що він забезпечує прямий зв'язок між браузерами без модулів, що підключаються, що значно спрощує обмін файлами і аудіо- і відеозв'язок для користувачів.

2.2.1 Переваги

- Не потрібне встановлення ПЗ.
- Висока якість зв'язку завдяки:
    1. використання сучасних відео- та аудіокодеків;
    2. автоматичне підстроювання якості потоку під умови з'єднання;
    3. вбудованій системі ехо- та шумозаглушення;
    4. автоматичне регулювання рівня чутливості мікрофонів учасників (АРУ).
- Високий рівень безпеки: всі з'єднання захищені та зашифровані згідно протоколів DTLS та SRTP. При цьому WebRTC працює тільки за протоколом HTTPS, а сайт, що використовує технологію, повинен бути підписаний сертифікатом.
- Підтримка технології SVC додана як частина реалізації кодеків VP9 та AV1. Незважаючи на те, що на даний момент все ще немає реалізації у самих браузерах, програмні рішення TrueConf дозволяють використання SVC у браузерних клієнтах.
- Існує вбудований механізм захоплення контенту, наприклад, робочого столу.
- Можливість реалізації будь-якого інтерфейсу керування на основі HTML5 та JavaScript.
- Проект з відкритим вихідним кодом можна впровадити у свій продукт або сервіс.
- Справжня крос-платформність: те саме WebRTC додаток буде однаково добре працювати на будь-якій операційній системі, десктопній або мобільній, за умови, що браузер підтримує WebRTC. Це значно економить ресурси для розробки ПЗ.

Також на наступному рисунку можна бачити діаграму, що відображає середнє

значення затримок WebRTC порівняно з іншими протоколами. Затримка з використанням WebRTC близька до мінімального значення затримки реального часу.

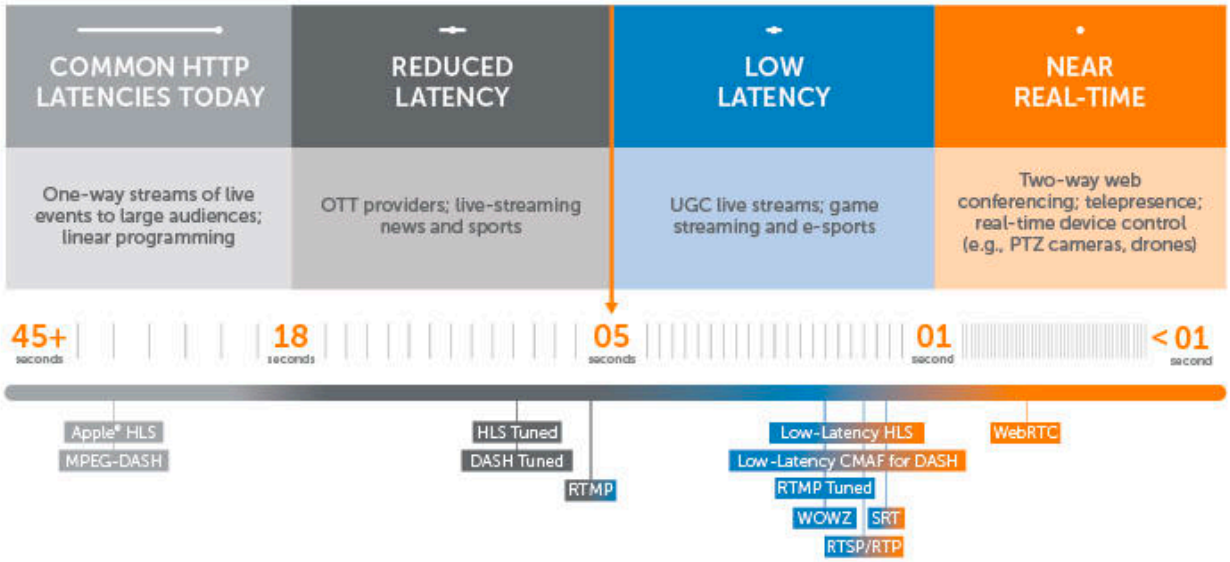

Рис. 2.2.1 – Середня затримка при використанні різних протоколів

2.2.2 Як працює WebRTC

Для підключення двох браузерів WebRTC необхідно виконати п'ять кроків для налаштування P2P-з'єднання.

- Обробка сигналу для видалення навколишнього шуму з аудіо або відео.
- Обробка кодека для стиснення та розпакування аудіо або відео.
- Маршрутизація від одного вузла до іншого через брандмауери (NAT) та ретранслятори для створення інтерактивного з'єднання (ICE)
- Дані користувача шифруються перед передачею через з'єднання.
- Управління смугою пропускання для користувача, що кожен бенкет повинен дати

Передача сигналів

P2P-з'єднання в браузері встановлюються сервером для забезпечення згоди всіх бенкетів на сеанс.

Така інформація, як сеансові ключі, повідомлення про помилки, метадані мультимедіа, кодеки, пропускну здатність та загальнодоступну IP-адресу та порти спільно використовується одноранговими вузлами для створення з'єднання.

Сервер повідомляє обом однорангові вузли, який формат мультимедіа використовувати і що кожен одноранговий вузол хоче відправити іншому.

2.2.3 Застосування WebRTC

Мета WebRTC — створення відеоконференцій з використанням однорангової технології без використання будь-якого додаткового сервера. Технологія ідеально підходить для розробки програм для відеоконференцій на основі браузера. Можна сказати, що це майже Skype усередині браузера.

WebRTC не використовує сторонні послуги, де можна було б реалізувати захист даних. Однак, технологія працює через перевірені мережеві протоколи, які забезпечують безпеку передачі даних.

2.3 SRT

SRT [4] — SRT (Secure Reliable Transport) – протокол передачі даних, розроблений на основі UDP. Він забезпечує два види гарантій – гарантію доставки даних або гарантію затримки доставки. За його допомогою забезпечується висока швидкість передачі відео. На відміну від RTMP, цей протокол підтримує всі види кодеків. За рахунок роботи зі стандартом стиснення відео H.265 (HEVC) SRT дозволяє знизити бітрейт без погіршення якості зображення та зменшити навантаження на мережу.

SRT (в основі якого UDP) допоможе збільшити швидкість передачі відео в 3-5 разів порівняно з RTMP (TCP).

Ще одна перевага: це відкритий протокол (відкритий вихідний код), з яким працюють сотні фахівців із компаній, що вступили до SRT-альянсу.

2.3.1 Переваги

• Високоякісні відео та аудіо з низькою затримкою надійно доставляються навіть через ненадійний Інтернет.

• SRT легко перетинає міжмережеві екрани (файєрволи) між джерелом та одержувачем.

• Контроль затримки, щоб пристосуватися до умов мережі, що змінюються.

• Пряма трансляція захищена за допомогою 256-розрядного шифрування AES.

2.3.2 Як працює SRT

Між джерелом SRT (кодер) та одержувачем SRT (декодер) встановлюється виділена лінія зв'язку для управління та відновлення пакетів. Одержувачем може бути сервер, CDN або інший пристрій SRT. SRT використовує свій власний метод відновлення після втрати пакетів, використовуючи UDP-пакети по мережі, які можна налаштувати для адаптації до умов мережі, що змінюються. Коли мережні умови погані, можна додати більше буферизації пакетів для покращення якості відео. У міру поліпшення умов мережі величина затримки може бути зменшена для потокової передачі практично в реальному часі.

SRT забезпечує проходження через будь-які фаєрволи між джерелом та одержувачем. Для цього протокол має три режими: рандеву та виклик/слухач.

Режим рандеву є найпростішим і зазвичай не вимагає участі IT-фахівців для налаштування проходження файрволів між джерелом та одержувачем SRT. Якщо ви не можете пройти через мережевий екран, слід використовувати режим виклик/слухач. Однак для налаштування пересилання трафіку потрібно певна участь IT-фахівців, щоб трафік, отриманий на загальнодоступну IP-адресу та порт пристрою-отримувача SRT, переадресовувався на пристрій у локальній мережі.

