Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ КОМУНІКАЦІЇ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ З ВИКОРИСТАННЯМ ПРОТОКОЛУ WEBSOCKET
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-1
Григоров Г.В
(прізвище, ініціали)

Спеціальності 122 Комп’ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник ст. викл.Путятіна О.Є.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ____________ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри ______________
(підпис)
«____»_______________ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Григорову Георгію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення вебзастосунку для комунікації в режимі реального часу з використанням протоколу WebSocket

затверджена наказом університету від 16 травня 2022 року № 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 30 травня 2022 р.

3. Вихідні дані до роботи матеріали про розробку вебзастосунків, інтернет-ресурси, протокол WebSocket, бібліотека з відкритим кодом Socket.IO

4. Перелік питань, що потрібно опрацювати в роботі
1. Зв'язок у реальному часі.
2. Протокол WebSocket.
3. Проектування системи комунікації в реальному часі
4. Програмна реалізація клієнтської частини
5. Програмна реалізація серверної частини

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) зв’язок за допомогою WebSocket, мікросервісної архітектури та зв’язку між двома сервісами, тестові зображення

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз технічних засобів	26.04.22-30.04.22	
5	Розробка методу	01.05.22-14.05.22	
6	Програмна реалізація	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-26.05.22	
8	Перевірка на плагіат	4.06.22	
9	Рецензування	4.06.22	
10	Підготовка презентації та доповіді	27.05.22-08.06.22	
11	Занесення роботи в електронний архів	3.06.22	
12	Попередній захист кваліфікаційної роботи	13.06.22	

Дата видачі завдання 18 квітня 2022 р.

Студент ______________________________

(підпис)

Керівник роботи ______________________ _____ст. викл.Путятіна О.Є._

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 41 с., 6 рис., 30 джерел.

WEBSOCKET, REACT, NODE.JS, EXPRESS, SOCKET.IO, PEERJS, WEBRTC.

Об’єктом роботи є сервіс для комунікації у реальному часі.

Метою роботи є розроблення вебзастосунку для комунікації в режимі реального часу з використанням протоколу WebSocket.

Використано бібліотеки та платформи з відкритим кодом, які були ствроені для взаємодії з протоколом Websocket та іншими технологіями для передачі інформації у реальному часі.

У результаті роботи була розроблена система для комунікації в режимі реального часу за допомогою платформи Node.js, бібліотеки React та протоколу Websocket.

WEBSOCKET, REACT LIBRARY, NODE.JS PLATFORM, EXPRESS, SOCKET.IO, PEERJS, WEBRTC.

The object of work is a service for real-time communication.

The aim of the work is to develop a web application for real-time communication using the WebSocket protocol.

Open source libraries and platforms have been created that interact with Websocket and other technologies to transmit real-time information.

As a result, a system for real-time communication was developed using the Node.js platform, the React library and the Websocket protocol.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів…….4
Вступ…………………………………………………………………………..5
1 Теоретичні аспекти комунікації в реальному часі………………………....6
1.1 Зв’язок у реальному часі (RTC)………..6
1.1.1 Еволюція комунікацій реального часу………........................7
1.1.2 Приклади комунікацій реального часу………………………7
1.1.3 Існуючі застосунки з комунікацією реального часу………...8
1.1.4 Спілкування в Інтернеті (WebRTC)………………………...8
1.2 Протокол WebSocket………………………………………………..9
1.2.1 Приклади використання протоколу WebSocket…………….10
1.2.2 Порівняння WebSocket та HTTP……………………………..11
1.3 Постановка задачі……………………………………………………12
2 Проектування системи комунікації в реальному часі…………………….13
2.1 Архітектура системи………………………………………………....13
2.2 Клієнтський сервіс…………………………………………………...16
2.3 Серверний сервіс……………………………………………………..20
3 Програмна реалізація системи для комунікації в реальному часі……….23
3.1 Інструментальні засоби реалізації…………………………………..23
3.2 Програмна реалізація клієнтської частини…………………………25
3.3 Програмна реалізація серверної частини…………………………...36
3.4 Перспективи подальшої роботи……………………………………..40
Висновки…………………………………………………….........................41
Перелік джерел посилання……………………………………………………42
Додаток А Тестові зображення……………………………………………….44

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP − HyperText Transfer Protocol (протокол передачі даних)

HTTPS – HyperText Transfer Protocol Secure

HTML – HyperText Markup Language (мова розмітки гіпертексту)

API – Application Programming Interface (програмний інтерфейс програми)

VoIP – voice over IP (голос через IP)

RTC – Real-time communication (комунікація в реальному часі)

DOM – Document Object Model (об'єктна модель документа)

REST – Representational State Transfer (передача репрезентативного стану)

PSTN – Public switched telephone network (телефонна мережа загального користування)

TCP – Transmission Control Protocol (протокол керування передаванням)

JSX – JavaScript Syntax Extension

UI – User interface (інтерфейс користувача)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

IM – Instant messaging (миттєві повідомлення)

XML – Extensible Markup Language (розширювана мова розмітки)

P2P – Peer-to-peer (рівний до рівного)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

CORS – Cross-origin resource sharing (спільне використання ресурсів з різних джерел)

GUI – Graphical user interface (графічний інтерфейс користувача)

IP – Internet Protocol (унікальний числовий номер)

MVC – Model-view-controller (модель-вид-контролер)

ВСТУП

Якщо є щось спільне між підприємствами в різних країнах і галузях, так це спілкування в реальному часі. Незалежно від того, чи потрібно це дозволити співробітникам обмінюватися ідеями та працювати разом, або віддаленим працівникам, щоб отримати негайну допомогу від експертів, керівників служби підтримки клієнтів, щоб відповісти на запити, чи обслуговуючого персоналу для вирішення проблем із обладнанням – спілкування в реальному часі стало невід’ємним аспектом сучасного бізнесу.

Щоб мати можливість безперебійно вести бізнес, розширювати можливості співробітників і задовольняти потреби клієнтів, організації будь-якого розміру впроваджують комунікаційні платформи на основі SaaS для просування вперед.

Оскільки клієнти починають очікувати миттєвого задоволення від брендів, у яких вони купують продукти та послуги, спілкування в реальному часі забезпечує миттєвість, якої клієнти звикли очікувати від бізнесу. Спілкування в реальному часі (RTC) відноситься до будь-якого спілкування, яке відбувається між двома (або більше) особами в режимі реального часу – з мінімальною затримкою і без затримок передачі. Деякі приклади спілкування в реальному часі включають стаціонарні телефони, мобільні дзвінки, обмін миттєвими повідомленнями, VoIP та відеоконференції.

Механізм зв’язку «рівний до рівного», інструменти спілкування в режимі реального часу дозволяють співробітникам спілкуватися один з одним – незалежно від того, де вони географічно розташовані. Спілкування в режимі реального часу – це ідеальне поєднання відео та аудіотехнологій, яке дозволяє співробітникам підключатися в режимі реального часу, щоб співпрацювати, вирішувати проблеми клієнтів або бізнесу та підтримувати зв’язок один з одним.

1 ТЕОРЕТИЧНІ АСПЕКТИ КОМУНІКАЦІЇ В РЕАЛЬНОМУ ЧАСІ

1.1 Зв'язок у реальному часі (RTC)

Зв'язок у реальному часі (RTC) – це будь-який вид телекомунікацій, у якому всі користувачі можуть обмінюватися інформацією миттєво або з незначною затримкою або затримкою передачі. У цьому контексті термін реальний час є синонімом живого.

У RTC завжди є прямий шлях між джерелом і призначенням. Хоча посилання може містити кілька проміжних вузлів, дані переходять від джерела до місця призначення, не зберігаючись між ними. На відміну від цього, асинхронні комунікації або комунікації зі зміщенням у часі, такі як електронна пошта та голосова пошта, завжди передбачають певну форму зберігання даних між джерелом і призначенням. У цих випадках існує очікувана затримка між передачею та отриманням інформації.

Спілкування в режимі реального часу може відбуватися в напівдуплексному або повнодуплексному режимах:

- напівдуплексний – передача даних може відбуватися в обох напрямках на одному носії або ланцюзі, але не одночасно;
- повнодуплексний – передача даних може відбуватися в обох напрямках одночасно на одному носії або ланцюзі.

RTC зазвичай відноситься до однорангового зв'язку (P2P), а не до широкомовної або багатоадресної передачі.

1.1.1 Еволюція комунікацій реального часу

Розвиток комутованої телефонної мережі загального користування (PSTN) на початку 1900-х років представив RTC американським масам, радикально змінивши природу міжміського зв'язку. У 1915 році дебют нової трансконтинентальної телефонної лінії означав, що вперше в історії користувачі, розділені на відстані понад 3000 миль, могли вести інтерактивну розмову, наче вони перебувають в одній кімнаті. Пізніше, у 20 столітті, високошвидкісний Інтернет, мобільний телефон і розумні пристрої зробили ще більшу революцію в комунікації в реальному часі, дозволивши обмінюватися миттєвими повідомленнями (IM), телефонувати за протоколом Інтернету (IP), відеодзвінки, відеоконференції тощо.

1.1.2 Приклади комунікацій реального часу

Інструментів і додатків для зв’язку в режимі реального часу багато й різноманітні, починаючи від старовинної телефонії до хмарних комунікаційних послуг.

Наприклад:

- фіксована телефонія;
- мобільна телефонія;
- голос через IP (VoIP);
- фіксована телефонія;
- телеконференції;
- відеодзвінки;
- відеоконференції;
- спільне використання екрана.

1.1.3 Існуючі застосунки з комунікацією реального часу

Застосунки, які використовують технологію RTC мають багато відмінностей, тому, що компанії, які розробляють ці застосунки можуть бути розміром з невеликий стартап або з велику телекомунікаційну компанію. Основні продукти та послуги зв'язку в реальному часі:

- Adobe Connect;
- AT&T Collaborate;
- Google Meet;
- Microsoft Teams;
- Zoom Meetings;
- Slack.

1.1.4 Спілкування в Інтернеті (WebRTC)

WebRTC (Web Real Time Communications) – це стандарт, який забезпечує одноранговий зв’язок у режимі реального часу та обмін медіаданими у браузерах, усуваючи потребу завантажувати та встановлювати додаткові програми чи додатки.