2.3.3 Застосування SRT

SRT ідеально підходить для відправлення кількох віддалених каналів новин непередбачуваними мережами до центрального пункту призначення для виробництва та розповсюдження, наприклад, у моделі мовлення, коли віддалені журналісти повідомляють у прямому ефірі про місцезнаходження. Він також відмінно підходить для залучення віддалених гостей із низькою затримкою для інтерв'ю або двосторонньої розмови. Щоразу, коли потрібне високоякісне відео та аудіо мережами з непередбачуваною якістю, SRT набагато перевершує якість будь-якого виклику по Zoom, потоку WebEx або WebRTC.

2.4 HLS

HLS [5] — протокол для потокової передачі медіа даних через Інтернет. HLS нарізує відео контент у форматі MP4 на короткі 10 секундні блоки, чанки. Ці короткі фрагменти доставляються HTTP, що робить протокол сумісним з більшістю пристроїв і фаєрволів.

HLS забезпечує насамперед відмінну якість онлайн трансляцій. Але потрібно враховувати, що затримка при онлайн мовленні складає 15-30 секунд. На серверній частині автор трансляції може призначити кодування потоку в кілька якостей. Потім плеєр динамічно запитує оптимальну якість, виходячи з ширини інтернет каналу в конкретний момент. Відповідно, якість фрагментів може відрізнятися.

Наприклад, мобільний телефон програє відео в HD якості, а хвилиною пізніше глядач потрапляє до зони поганого прийому. Коли плеєр виявляє зниження якості зв'язку, він запитує відео меншої якості. Таким чином знижується буферизація, зависання та інші проблеми.

Перевага HLS полягає в тому, що він призначений для адаптації до різних умов мережі. Різні версії потоку відправляються з різними дозволами та

бітрейтами. Глядачі можуть вибрати якість потоку, що вони хочуть. HLS також підтримує кілька звукових доріжок, що означає, що ваш потік може мати кілька мовних доріжок, з яких користувачі можуть вибирати потрібну. Інші переваги включають підтримку прихованих титрів, метаданих, управління цифровими правами (DRM) і навіть вбудованих рекламних оголошень (не надто далекому майбутньому).

Підтримується безпечна потокова передача HTTP, а також алгоритми хешування MD5 і SHA для автентифікації імені користувача та пароля.

2.4.1 Переваги

- Високоякісне відео (до 4K) та аудіо надійно доставляються неякісними мережами, де низька затримка не є обов'язковою вимогою.
- Легко проходить через фаейрволи.
- Адаптується до різних умов мережі та відправляє кілька відеопотоків з різними дозволами та бітрейтами.
- Підтримує кілька звукових доріжок для таких речей, як багатомовні потоки.
- Підтримує метадані та інші розширені функції.
- Економічний у розгортанні та легко масштабується з використанням традиційних мережевих серверів та технологій.
- Безпечний прямий ефір з використанням HTTP та алгоритмів аутентифікації MD5 та SHA.

2.4.2 Як працює HLS

Підхід дуже подібний до передачі файлів. Сегменти потокового мультимедіа через порт HTTP 80 (або порт 443 для HTTPS), який зазвичай вже відкритий для трафіку мережі. Таким чином, контент може легко проходити через фаєрволи практично без участі IT-фахівців.

HLS використовує контейнер транспортного потоку MPEG2-TS з напівконфігурованою тривалістю сегмента, а також з розміром списку

відтворення, що настроюється, для повторного складання прийнятих сегментів на центральному сервері. Також підтримується фрагментований MP4.

Оскільки HLS використовує технологію, засновану на TCP, метод втрати та відновлення мережних пакетів є інтенсивним. Це одна із причин збільшення затримки. Хоча є певний контроль над розміром сегмента мультимедіа, можливість зменшити затримку обмежена – особливо якщо сервер вимагає завантаження середнього сегмента певного розміру.

2.4.3 Застосування HLS

HLS, як і раніше, є стандартом для потокової передачі на мобільні пристрої та планшети. Ви також можете використовувати HLS для потокової передачі CDN, який не підтримує RTMP, коли низька затримка не є обов'язковою вимогою. Важливо відзначити, що RTMP вже вважається застарілим у дедалі більшій кількості CDN. HLS також добре підходить для безпечної потокової передачі корпоративного навчання та трансляцій через локальні мережі (LAN), коли низька затримка не є обов'язковою вимогою, а умови мережі погані (за умови, що мережа підтримує HLS).

2.5 MPEG-DASH

MPEG-DASH [6] — це відкритий стандарт адаптивного протоколу потокової передачі на основі HTTP, який відправляє відео та аудіоконтент через мережу у вигляді невеликих сегментів потокового мультимедіа на основі TCP, які повторно збираються в місці призначення. Міжнародна організація зі стандартизації (ISO) та команда MPEG та MPEG-DASH спроектували кодування та дозвіл незалежно від інших, що означає, що MPEG-DASH може передавати потокове відео (і аудіо) будь-якого формату (H.264, H.265 тощо). д.) та підтримує дозволи до 4K. В іншому MPEG-DASH функціонує майже так само, як і HLS.

Вартість розгортання MPEG-DASH низька, оскільки використовується існуюча мережна технологія на основі TCP, що є привабливим для CDN. Але

оскільки HLS використовує TCP, він працює за принципом «якість важливіша затримки», тому час затримки може бути високим

MPEG-DASH також призначений для адаптації до різних умов мережі. Різні версії потоку відправляються з різними дозволами та бітрейтами. Глядачі можуть вибрати якість потоку, що вони хочуть. Також підтримуються кілька звукових доріжок, а також розширені функції, такі як приховані титри, метадані та керування цифровими правами (DRM). Інфраструктура призначена для майбутніх розробок, наприклад, вбудованої реклами.

Підтримується безпечна потокова передача HTTP, а також алгоритми хешування MD5 і SHA для автентифікації імені користувача та пароля.

2.5.1 Переваги

- Високоякісне відео (до 4K) та аудіо надійно доставляються неякісними мережами, де низька затримка не є головною вимогою.
- Легко проходить через фаєрволи.
- Адаптується до різних умов мережі та відправляє кілька відеопотоків з різними дозволами та бітрейтами.
- Підтримує різні відео та аудіо кодеки.
- Підтримує кілька звукових доріжок, наприклад, багатомовних потоків.
- Підтримує метадані та інші розширені функції.
- Економічний у розгортанні та легко масштабується з використанням традиційних мережевих серверів та технологій.
- Безпечний прямий ефір з використанням HTTP та алгоритмів аутентифікації MD5 та SHA.

2.5.2 Як працює MPEG-DASH та його застосування

MPEG-DASH працює так само, як HLS – відправляє короткі середні сегменти HTTP (порт 80) або HTTPS (порт 443) для полегшення обходу фаєрвола. Він використовує контейнер транспортного потоку MPEG2-TS з половиною

настроюваної тривалості сегмента, а також розмір списку відтворення для повторного складання прийнятих сегментів на центральному сервері. Також підтримується фрагментований MP4.

Висока затримка MPEG-DASH обумовлена головним чином втратою мережевих пакетів та методом відновлення, що використовується у всіх мережах на основі TCP. І хоча MPEG-DASH пропонує деякий контроль над розміром сегмента мультимедіа, можливість зменшити затримку обмежена – особливо якщо сервер вимагає завантаження середнього сегмента певного розміру.

MPEG-DASH найкраще підходить для потокової передачі CDN, які не підтримують RTMP, у випадках, коли низька затримка не є обов'язковою вимогою. MPEG-DASH також добре підходить для безпечної потокової передачі корпоративного навчання та трансляцій через локальні мережі (LAN), коли низька затримка не є обов'язковою вимогою, а умови мережі погані

2.6 Висновок

Хоча WebRTC, безумовно, все ще є одним з найбільш популярних потокових протоколів, такі протоколи як SRT, HLS і MPEG-DASH, кидають йому виклик. То що вони вміють такого, чого не вміє WebRTC?

- Транслювати кілька звукових каналів для одного відеоканалу, що дозволяє транслювати багатомовний контент.
- Включати метадані та інші типи вбудованого контенту.
- Підтримувати керування цифровими правами (DRM).
- Надсилати кілька версій потоку з різними дозволами та бітрейтами, щоб глядачі могли вибрати якість, що відповідає їх умовам мережі або розміру екрана.