За допомогою WebRTC можна додавати можливості зв’язку в реальному часі до власних програм, які працюють поверх стандарту спілкування в Інтернеті. Він підтримує відео, голос і загальні дані для передачі інформації, що дозволяє розробникам створювати потужні рішення для голосового та відеозв’язку. Технологія доступна у всіх сучасних браузерах. Технології WebRTC реалізовані як відкритий вебстандарт і доступні як звичайні API у всіх основних браузерах. Проект WebRTC є відкритим вихідним кодом і підтримується Apple, Google, Microsoft і Mozilla.

1.2 Протокол WebSocket

WebSocket є двонаправленим, повнодуплексним протоколом, який використовується в тому ж сценарії спілкування клієнт-сервер, на відміну від HTTP, він починається з ws:// або wss://. Це протокол із збереженням стану, що означає, що з’єднання між клієнтом і сервером буде існувати, поки його не розірве будь-яка сторона (клієнт або сервер). Після закриття з’єднання клієнтом і сервером з’єднання припиняється з обох сторін.

Давайте візьмемо приклад спілкування клієнт-сервер, є клієнт, який є браузером і сервером, щоразу, коли ми ініціюємо з'єднання між клієнтом і сервером, клієнт-сервер зробив рукостискання і вирішив створити нове з’єднання і це з'єднання залишиться в живих, поки не буде припинено будь-яким із них. Коли з’єднання встановлено і діє, зв’язок відбувається за допомогою того самого каналу з’єднання, доки не буде розірвано (рис. 1.1).

Рисунок 1.1 – Приклад зв’язку за допомогою WebSocket

Ось як після зв’язку клієнт-сервер клієнт-сервер вирішує встановити нове з’єднання, щоб зберегти його живим, це нове з’єднання буде відоме як WebSocket. Після встановлення каналу зв'язку та відкриття з'єднання обмін

повідомленнями буде відбуватися в двонаправленому режимі, доки з'єднання між клієнтом-сервером не буде зберігатися. Якщо хтось із них (клієнт-сервер) загине або вирішить закрити з’єднання, обидві сторони закривають.

Спосіб роботи сокета дещо відрізняється від того, як працює HTTP, код статусу 101 позначає протокол перемикання в WebSocket.

Більшість популярних браузерів підтримують цей протокол, наприклад:

- Google Chrome;
- Safari;
- Opera;
- Firefox;
- Microsoft Edge;
- Internet Explorer.

1.2.1 Приклади використання протоколу WebSocket

Існує багато сфер, де застосовується протокол WebSocket. Взагалі це ті сфери, де є критичним час між передачею даних між клієнтом та сервером:

− вебзастосунок у режимі реального часу: вебзастосунок у режимі реального часу використовує вебсокет для відображення даних на стороні клієнта, які постійно надсилаються серверним сервером. У WebSocket дані безперервно передаються/передаються в те саме з’єднання, яке вже відкрите, тому WebSocket працює швидше та покращує продуктивність програми. Наприклад біржа, або криптоволютна біржа у якій дані постійно передаються між бекенд-сервером та клієнтом;

− ігри: в ігровій програмі можна зосередитися на тому, що дані постійно отримуються сервером, і без оновлення інтерфейсу користувача вони почнуть діяти на екрані, інтерфейс користувача автоматично оновлюється, навіть не встановлюючи нове з’єднання, тому це дуже корисно в ігровій програмі;

− чат: програми чату використовують WebSockets для встановлення з'єднання лише один раз для обміну, публікації та трансляції повідомлення серед передплатників. Він повторно використовує те саме з'єднання WebSocket для надсилання та отримання повідомлення та для їх передачі.

WebSocket можна використовувати, якщо ми хочемо, щоб у реальному часі оновлювалися або безперервні потоки даних, які передаються по мережі. Якщо ми хочемо отримати старі дані або хочемо отримати дані лише один раз, щоб обробити їх за допомогою програми, ми повинні використовувати протокол HTTP, старі дані, які не потрібні дуже часто або отримані лише один раз, можуть бути запитані за допомогою простого запиту HTTP, тому в цьому випадку краще не використовувати WebSocket.

1.2.2 Порівняння WebSocket та HTTP

WebSocket та HTTP користуються більшість Інтернету. Незважаючи на те, що протокол HTTP з'явився набагато раніше, протокол WebSocket при вирішенні деяких проблем набуває більшої переваги від розробників програмного забезпечення.

Визначимо перелік характеристик протокола HTTP, для того, шоб показати різницю з WebSocket.

Протокол HTTP – це односпрямований протокол, який працює поверх протоколу TCP, який є протоколом транспортного рівня, орієнтованим на з'єднання. Ми можемо створити з'єднання за допомогою методів запиту HTTP після закриття відповідного HTTP-з'єднання.

Проста програма RESTful використовує протокол HTTP, який не має стану.

Коли не потрібно зберігати з'єднання протягом певного часу або повторно використовувати з'єднання для передачі даних.

HTTP-з'єднання повільніше, ніж WebSockets.

1.3 Постановка задачі

Об’єктом роботи є сервіс для комунікації у реальному часі.

Метою роботи є розроблення вебзастосунку для комунікації в режимі реального часу з використанням протоколу WebSocket.

Додаткових вимог до користувачів застосунка не має тому, що користувач може взаємодіяти з застосунком за допомогою браузера та більшість існуючих браузерів підримують усі технології використані для створення застосунку.