HLS і MPEG-DASH забезпечують набагато простішу і дешевшу масштабованість, ніж WebRTC. Так як WebRTC і зазвичай вимагає, щоб порти були відкриті вручну для проходження через фаєрволи.

Якщо затримка або погані умови мережі не є проблемою, HLS або MPEG-DASH перевищує SRT. Протоколи адаптивної потокової передачі на основі HTTP забезпечують найкращу можливу якість відео для глядачів з різними умовами мережі та простіші в налаштуванні, ніж SRT.

Якщо потрібна низька затримка і ви використовуєте потокову передачу мережами з непередбачуваною якістю, тоді SRT є кращим протоколом потокової передачі. SRT встановлює своє власне з'єднання для відновлення пакетів, яке набагато ефективніше, ніж TCP. Це дозволяє SRT забезпечувати двосторонній зв'язок між хостом та віддаленими гостями у режимі практично реального часу. Ви також можете налаштувати затримку, щоб пристосуватися до умов мережі.

У сучасних системах, якщо ви бачите відео в браузері, відеопотік і сигналізація швидше за все будуть оброблятися різними серверами. Якщо з відео все зрозуміло, то "сервер сигналізації" забезпечує дві задачі: “discovery”” та "handshake". Перше, “discover”, це вибір способу передачі: IP-адреси, проміжний сервер (якщо потрібно). «Handshake» – про домовитися між учасниками передачі відео та звуку: кодеки, роздільна здатність, частота кадрів, якість. Цікаво, що у стародавньому Flash сигналізація та передача медіа були розділені як у VoxIP чи WebRTC і забезпечувалися одним протоколом: RTMP.

Протокол сигналізації визначає мову, за допомогою якої браузер та інші учасники передачі відео домовлятимуться про discovery та handshake. Це може бути SIP для discovery у VoIP або WebRTC, і він же з offer/answer для handshake. Давним-давно у Flash використовувався RTMP/AMF. А за бажання WebRTC можна використовувати не SIP, а незвичайний JSEP.

Протокол транспорту сигналізації з того ж стека, але нижче: це те, яким чином фізично будуть передаватися пакети протоколу сигналізації. Традиційно для Flash+SIP використовували TCP або UDP, але зараз у зв'язці WebRTC+SIP все частіше можна зустріти WebSockets. Транспортний протокол WebSockets займає нішу TCP у браузерах, де не можна використовувати чисті TCP-і UDP-сокети.

Повний стек сигналізації зараз популярно описувати фразами на кшталт "SIP поверх веб-сокетів", "JSEP поверх веб-сокетів", що застаріває "SIP поверх TCP/UDP" або давнє "частина RTMP”.

Протокол стрімінгу медіа визначає, як ділити відеопотік на невеликі пакети, які надсилаються по мережі транспортним протоколом. Зазвичай протокол стримінгу ще забезпечує механізми роботи з мережевими проблемами: втрати та затримки пакетів. Jitter buffer, retransmission (RTC), redundancy (RED) і Forward Error Correction (FEC).

Вибір протоколу та мережевих портів раніше вирішувався «хардкодингом», але зараз ми використовуємо такі протоколи як ICE у WebRTC, який дозволяє домовитись про порти та транспорт у кожному конкретному підключенні. У найближчому майбутньому можливе використання протоколу QUIC (назад сумісного з UDP), який активно обговорюється IETF і має переваги над TCP та UDP за швидкістю та надійністю. Нарешті, можна згадати такі протоколи стрімінгу медіа, як MPEG-DASH та HLS, які використовують HTTP як транспорт і отримають переваги від впровадження HTTP/2.0.

Деякі протоколи захищають дані під час передачі по мережі: або сам медіапотік, або пакети транспортного рівня. Процес включає ту саму передачу ключів шифрування, для чого використовуються окремі протоколи: SDES VoIP і DTLS WebRTC. Останній має перевагу, оскільки крім даних захищає і саму передачу ключів шифрування.

Деякі розробники, наприклад автори цієї статті [7] , поміщають суто транспортні протоколи WebSocket і QUIC на той же рівень, що і WebRTC, Flash, або HLS. Для мене таке угруповання виглядає дивно, адже три останні протоколи — це історія про стрімінг медіа. Кодування та розбиття на пакети відбувається до використання WebSocket або QUIC. Еталонна реалізація WebRTC (libwebrtc/chrome) від Google та ORTC від Microsoft використовують QUIC як транспортний протокол.

		Flash					WebRTC	SRT	HLS	MPEG-DASH	CMAF
Signalling path	Signalling protocol	RTMP	RTMPS	RTMPE	RTMPT	RTMFP	open		URL	URL	
	Discovery	Hardcoded IP:port					JSEP		URL	URL	
	Handshake	3 bits after TCP session est.					SDP O/A		HTTP	HTTP	
	Signalling transport protocol	TCP	TLS/SSL	TCP	HTTP	UDP	open (e.g WS)		HTTP	HTTP	
Media Path	Video Codec	FLV1 / H.264					H.264 / VP8		H.264 / H.265		
	bitrate adaptive encoder	No					yes	No	transcoder / file-based		
	Media Streaming protocol	RTMP					RTP	SRT	MPEG-2 TS fMP4	fMP4	fMP4
	Network feedback protocol	No					RTCP	yes	client / player-based		
	Reliability protocols(s)	No					RTX / RED / FEC	yes	No		
	Media Streaming Security	No	from transp.	adobe's	No	No	DTLS-SRTP	AES	AES/SDES, DRM, cookies		
	Media Streaming Transport	TCP	TLS/SSL	TCP	HTTP (80)	UDP	TCP, TLS, UDP, DTLS, QUIC, HTTP	UDP	HTTP		
	Allocation	hardcoded					ICE	1-hop max	URL		

Рис. 2.6.1 – Порівняння різних протоколів за їх можливостями [8]

Не менш дивовижною є відсутність згадки HTTP/2.0 як оптимізації для протоколів, заснованих на HTTP, таких як HLS і MPEG-DASH. А згаданий CMAF – не більше ніж формат файлів для HLS та MPEG-DASH, але ніяк не їх заміна.

Нарешті, SRT є лише транспортним протоколом. Він, звичайно, додає ряд фішок у порівнянні з заснованими на файлах HLS і MPEG-DASH, але всі ці фішки вже є на іншому рівні стека і реалізуються в RTMP або WebRTC. Ще SRT поділяє кодування медіапотоку та статистики, що не дозволяє кодеку тримати цю інформацію максимально близько одна від одної. Таке рішення може негативно позначитися на можливості адаптувати відео, що пересилається під змінну пропускну здатність мережі.

Засновані на файлах протоколи, такі як HLS, кодують кілька потоків і вибирають необхідні адаптації до ширини каналу. WebRTC дозволяє адаптувати

кодування кожного кадру в реальному часі: це набагато швидше, ніж вибір іншого потоку HLS, який вимагає рахувати до 10 секунд вже відправлених даних.

WebRTC гарний вибір тому, що дозволяє встановити зв'язок між користувачами, використовуючи тільки браузер. Для деяких розробників, коли вони дізнаються про WebRTC, стає відкритим: адже можна створити відеочат без використання стороннього сервера — потрібен тільки браузер.

WebRTC не вимагає встановлення додаткових плагінів. Потрібно просто написати код на HTML або JavaScript, і відеопотоки в браузері будуть працювати плавно. WebRTC призначений для високопродуктивної та якісної передачі відео, аудіо та довільних даних.

Додатки WebRTC потребують служби, за допомогою якої вони можуть обмінюватися мережевими та мультимедійними метаданними - цей процес називається сигналізацією. Однак після передачі сигналу відео/аудіо/дані передаються безпосередньо між клієнтами, що дозволяє уникнути витрат на продуктивність при передачі через проміжний сервер.

WebSocket, з іншого боку, призначений для двонаправленого зв’язку між клієнтом і сервером. Через WebSocket можна передавати потокове аудіо та відео, але технологія та API за своєю суттю не призначені для ефективної та надійної потокової передачі, як це робить WebRTC.