Для досягнення мети необхідно виконати наступні дії:

- вивчити теоретичні поняття: WebSocket, RTC, WebRTC;
- створити клієнтський застосунок;
- створити серверний застосунок;
- створити реалізацію комунікації у реальному часі за допомогою WebRTC та Peerjs;
- створити взаємодію між клієнтською та серверною частиною сервісу за допомогою протоколу WebSocket.

2 ПРОЕКТУВАННЯ СИСТЕМИ КОМУНІКАЦІЇ В РЕАЛЬНОМУ ЧАСІ

2.1 Архітектура системи

Вже багато років створюються і вдосконалюються системи. За ці роки з’явилося кілька технологій, архітектурних моделей та передового досвіду. Мікросервіси є одним із тих архітектурних шаблонів, які виникли у світі дизайну, керованого доменом, безперервної доставки, автоматизації платформ та інфраструктури, масштабованих систем, поліглотного програмування та стійкості.

Архітектура мікросервісів використовує той самий підхід і поширює його на слабо пов’язані служби, які можна розробляти, розгортати та підтримувати незалежно. Кожна з цих служб відповідає за окреме завдання і може взаємодіяти з іншими службами через прості API для вирішення більшої складної бізнес-задачі.

Оскільки складові служби невеликі, вони можуть бути створені однією або кількома невеликими командами, що полегшує масштабування зусиль з розробки, якщо це необхідно.

Після розробки ці служби також можуть бути розгорнуті незалежно один від одного, тому їх легко ідентифікувати гарячі служби та масштабувати їх незалежно від усієї програми. Мікросервіси також пропонують покращену ізоляцію помилок, завдяки чому в разі помилки в одній службі вся програма не обов’язково припиняє функціонувати. Коли помилку виправлено, її можна буде розгорнути лише для відповідної служби замість повторного розгортання всієї програми.

Ще одна перевага, яку має архітектура мікросервісів, полягає в тому, що вона полегшує вибір технологічного стека (мови програмування, бази даних тощо), який найкраще підходить для необхідної функціональності (сервісу),

замість того, щоб використовувати більш стандартизований та універсальний підхід.

Є організації, які досягли успіху з мікросервісами і дотримувались моделі, коли команди, які створюють сервіси, піклуються про все, що стосується цієї служби. Саме вони розробляють, розгортають, підтримують і підтримують його. Немає окремих команд підтримки чи обслуговування.

Інший спосіб досягти того ж – мати внутрішню модель з відкритим кодом. Використовуючи цей підхід, розробник, якому потрібні зміни в службі, може перевірити код, попрацювати над функцією та подати PR самостійно, замість того, щоб чекати, поки власник служби забере і попрацює над необхідними змінами. Щоб ця модель працювала належним чином, необхідна відповідна технічна документація, а також інструкції з налаштування та вказівки для кожної служби, щоб будь-кому було легко отримати послугу та працювати з нею.

Ще одна прихована перевага цього підходу полягає в тому, що він зосереджує розробників на написанні високоякісного коду, оскільки вони знають, що інші будуть дивитися на нього. Є також деякі архітектурні шаблони, які можуть допомогти в децентралізації речей. Наприклад, у вас може бути архітектура, де набір служб обмінюється інформацією через центральну шину повідомлень.

Незважаючи на швидкість ведення бізнесу сьогодні та розвиток нових технологій, які роблять управління мікросервісами ще простішим, список переваг стає довшим, ніж недоліків, але деякі недоліки все ж є. Хоча значна частина процесу розробки спрощена за допомогою мікросервісів, є кілька областей, де мікросервіси насправді можуть спричинити нову складність.

Приклад недоліків мікросервісів:

− розробники повинні мати справу з додатковою складністю створення розподіленої системи;

− розробники повинні впровадити механізм міжсервісного зв’язку та впоратися з частковими збоями;

− реалізувати запити, які охоплюють декілька служб, є більш складними;

− реалізація запитів, які охоплюють декілька служб, вимагає ретельної координації між командами

− складніше тестувати взаємодію між службами.

Переваги та недоліки архітектури мікросервісів значно відрізняються від традиційної монолітної архітектури, і ця модель не ідеальна для кожної організації. Однак велика увага до цього модульного архітектурного стилю відбувається неспроста – все більше підприємств усвідомлюють необхідність швидшої, легшої та гнучкішої розробки додатків, а мікросервіси дозволяють це зробити так, як монолітна архітектура просто не може.

Після врахування переваг та недоліків мікросервісної архітектури, зазначимо, які мікросервіси необхідно створити для того, щоб реалізувати стабільну та легку в підтримкі систему для комунікації в реальному часі.

Для того, щоб забезпечити стабільну мікросервісну архітектуру для комунікації в Інтернеті, необхідно створити клієнтський сервіс (frontend service) та серверний сервіс (backend service).

Створимо схему того, як будуть взаємодіяти серверний та клієнтський сервіс між собою (рис. 2.1).

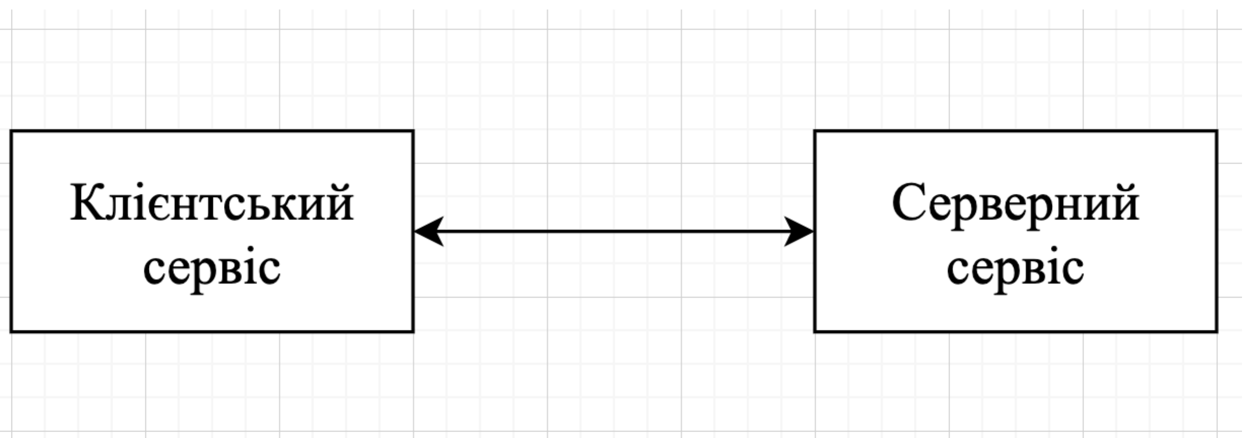

Рисунок 2.1 – Приклад мікросервісної архітектури та зв’язку між двома сервісами

2.2 Клієнтський сервіс

Клієнтська частина мікросервісної архітектури є дуже важливою частиною всієї системи, тому, що юзер не може побачити, що відбувається на стороні серверу.

Треба приділити особу увагу створенню зрозумілого та простого інтерфейсу користувача.

Кожного разу, коли користувач натискає клавішу, введений символ повинен бути перевірений і викинутий, якщо це не число або клавіша переміщення курсору. Уявіть собі затримку, якби програмі довелося чекати зворотного зв'язку між клієнтом і сервером і перемалювати сторінку для кожного натискання клавіші. Очевидно, що така функція ніколи не була б практичною у вебзастосунку, якби не сценарії на стороні клієнта. Скрипти на стороні клієнта дозволяють легко інтегрувати багато функцій у вебпрограми, які в іншому випадку були б непрактичними.

Крім того, сценарії на стороні клієнта дають перевагу розвантаження частини обчислювальних вимог програми від сервера до клієнта. Комп'ютер на стороні клієнта зазвичай витрачає більшу частину свого потенціалу обробки, просто чекаючи, поки користувач щось зробить. Вебсервери, з іншого боку, зазвичай обкладаються податком до ліміту, обслуговуючи сотні чи тисячі одночасних користувачів. За ці роки з'явилося кілька технологій, архітектурних моделей та передового досвіду. Мікросервіси є одним із тих архітектурних шаблонів, які виникли у світі дизайну, керованого доменом, безперервної доставки, автоматизації платформ та інфраструктури, масштабованих систем, поліглотного програмування та стійкості.

Однією з найпопулярніших бібліотек для використання готових та стандартизованих вебкомпонентів є Material UI.

Material UI – це найпотужніший та найефективніший інструмент для створення додатків шляхом додавання дизайнів та анімацій та використання

їх із технічними та науковими інноваціями. По суті, це мова дизайну, розроблена компанією Google у 2014 році. Вона використовує більше дизайну та анімації, сітку та забезпечує ефекти тіней.

Його можна використовувати з усіма фреймворками JavaScript, такими як AngularJS, VueJS, і бібліотеками, такими як ReactJS, щоб зробити додаток більш дивовижним і чуйним. Маючи понад 35 000 зірок на GitHub, тобто Material UI є однією з найкращих бібліотек інтерфейсу користувача для React.

Для того, щоб зробити інтерактивними вебкомпоненти, необхідно застосувати мову програмування JavaScript. Через те, що створення інтерактивних веб сторінок без використання сторонніх бібліотек, може займати багато часу та можливо може бути не настільки стабільним аніж користуючись ними.

React – це бібліотека для створення компонованих інтерфейсів користувача. Це заохочує до створення повторно використовуваних компонентів інтерфейсу користувача, які представляють дані, які змінюються з часом. Багато людей використовують React як V в MVC. React абстрагує від DOM, пропонуючи просту модель програмування та кращу продуктивність.

DOM - спосіб представлення структурного документа за допомогою об'єктів. Це кросплатформна та мовно-незалежна угода для представлення та взаємодії з даними у HTML, XML тощо.

Браузери обробляють складові DOM, і ми можемо взаємодіяти з ними, використовуючи JavaScript та CSS. Ми можемо працювати з вузлами документа, змінювати їх дані, видаляти та вставляти нові вузли. Зараз DOM API є практично кросплатформним та кросбраузерним.

Спочатку DOM був призначений для статичних інтерфейсів
користувача - сторінок, які відображаються сервером, які не потребують динамічних оновлень. Коли DOM оновлюється, він повинен оновити вузол, а також перефарбувати сторінку з відповідним CSS і макетом.