Як було зазначено в інших відповідях, WebSocket можна використовувати для передачі сигналів. Проте одним із ключових факторов було те, що наш браузерний застосунок буде на базі service worker з обмеженим часом існування. WebRTC має механізм часткового перепідключення, тому наш зв’язок буде більш стабільним при конфігурації за замовчуванням, ніж при використанні WebSockets.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Конфігурація manifest json файлу

Почати слід з ініціалізації **manifest.ts** файлу, що буде містити інформацію про розширення, його назву, версії застосунку, версії manifest API, також має містити інформацію про шляхи до виконуваних файлах та дозволах, що будуть надані браузером для данного розширення.

```
export const MANIFEST = {
    "manifest_version": 3,
    "version": "1.0",
    "name": "WorkCheck",
    "permissions": [
        "storage",
        "tabs",
        "management"
    ],
    "action": {
        "default_title": "Click me",
        "default_popup": "popup.html"
    },
    "chrome_url_overrides": {
        "newtab": "newtab.html"
    },
    "content_scripts": [
        {
            "matches": ["http://*/*", "https://*/*"],
            "js": ["contentScript.js"]
        }
    ],
    "background": {
        "service_worker": "background.js",
        "type": "module"
      }
}
```

Рис. 3.1.1 – Конфігурація manifest.ts

3.2 Білд проекту за допомогою Webpack

Так як ми оперуємо нашими файлами як окремами компонентами, а не модулями однієї аплікації, то і білдити їх ми будемо так само окремо. [9]

Підключаємо відповідні завантажувачів різних модулів (css , ts, tsx, etc. Loaders).

```
entry: {
    popup: path.resolve('./src/popup/index.tsx'),
    background: path.resolve('./src/background/background.ts'),
    contentScript: path.resolve('./src/content/contentScript.tsx'),
    newTab: path.resolve('./src/newTab/index.tsx')
},
```

Рис. 3.2.1 – Білд ts{x} файлів

Оскільки нативне браузерне середовище не підтримує взаємодію з буферами, то нам треба реалізувати підтримку буферів і потоків даних.

```
resolve: {
    extensions: ['.ts', '.tsx', '.js', '.json'],
    fallback: {
        "stream": require.resolve("stream-browserify"),
        "buffer": require.resolve("buffer")
    }
},
```

Рис. 3.2.2 – Підтримка буферів і потоків даних

Залишилось все це поєднати у вихідний файл і підключити конфігурацію

```
output: {
    filename: '[name].js',
},
plugins: [
    ...getHtmlPlugins([
        'popup',
        'newTab',
    ]),
    new webpack.ProvidePlugin({
        Buffer: ['buffer', 'Buffer'],
    }),
    new webpack.ProvidePlugin({
        process: 'process/browser',
    }),
    new CopyPlugin({
      patterns: [
        { from: path.resolve('src/assets'), to: path.resolve('dist') },
      ],
    }),
  ],
```

Рис. 3.2.3 – Білд результуючого файлу з відповідними плагінами

3.3 Компоненти браузерного розширення

Далі підключаємо логіку, яка буде реалізована у фонових процесах за допомогою service workers. Першим кроком буде перевірка чи взагалі наше браузерне розширення увімкнене. Потім, відповідно до вибранного нами викладача, ми будемо перевіряти чи url нашого пошуку збігається з одним із значень із списку доступних сервісів для використання. Якщо ні — таба буде закрита ще до її рендеру.

```
chrome.tabs.onCreated.addListener(
    (tab) => {
        chrome.storage.sync.get(["isEnabled"], (res) => {
            if (res.isEnabled) {
                chrome.tabs.onUpdated.addListener(async (tabId, changeInfo, tab) => {
                    const {User} = await chrome.storage.sync.get("User")
                    const response = await fetch(
                        'http://localhost:3001/professor/getPolicies?' +  new URLSearchParams({
                            name: User.profId,
                        }),
                        {
                            method: 'GET',

                        },
                    );
                    const {policies} = await response.json();
                    if(policies.includes(matchParse(tab.url)) || tab.url === 'chrome://newtab/'
                    } else {
                        chrome.tabs.remove(tab.id)
                    }
                })
            }
        })
    }
)
```

Рис. 3.3.1 – background.ts

Як вже було вище сказано, то для UI ми будемо використовувати бібліотеку React, а саме для відображення компоненту вспливаючого вікна, якщо ми говоримо в контексті саме браузерного розширення. Спочатку перевіряємо чи взагалі наше розширення увімкнене, а потім якщо в нас немає даних про юзера, то ми будемо показувати форму авторизації, у іншому випадку — ми підтягуємо дані про юзера і відображаємо саме їх без додаткового етапу авторизації

```
useEffect(() => {
    chrome.storage.sync.get(["isEnabled"], (res) => {
        if (res.isEnabled) {
            toggle(true)
        }
    })
    chrome.storage.sync.get(["User"], ({ User }) => {
        setUser({ name: User.name, profId: User.profId })
    })

}, [])
```

Рис. 3.3.2 – popup.tsx

Слід також сказати, що для встановлення стилів було використано UI-бібліотеку Tailwind з наступною конфігурацією:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Рис. 3.3.3 – popup.tsx

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Рис. 3.3.4 – tailwind.config.ts

Останнім кроком буде реалізація механізму передачі аудіо і відео потоків під час переходу на табу з дозволенним URL. Спочатку перевіряємо чи наш застосунок увімкнений і користувач, під яким ми авторизовувались, передається правильно. Потім встановлюємо зв’язок через Peer JS[10] сервер, реалізацію якого буде розглянуто в [2.6].

```
peer = new Peer(res.User.name, {
    host: "localhost",
    port: 9000,
    path: "/",
    secure: false
})
```

Рис. 3.3.5 – встановлення Peer-to-peer зв’язку

Якщо все валідно, то конектимось до peer з іншої сторони (нашого бекенду) і вже туди після успішного під'єднання ми будемо передавати наші потокові дані шляхом створення "дзвінка".

```
const conn = peer.connect(res.User.profId)
conn.on('open', () => {
    conn.send("Hi there:3")
})
const shareScreen = async () => {
    if (navigator.mediaDevices.getDisplayMedia) {
        try {
            stream = await navigator.mediaDevices.getDisplayMedia({
                audio: false,
                video: true
            })
            const call = peer.call(res.User.profId, stream, {
                metadata: {
                    userName: res.User.name,
                },
            });
            call.on('stream', function(remoteStream) {
                videoRef.current.srcObject = remoteStream
            });
            socket.on('call', (call) => {
                call.answer(stream)
            })
        } catch (e) {
            console.log(e)
        }
    }
}
shareScreen()
```

Рис. 3.3.6 – функція захвату екрана

Ми робимо запит до безпосередньо користувача (ймовірно, студента) цього браузерного розширення на показ його екрану під час роботи. Контроль вимкнення поза роботою відбувається за допомогою кнопок "Enable / Disable"

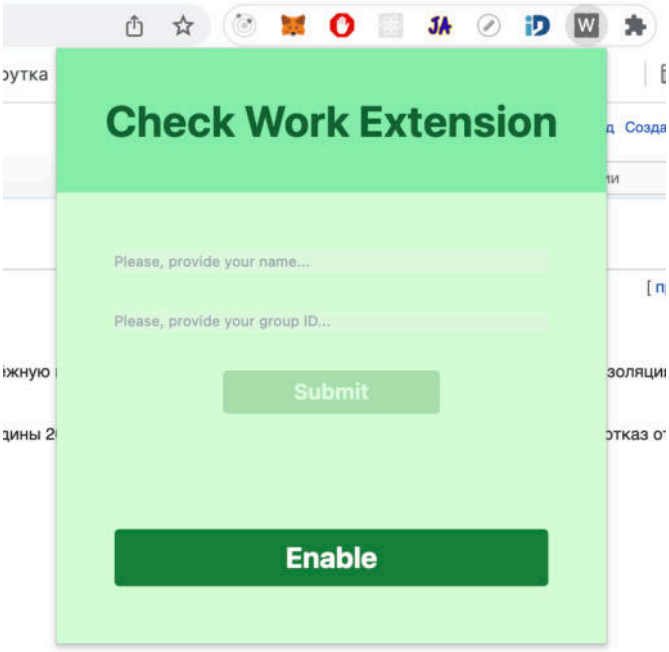

Рис. 3.3.7 – “disabled” стан

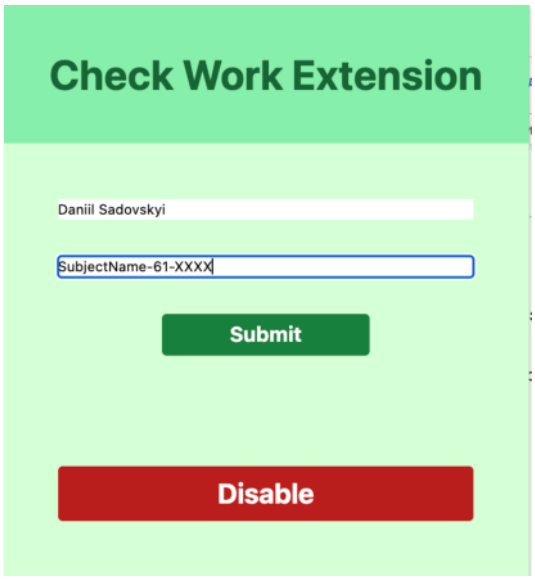

Рис. 3.3.8 – “enabled" стан з заповненими полями

Запит складається з вибору вікна, що буде транслюватись та взагалі питання чи він бажає транслювати той чи інший екран / табу. Якщо все добре і юзер погоджується транслювати свох дані, то далі передається потік відповідному peer користувачу, з яким напередодні було встановлено зв’ язок. І на запити “дзвінка” з його сторони, ми так само будемо транслювати свої дані і навіть зможемо зродити

в дуплексному режимі, як це зроблено під час звичайник відео дзвінків, коли користувачі можут бачити приймати потік мультимедійних даних один одного одначасно.

Також слід зауважити реалізацію допоміжної функцію, що полегшить процес роботи з посиланнями (хоча це також зроблено і на стороні серверу, проте, добре мати додатковий захист від неочікуваних результатів також на стороні клієнта) . Ця функція дозволяє скоротити посилання лише до доменного імені без додаткових параметрів чи інших запитів.

```
export const matchParse = (url:string):string => {
    const domain = new URL(url).origin;
    return `${domain}/**/*`
}
```

Рис. 3.3.9 – matchParse.ts

3.4 Peer-to-Peer сервер

Peer-to-pear — різновид архітектури сиестеми, в базі якої лежить мережа рівноправних вузлів.

Комп'ютерні мережі типу peer-to-peer (або P2P) засновані на принципі рівноправності учасників і характеризуються тим, що їх елементи можуть зв'язуватися між собою, на відміну від традиційної архітектури, коли лише окрема категорія учасників, яка називається серверами, може надавати певні сервіси іншим.

Однорангові з’єднання — це частина специфікацій WebRTC, яка стосується з’єднання двох програм на різних комп’ютерах для обміну даними за допомогою однорангового протоколу. Зв’язок між вузлами може бути відео, аудіо або довільними двійковими даними (для клієнтів, які підтримують API RTCDataChannel). Щоб дізнатися, як два однорангові вузли можуть підключитися, обидва клієнти повинні надати конфігурацію сервера. Це або STUN, або TURN-сервер [11] , і їх роль полягає в наданні кандидатів ICE кожному клієнту, який потім передається віддаленому одноранговому серверу. Ця передача кандидатів ICE зазвичай називається сигналізацією.

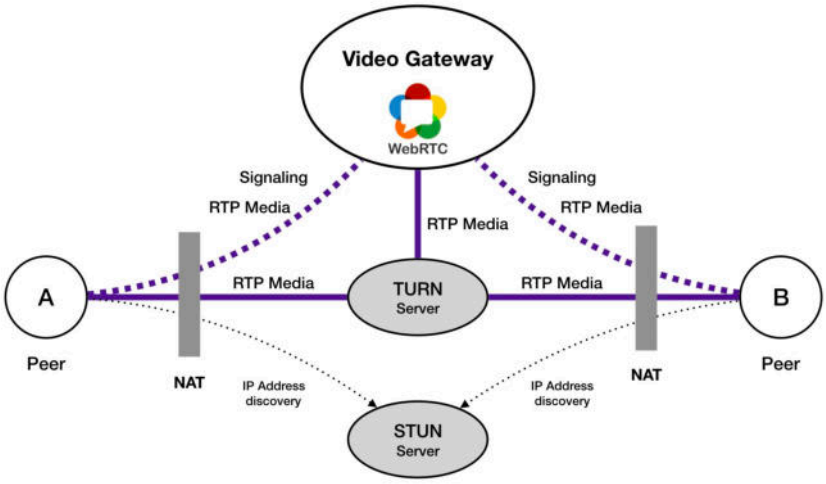

Рис. 3.4.1 – схема TURN + STUN

Для створення такого серверу і подальшій передачі даних ми будемо використовувати готове рішення PeerJS.

PeerJS обгортає реалізацію WebRTC браузера, щоб забезпечити повний, настроюваний і простий у використанні API одноранговогo з'єднання. Не маючи нічого, крім ідентифікатора, одноранговий вузол може створити з'єднання даних P2P або медіа-потоку з віддаленим одноранговим вузлом.

```
(base) MacBook-Pro-Daniil:~ danielsadovskiy$ peerjs --port 9000 --key peerjs --path /
```

Рис. 3.4.2 – підняття локального peer серверу

Після того як сервер був піднятий і ми не отримали помилок, тепер ми можемо обмінюватись потоками даних між підключенними користувачами.

3.5 Браузерне відображенння

Було створено інтерфейс адміністратора (викладача у нашому випадку) за допомогою бібліотеки React.

Першим кроком була ініціалізація з додаванням компоненту роутера для відображення відповідних сторінок.

```
import React from 'react';  6.9k (gzipped: 2.7k)
import ReactDOM from 'react-dom/client';  513 (gzipped: 319)
import './index.css';
import {
  RouterProvider,
} from "react-router-dom";  11k (gzipped: 4.4k)
import { router } from './routes';
import { ToastContainer } from 'react-toastify';  17.4k (gzipped: 6.6k)
import 'react-toastify/dist/ReactToastify.css';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
      <RouterProvider router={router} />
      <ToastContainer/>
  </React.StrictMode>
);
```

Рис. 3.5.1 – ініціалізація браузерного відображення

Далі було додано компонент ProtectedRoute, що дозволяв входити лише авторизуваному користувачу. Оскільки перший пароль для входу встановлювався на рівні системи, тому для перегляду налаштувань та безпосередньо головної сторінки, необхідно було спочатку змінити початковий пароль на більш захищений, що і робив при першій авторизації викладач.

```tsx
export const ProtectedRoute = ({
    permissions,
    redirectPath = '/dashboard',
    children,
  }: IProtectedRoute) => {
    const user =  JSON.parse(localStorage.getItem('user') as string) || undefined;

    let isAllowed = false;

    if(permissions.includes('passwordChanged')) {
        if(user && user.passwordIsChanged) {
            isAllowed = true;
        }
    }

    if(permissions.includes('user') && user) {
        isAllowed = true
    }


    if (!isAllowed) {
      return <Navigate to={redirectPath} replace />;
    }

    return children ? <>{children}</> : <Outlet />;
  };
```