Віртуальний DOM - це легка абстракція DOM. Можна розглядати це як копію DOM, яку можна оновлювати, не впливаючи на DOM. Він має всі ті ж

властивості, що й справжній об’єкт DOM, але не має можливості записувати на екран, як справжній DOM. Віртуальний DOM отримує швидкість та ефективність завдяки тому, що він легкий. Фактично, новий віртуальний DOM створюється після кожного повторного візуалізації.

Узгодження - це процес порівняння та синхронізації двох файлів (реального та віртуального DOM). Алгоритм розрізнення - це техніка узгодження, яку використовує React.

JSX - є розширенням синтаксису JavaScript. React визнає той факт, що логіка візуалізації поєднана з логікою інтерфейсу користувача: як обробляються події, як змінюється стан з часом і як дані готуються до відображення.

Замість того, щоб штучно розділяти технології шляхом розміщення розмітки та логіки в окремих файлах, React розділяє проблеми за допомогою слабо пов’язаних одиниць, які називаються «компонентами», які містять обидва.

Використовувати JSX у розробці React не обов’язково, але рекомендується.

Кожен інтерфейс користувача в додатку React є компонентом. Компоненти – це незалежні фрагменти коду, які можна використовувати повторно. Одна програма може мати кілька компонентів.

Компоненти можуть бути двох типів: компоненти класу та функціональні компоненти. Компоненти класу зберігають стан, оскільки вони можуть використовувати свій «стан» для зміни інтерфейсу користувача. Функціональні компоненти є компонентами без стану. Вони діють як функція JavaScript, яка може приймати довільний параметр під назвою "props".

Нещодавно були введені React Hooks для використання стану у функціональних компонентах.

React також може відтворюватись на сервері за допомогою Node, і він може запускати власні програми за допомогою React Native. React реалізує

односторонній реактивний потік даних, що зменшує шаблон і легше міркувати, ніж традиційні дані.

Особливості React:

− використовує JSX;

− необхідно думати про всі елементи зовнішнього інтерфейсу сайту, як про компонент. Це допоможе підтримувати код під час роботи над великими проектами;

− React реалізує односторонній потік даних, що дозволяє легко міркувати про вашу програму. Flux – це шаблон, який допомагає підтримувати однонаправленість ваших даних;

− React розробляється компанією Meta.

Переваги React:

− використовує віртуальний DOM, який є об’єктом JavaScript. Це покращить продуктивність програм, оскільки віртуальна DOM JavaScript швидше, ніж звичайна DOM;

− може використовуватися на стороні клієнта і сервера, а також з іншими фреймворками;

− шаблони компонентів і даних покращують читабельність, що допомагає підтримувати більші програми.

Недоліки React:

− охоплює лише рівень перегляду програми, тому все одно потрібно вибрати інші технології, щоб отримати повний набір інструментів для розробки;

− використовує вбудовані шаблони та JSX, що деяким розробникам може здатися незручним.

Для того щоб сповістити сервер про дію користувача за допомогою протоколу WebSocket необхідно застосувати бібліотеку Socket.IO, яка має бути присутня не тільки на стороні клієнтського сервісу, а на стороні серверної частини також.

Socket.IO – це бібліотека, яка забезпечує двонаправлений зв’язок між клієнтом і сервером із низькими затримками та на основі подій. Вона побудована на основі протоколу WebSocket і надає додаткові гарантії, такі як відхід до тривалого опитування HTTP або автоматичне повторне підключення.

Передача аудіо та відео інформації за допомогою WebRTC буде здійснена бібліотекою PeerJS.

PeerJS обгортає реалізацію WebRTC браузера, щоб забезпечити повний, настроюваний і простий у використанні API однорангового підключення. Не має нічого, крім ідентифікатора, вузол може створити з’єднання з віддаленим вузлом P2P або медіа-потоком.

2.3 Серверний сервіс

Серверна частина мікросервісної архітектури, незважаючи на те, що користувач не може її побачити, теж має вплив на те яке враження він отримає від користування додатком.

Для створення серверної частини вебзастосунка обрано платформу Node.js, яка дозволяє створювати серверні додатки за допомогою мови програмування JavaScript, який до створення платформи Node.js використовувався тільки для створення клієнтських додатків.

Node.js – це міжплатформне середовище виконання JavaScript із відкритим вихідним кодом, яке працює на движку V8 і виконує код JavaScript за межами браузера. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для сценаріїв на стороні сервера – запуск сценаріїв на стороні сервера для створення динамічного вмісту вебсторінки, перш ніж сторінка буде відправлена у браузер користувача.

Завдяки Node.js можна створити базовий серверний сервіс, але для більш стабільної работи, додаткових функцій та спрощення процесу розробки потрібно застосовувати фреймворк на основі цієї платформи.

ExpressJS – це фреймворк, який надає вам простий API для створення вебсайтів, вебпрограм і серверних програм. З ExpressJS не потрібно турбуватися про низькорівневі протоколи, процеси тощо.

Особливості Express Framework:

− Express можна використовувати для розробки односторінкових, багатосторінкових і гібридних вебзастосунків;

− Express дозволяє налаштувати проміжне програмне забезпечення для відповіді на HTTP-запити;

− Express визначає таблицю маршрутизації, яка використовується для виконання різних дій на основі методу HTTP та URL-адреси;

− використовує функції Middleware.