Рис. 3.5.2 – створення защиненного роуту

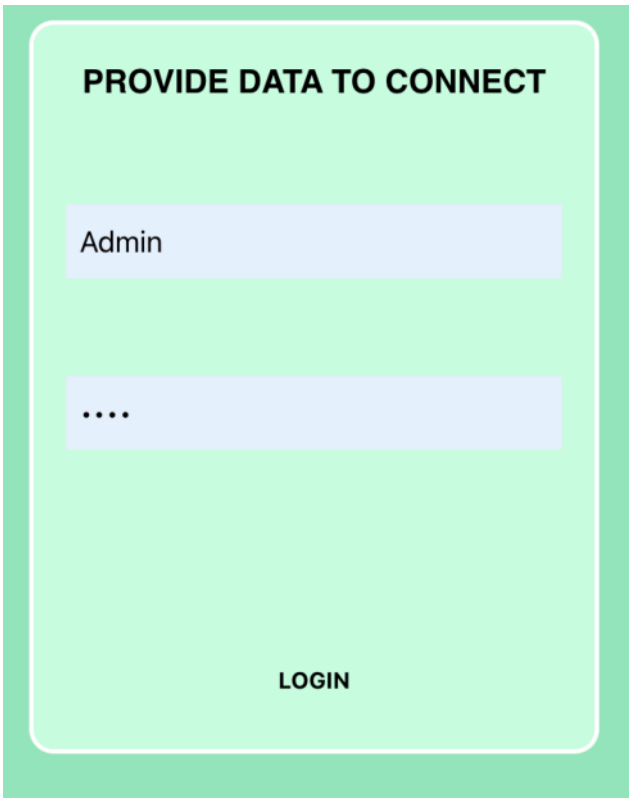

Рис. 3.5.3 – відображення форми для авторизації

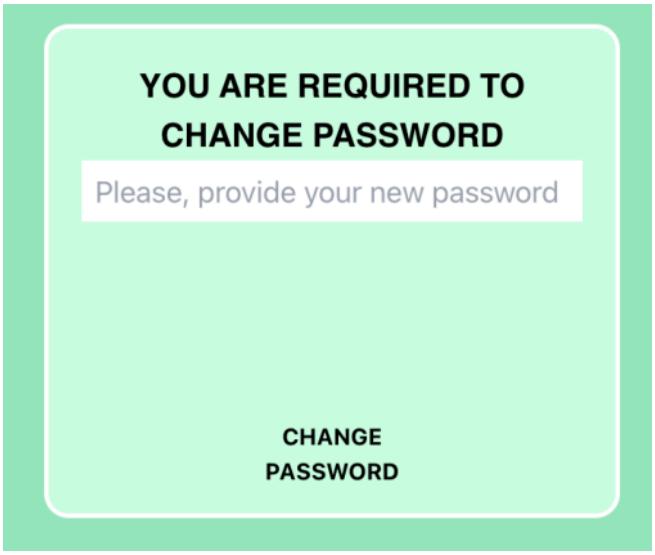

Рис. 3.5.4 – відображення форми для обов'язкової зміни паролю

Після того, як тимчасовий пароль було змінено, викладач отримує можливість перейти до головної сторінки, де можна бачити трансляції активних користувачів та слідкувати за їх статусом після спроби обійти контроль, також є можливість перейти до сторінки налаштувань, де можна повторно змінити пароль за власним бажанням чи додати домени перегляд яких, на думку викладача, можуть створити дисперсію у аналізі справедливих знань студента і впливати на результуючу оцінку.

Рис. 3.5.5 – відображення сторінки налаштувань викладача

Тут слід звернути увагу на те, що коли буде додано новий заборонений домен, він буде хендлитись за допомогою наступної функції на стороні браузерного за стосунку. І якщо користувач (студент) пробує перейти на заборонений домен, то браузерно автоматично буде закривати цю табу і повертатись до найближчої таби, що знаходиться за дозволеним доменом.

```
const { policies } = await response.json();
if (policies.includes(matchParse(tab.url)) || tab.url === 'chrome://newtab/') {
} else {
    chrome.tabs.remove(tab.id)
}
```

Рис. 3.5.6 – відображення сторінки налаштувань викладача

На головній сторінці викладач матиме змогу спостерігати за підключеними студентами та водночас спостерігати за станом їхньої роботи.

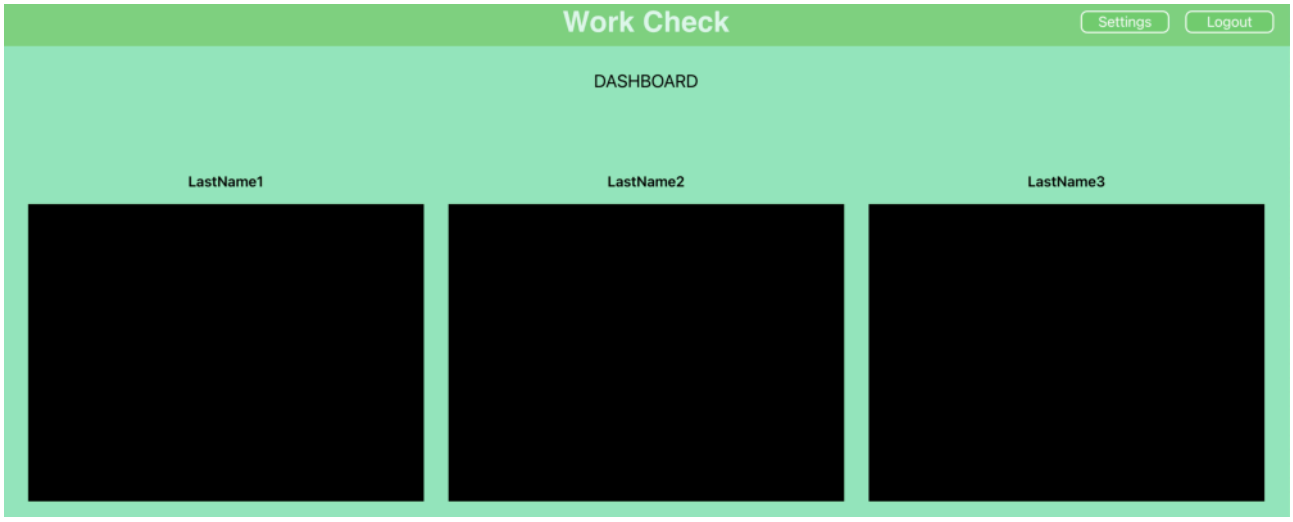

Рис. 3.5.7 – відображення головної сторінки викладача

Насправді, ці чорні квадрати — замокані тестові дані, що виконують дві функції. Перша — перевірка підтримки MediaDevice функціоналу в браузері, що необхідний для коректної роботи передачі даних. Хоча оскільки ми використовуємо WebRTC, то ймовірність, що браузер не підтримує цю технологію дуже низька. Друга функція — приклад відображення блоків, де відбуватиметься трансляція роботи студента.

В коді ми створюємо функцію-приймач, що будет отримувати потоки даних від трансляції студентів. Для цього ініціалізуємо Peer інстанс, до якого і будуть під’єднюватись всі решта.

```
const peer = new Peer(myPeer, {
    host: "localhost",
    port: 9000,
    path: "/",
    secure: false
})
```

Рис. 3.5.8 – ініціалізація peer-абонента викладача

В цьому випадку myPeer — це об’єкт з даними викладача і ідентифікатором, за яким можна буде приєднатись саме до нього.

Наступна функція-слухач буде очікувати спроби під’єднатись від студента. Оскільки протокол WebRTC працює в обох напрямках, то ми відповідаємо створюючи фейковий потік даних (той самий, який ми використовуєм для перевірки підтримки необхідно функціоналу браузером) та додаємо цей конекшн до списку, щоб відобразити.

```
peer.on('call', async function (call) {
    console.log('call', call);
    if (navigator.mediaDevices.getDisplayMedia) {
        try {
            call.answer(createMediaStreamFake());
            call.on('stream', function (remoteStream) {
                if (!Object.keys(peers).includes(call.metadata.userName)) {
                    setPeers(prevState => ({
                        ...prevState,
                        [call.metadata.userName]: remoteStream
                    }))
                }
            });
        } catch (e) {
            console.log(e)
        }
    }
});
```

Рис. 3.5.9 – функція відповіді викладача на запит з’єднання від студента

Тобто, ми створюємо фейковий потік аудіо та відео формату для того, щоб відправити його студенту, проте оскільки мета програми не зацікавлена в спостеріганні за діями викладача, то зі сторони студента обробити потік, чи навіть зрозуміти, що він не несе в собі ніяких реальних даних неможливо.