Express дозволяє динамічно відображати сторінки HTML на основі передачі аргументів шаблонам.

Коли потрібно визначити маршрут в Express, функція-обробник маршруту, що вказується для цього маршруту, є функцією Middleware.

У Express Middleware - це певний стиль функцій, які ви налаштовуєте для використання вашим додатком. Вони можуть виконувати будь-який код, який вам подобається, але зазвичай вони заботяться об обробці вхідних запитів, відправці відповідей та обробці помилок. Вони є будівельними блоками кожного додатка Express.

Функції Middleware є досить гнучкими. Можна вказати, що Express запускає одну і ту саму функцію проміжного програмного забезпечення для різних маршрутів, що дозволяє робити такі речі, як загальну перевірку для різних кінцевих API-ендпоінтів.

Окрім опису власних функцій Middleware, також можна встановити сторонні для використання у власному застосунку. У документації Express наведено деякі популярні модулі, які містять функції Middleware.

Додатки створені за допомогою фреймворку Express працюють, надсилаючи послідовність запитів на середній рівень. Проміжне програмне забезпечення має двокурсовий доступ до об’єктів запиту та об’єктів відповіді. Це означає, що використання платформи Express дає вам контроль над усіма об’єктами запиту та відповіді. Це дає вам можливість додавати сеанси, додавати параметри публікації та шаблон за допомогою ejs.

Для того, щоб створити WebSocket з’єднання з клієнтською частиною додатку, необхідно додати бібліотеку Socket.IO на серверну частину. Серверна частина буде отримувати повідомлення про дії користувачів та також сповіщати інших користувачів про події які були зроблені на стороні серверу.

Якщо серверна та клієнтська частина додатку розташовані не на одному домені, що звично для додатка, який використовує мікросервісну архітектуру, виникає проблема зв’язана за передачею даних між клієнтом та сервером. Цю проблему можна вирішити за допомоги пакету CORS, завдяки якому можна вказати з яких доменів може прийматися інформація на сервері.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ КОМУНІКАЦІЇ В РЕАЛЬНОМУ ЧАСІ

3.1 Інструментальні засоби реалізації

Visual Studio Code – це дуже потужний і простий у використанні редактор коду. Він поставляється з широкою підтримкою мови програмування, легко настроюється за допомогою різних розширень і є безкоштовним.

Основні характеристики редактору:

- має інтегрований термінал;
- має графічний інтерфейс;
- має багато разшорень;
- підсвічування синтаксису;
- безкоштовний;
- код редактору знаходиться у відкритому доступі.

Postman – це інструмент для тестування інтерфейсу прикладного програмування (API). API діє як інтерфейс між кількома додатками і встановлює зв’язок між ними.

Postman був розроблений у 2012 році розробником програмного забезпечення та підприємцем Абхінавом Астханою, щоб спростити розробку та тестування API. Це інструмент для тестування програмного забезпечення API. Його можна використовувати для проектування, документування, перевірки, створення та зміни API.

Postman має функцію надсилання та спостереження за запитами та відповідями протоколу передачі гіпертексту (HTTP). Він має графічний інтерфейс користувача (GUI) і може використовуватися на таких платформах, як Linux, Windows і Mac. Він може створювати кілька HTTP-запитів – POST, PUT, GET, PATCH і перекладати їх у код. Також за допомогою додатка Postman можна тестувати API, які використовують протокол WebSocket.

Основні характеристики Postman:

- поставляється без будь-яких ліцензійних витрат і підходить для використання командами з будь-якою місткістю;
- можна використовувати дуже легко, просто завантаживши його;
- дозволяє легко підтримувати тестові набори за допомогою колекцій, користувачі можуть здійснювати набір викликів API, які можуть мати різноманітні запити та підпапки;
- дає можливість імпортувати/експортувати середовища та колекції, що дозволяє легко обмінюватися файлами;
- безкоштовний;
- має велику підтримку громади.

3.2 Програмна реалізація клієнтської частини

Для того, щоб розпочати програмну реалізацію клієнтської частини необхідно заздалегідь визначити необхідні модулі для створення додатка за допомогою бібліотеки React.

Під час створення додатку необхідно створити файл з назвою package.json (рис. 3.1), який містить у собі бібліотеки для ініціалізації клієнтської частини.

```
{
    "name": "client",
    "version": "0.1.0",
    "private": true,
    "dependencies": {
        "@material-ui/core": "^4.11.3",
        "@material-ui/icons": "^4.11.2",
        "@testing-library/jest-dom": "^5.11.4",
        "@testing-library/react": "^11.1.0",
        "@testing-library/user-event": "^12.1.10",
        "add": "^2.0.6",
        "peerjs": "^1.3.1",
        "react": "^17.0.1",
        "react-copy-to-clipboard": "^5.0.3",
        "react-dom": "^17.0.1",
        "react-scripts": "4.0.3",
        "simple-peer": "^9.10.0",
        "socket.io-client": "^4.0.0",
        "web-vitals": "^1.0.1",
        "yarn": "^1.22.10"
    },
    ▷ Debug
    "scripts": {
        "start": "react-scripts start",
        "build": "react-scripts build",
        "test": "react-scripts test",
        "eject": "react-scripts eject"
    },
    "eslintConfig": {
        "extends": [
            "react-app",
            "react-app/jest"
        ]
    },
    "devDependencies": {
        "eslint": "^7.22.0",
        "eslint-config-airbnb": "^18.2.1",
        "eslint-plugin-import": "^2.22.1",
        "eslint-plugin-jsx-a11y": "^6.4.1",
        "eslint-plugin-react": "^7.22.0",
        "eslint-plugin-react-hooks": "^4.2.0"
    }
}
```