У випадку, коли зв'язок було відключено повністю, а не перервано тимчасово (наприкад, у випадку роз'єднання інтернет зв'язку), ми вилучаємо вже закінчену трансляцію студента зі списку спостереження.

```
peer.on('close', function(){
    console.log('close event')
    const activePeers = Object.entries(peers).filter(([key,value]) => value.active)
    setPeers(Object.fromEntries(activePeers))
})
```

Рис. 3.5.10 – вилучення від'єднанного підключення

Якщо ж відбулось тимчасове відключенння, то при появі стабільного зв'язку лише втрачені пакети знову будуть перезапрошені, а не вся послідовність, та трансляція автоматично продовжиться за допомогою вбудованного механізму WebRTC. Що ж щодо спроби перервати зв'язок та спробувати обійти контроль, то в такому разі відправлення попередження відбувається за допомогою WebSocket [12], які ми розглянемо далі в серверній частині. Хоча і можна було би підтягувати історичні дані переходу між доменами під час відключення браузерного розширення чи втрати зв'язку проте це не буде ефективним у випадку використання іншої програми, оскільки контроль за цим можливо вести лише на рівні доступу до операційної системи, що не є можливим в контексті даної реалізації.

3.6 Серверна частина

Слід почати з того, що я ініціалізував сервер, додам мідлвари для обробки тіла запиту і коректної роботи з ним, додав можливість хендлити запити за відповідними роутами, мідлвар для обробки помилок (щоб не було неочікуванного падіння серверу і неможливості подальшої роботи з ним). Далі йде

функція підключення до БД і заповнення таблиці даними викладачів з початковими паролями, які, як я вже зазначив, необхідно змінити при першому вході.

```
app.use(cors({
    origin: "http://localhost:3000",
    allowedHeaders: ["Content-Type", "Authorization", "Access-Control-Allow-Methods", "Access-Control-Request-Headers"
    credentials: true,
    preflightContinue: true
}))

app.use(cookieParser())
app.use(loggerMiddleware)
app.use('/user', userRouter)
app.use('/professor',professorRouter)

app.use(errorMiddleware);


const runApp = () => {
    server.listen(config.server.app_port, () => {
    console.log(`Listen on the port ${config.server.app_port}...`);
    });
}

try {
    mongoServices.mongoConnect(() => {
        seedDB()
        runApp()
    })
} catch (e) {
    console.error(e)
}
```

Рис. 3.6.1 – Ініціалізація сервера

Можна спостерігати наступну структуру файлів/папок.

В папці config зберігаються константи, так як порти для бд, для вебсокет, peer серверу, порт для ініціалізації сервера, дані для під’єднання до БД.

Controllers зберігають функції для обробки запитів за відповідними роутами.

Middlewares містять проксі функції.

Models містять моделі, що зберігаються в БД та функцію заповнення при початковій ініціалізації таблиць.

В Routes можна бачити список роутів з якими ми будемо працювати та відповідні хендлери.

Services містять функції, що обробляють дані поза локальним сервером. В нашому випадку це робота з БД.

Utils для допоміжних функцій.

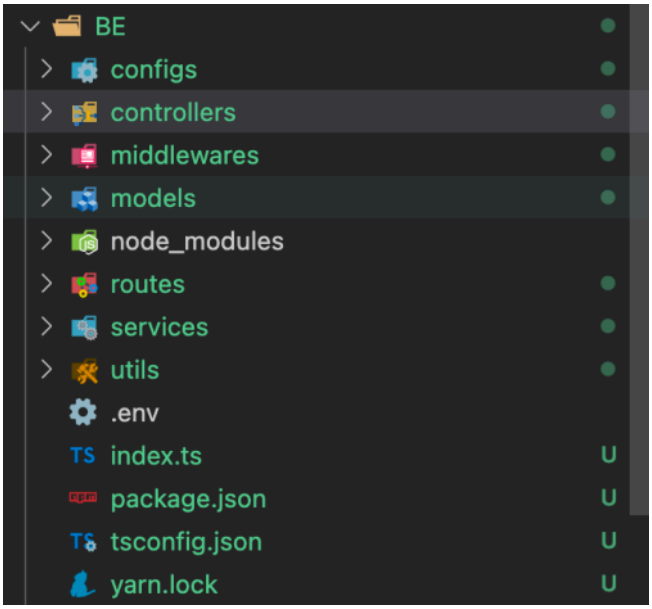

Рис. 3.6.2 – Структуру файлів / папок

```
const getPolicies = async (req: express.Request, res:express.Response) => {
    console.log(req.query)
    const name = req.query.name as string
    try {
        const {policies} = await professorService.getPolicies({name})
        res.status(200).send({policies});
    } catch(e){
        res.status(e.status || 500).send({message: e.message})
    }
};

const userChangedFocusHandler = (req: express.Request, res: express.Response) => {
    const { name } = req.body;
    console.log(req.body)
    try{
        io.emit("focus", {name})
    } catch(e) {
        console.log(e)
    }
    res.status(200)
}

const changeOldPasswordHandler = async ( req: express.Request, res: express.Response) => {
    try{
        const { id, oldPassword, newPassword } = req.body;
        const response = await professorService.changeOldPassword({id, oldPassword, newPassword})
        res.status(201).send(response)
    } catch(e) {
        res.status(e.status || 500).send({message: e.message})
    }
}
```

Рис. 3.6.3– Хендлери для роботи з запитами

На стороні серва ми обробяємо такі запити, які викликаються безпосередньо зі сторони UI для викладача та ті, що приходять через WebSocket зі сторони браузерного розширення. Деякі з хендлерів можна побачити на наступному рис. 3.6.2, 3.6.3.

getPolicies для отримання поточного списку забороненних ресурсів від данного викладача.

Хендлер запиту на зміну паролю та хендлер для контролю роботи студента згідно його браузерного вікна з цим браузерним застосунком.

В Services можемо бачити хендлери для обробки запитів на рівні БД, наприклад, зберіганння паролю чи модифікація дозволенних ресурсів

```
const addNewPolicy = async ({id, policy}: IAddPolicy) => {
    const user = await Professor.findById(id)
    if (!user) {
        throw new CustomError(404, "Professor with provided name not found.");
    }
    if(user.policies.includes(policy)){
        throw new CustomError(404, "Provided policy already exists");
    }

    try {
        const response = await Professor.findOneAndUpdate({_id: id} , {policies: [...user.policies, policy]})
        return {message: "Policies succesfully updated", policy}
    } catch (err) {
        return err
    }
}
```

Рис. 3.6.4 – Хендлери для роботи з запитами на рівні БД

```
chrome.tabs.onCreated.addListener(
    (tab) => {
        chrome.storage.sync.get(["isEnabled"], (res) => {
            if (res.isEnabled) {
                chrome.tabs.onUpdated.addListener(async (tabId, changeInfo, tab) => {
                    const { User } = await chrome.storage.sync.get("User")
                    const response = await fetch(
                        'http://localhost:3001/professor/getPolicies?' + new URLSearchParams({
                            name: User.profId,
                        }),
                        {
                            method: 'GET',

                        },
                    );
                    const { policies } = await response.json();
                    if (policies.includes(matchParse(tab.url)) || tab.url === 'chrome://newtab/') {
                    } else {
                        chrome.tabs.remove(tab.id)
                    }
                })
            }
        })
    }
)
```

Рис. 3.6.5 – Встановлення зв’язку зі сторони клієнта БЗ

Більша частина вже готова для роботи, залишилось додати функціонал для встановення зв’язку і транслювання екрану на стороні користувача браузерного застосунку (БЗ) .

Тобто, відбувається наступне: при переході на нову вкладку і при подальшому переході на певний домен, background script спочатку перевіряє чи знаходиться цей домен у списку заборонених викладачем, якщо так — вкладка закривається, якщо ні — то користувачу буде запропоновано транслювати екран. Після підтвердження буде створений медіа потік, що включає в собі аудіо та відео стрім. Після цього відбувається спроба під’єднатись до викладача за заданим параметром (ім’я чи назва предмету, який ініціалізує викладач) . Якщо такий було знайдено, відбувається зв’язок, викладач автоматично приймає і зі своєї сторони передає фейковий пустий потік аудіо та відео. На стороні користувача БЗ відбувається запис екрану в такий самий потік, проте він вже не пустий. І передається через peer сервер. Коли ми вимикаємо трансляцію, то можна спостерігати як блок відображення на стороні викладача зникає зі списку.

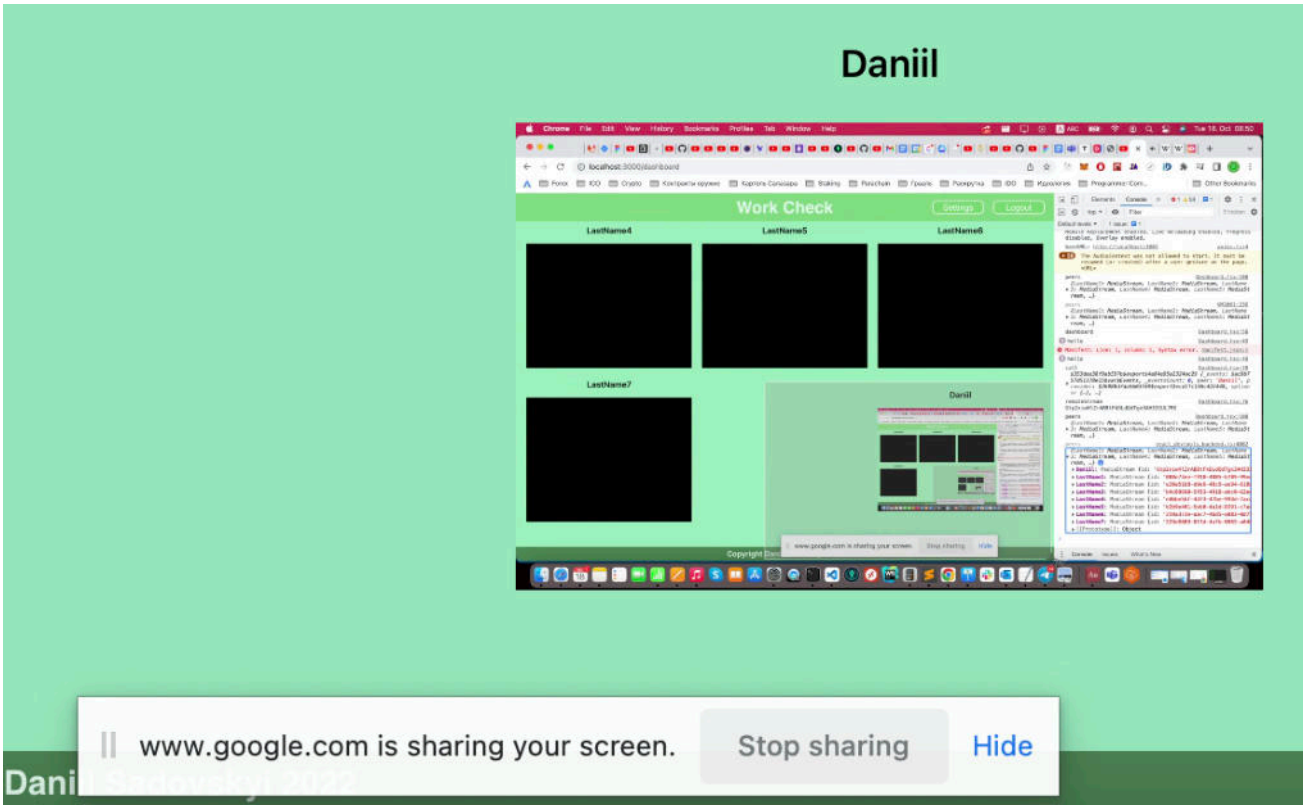

Рис. 3.6.6 – Відображення трасляції студента на головній сторінці викладача

Якщо студент під час роботи переходить до інших інстансів програм , крім вікна браузера, де був налаштований цей застосунок, то викладач може бачити його логін підсвіченим червоним кольором на 10 секунд (цей параметр дуже легко кастомізувати) . Хоча це і є недоліком, оскільки під час роботи бувають випадки коли можна використовувати інші програми, наприклад, калькулятор для певних обчислень, проте, так як студент транслює свій екран, то можна спостерігати, що саме він робить і таким чином контролювати процес роботи.

Рис. 3.6.6 – Функція, що перевіряє фокус на вікні браузера і тригерить застереження на стороні викладача

Можна спостерігати, що робота відбувається за очікуваним механізмом. При втраті зв’язку і його відновлені відбувається перепідключення. Трансляція відео йде без затримок. При спробі перейти до іншої програми, викладач протягом 10 секунд может бачити застереження про певного студента. БД зберігає значення

коректно. При спробі переходу на заборонений домен відповідна таба закривається одразу ж.

Бачимо, що всі запити до всіх вимог було виконано і роботу можна вважати успішною.

ВИСНОВОК

У результаті виконання роботи вдалося реалізувати браузерний застосунок для контролю роботи студента за допомогою протоколу передачі потокових даних WebRTC. Було створено архітектуру “клієнт-сервер” на базі React та NodeJS. В контексті сховища даних було використано NoSQL MongoDB, хоча для більш складної кастомізації, нормалізації бази даних, зменьшення надлишковості, eventual consistency для збору інформації про дії користувачів, за якими ведеться спостереження та усунення неузгодженності операцій можна було використувати SQL базу даних з підтримкою ACID вимог. Було проаналізовано різні види протоколів передачі даних, проте, для конкретної задачі було вибрано WebRTC, оскільки конфігурація досить не складна, цей протокол реалізує вбудовані механізми для підвищення якості передачі даних та немає необхідності для створення додаткового серверу. Оскільки я працював в системі одного ізольованого застосунку без необхідності контактувати з іншими програмами, віддаленого доступу до інших ПК, анонімного підключення та гарантії отримання даних іншою стороною, то недоліками використання WebRTC можна знехтувати. На практиці було реалізовано та перевірено методи протидії контролю, такі як: доступ до заборонених адміністратором каналу ресурсів та можливість обійти застосунок використовуючи інші програми.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Chrome Manifest V3. [Електронний ресурс]: Chrome – Режим доступу: https://developer.chrome.com/docs/extensions/mv3/intro/
2. Chrome Manifest V3. [Електронний ресурс]: Chrome – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket?retiredLocale=uk
3. WebRTC for App and Server. [Електронний ресурс]: Tyler Liu – Режим доступу: https://medium.com/ringcentral-developers/webrtc-app-for-both-browser-and-server-9600fd08aacf
4. SRT protocol for streaming. [Електронний ресурс]: Epiphany – Режим доступу: https://www.epiphan.com/blog/srt-protocol/
5. What Is HLS Streaming and When Should You Use It [Електронний ресурс]: Max Wilbert – 2022 – Режим доступу: https://www.dacast.com/blog/hls-streaming-protocol/
6. What is MPEG-DASH Video Streaming Protocol. [Електронний ресурс]: Krishna Rao Vijayanagar – 2021 – Режим доступу: https://ottverse.com/mpeg-dash-video-streaming-the-complete-guide/
7. What Is Low Latency and Who Needs It. [Електронний ресурс]: Wowza Media Systems — 2022 – Режим доступу: https://www.wowza.com/blog/what-is-low-latency-and-who-needs-it
8. Comparing the Best Protocols for Live Streaming. [Електронний ресурс]: Emily Krings — 2022 – Режим доступу: https://www.dacast.com/blog/rtmp-vs-hls-vs-webrtc/
9. How to Build a Chrome Extension With React, TypeScript, and Webpack. [Електронний ресурс]: Jakub Kozak – 2020 – Режим доступу: https://medium.com/swlh/how-to-build-a-chrome-extension-with-react-typescript-and-webpack-92e806ce2e16
10. Simplified Peer to Peer Communication with PeerJS. [Електронний ресурс]: Dulanka Karunasena – 2021 – Режим доступу: https://blog.bitsrc.io/simplified-peer-to-peer-communication-with-peerjs-e37244267723

11. STUN vs TURN servers in WebRTC. [Електронний ресурс]: Robert Strobl — 2021 – Режим доступу: https://www.digitalsamba.com/blog/stun-vs-turn
12. WebRTC signaling with WebSocket and Node.js. [Електронний ресурс]: Alexander Nnakwue — 2022 – Режим доступу: https://blog.logrocket.com/webrtc-signaling-websocket-node-js/