Рисунок 3.1 – Файл package.json, який містить бібліотеки

Створимо файлову структуру клієнтської частини (рис. 3.2), яка містить:

– папку public, де знаходяться статичні компоненти вебзастосунку;

– папку src, яка містить функціональну частину застосунку, а також компоненти, з яких він складається;

– файл eslintrc.js, який містить правила форматування коду;

– файл Readme.md, який містить опис клієнтської частини вебзастосунку;

– файл package.json, який містить перелік та версії бібліотек та модулів використані застосунком;

– папку node_modules, яка містить бібліотеки використані застосунком;

– файл package-lock.json, який містить залежності бібліотек.

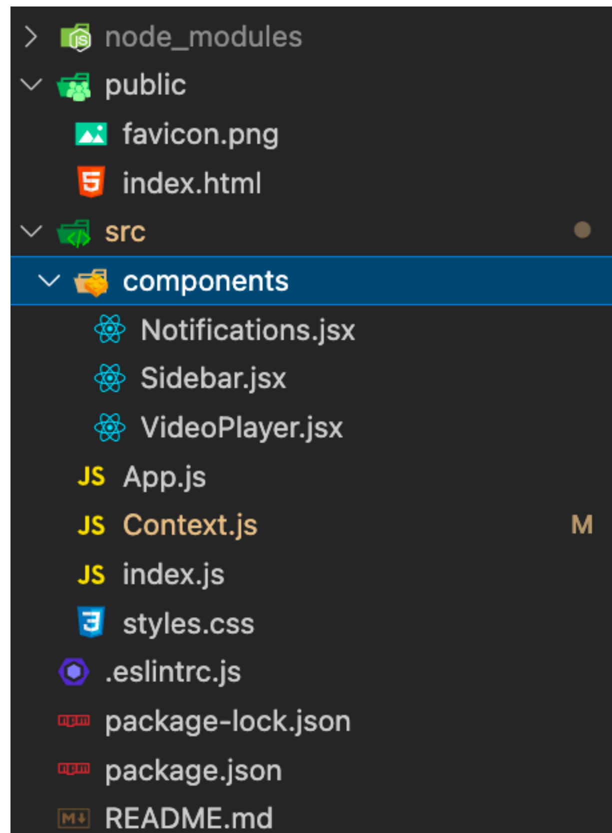

Рисунок 3.2 – Файлова структура клієнтської частини

Вхідною точкою клієнтської частини є рендерінг головного компоненту програми.

Лістинг 3.1 Фрагмент програмного коду з рендерінгом головного компонента програми:

```
ReactDOM.render(
    <ContextProvider>
        <App />
    </ContextProvider>,
    document.getElementById('root'),
);
```

Необхідно створити HTML структуру клієнтської частини, у яку буде динамічно додаватися HTML елементів застосунку, який автоматично буде згенерований за допомогою засобів React.

Лістинг 3.2 HTML структура:

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="utf-8" />
	<link rel="icon" href="favicon.png" />
	<meta name="viewport" content="width=device-width, initial-
scale=1" />
	<meta name="theme-color" content="#000000" />
	<meta name="description" content="video chat" /> <title>Video
	Chat</title>
</head>
<body>
	<div id="root">
	</div>
 </body>
</html>
```

Створимо стилі CSS для клієнтської частини, також створимо стилі засобами бібліотеки Material UI та React. Стилі створені за допомогою Material UI будуть використані в загальному компоненті та в компонентах, які він містить.

Лістинг 3.3 CSS структури клієнтської частини:

```
body {
```

```
    background-image: linear-gradient(#303F9E , lightblue);
}
 * {
    margin: 0;
    padding: 0;
}
```

Лістинг 3.4 Стилі створені бібліотекою Material UI:

```
const useStyles = makeStyles((theme) => ({
   appBar: {
      borderRadius: 0,
      margin: '30px 100px',
      display: 'flex',
      flexDirection: 'row',
      justifyContent: 'center',
      alignItems: 'center',
      width: '600px',
      border: '2px solid black',
      [theme.breakpoints.down('xs')]: {
         width: '90%',
```

```
    },
  },
  wrapper: {
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
    width: '100%',
  },
}));
```

Створимо загальний компонент, який містить усі інші компоненти та передає їм необхідну інформацію для відображення даних та взаємодії з користувачем. Також застосуємо створені стилі для загального компоненту.

Лістинг 3.4 Код загального компоненту:

```
export const App = () => {
  const classes = useStyles();
  return (
    <div className={classes.wrapper}>
      <AppBar className={classes.appBar} position="static" color="inherit">
        <Typography variant="h2" align="center">Video chat</Typography>
```

```
      </AppBar>
      <VideoSection />
      <UserPanel>
        <IncomingCallNotifications />
      </UserPanel>
    </div>
  );
};
```

Потрібно створити контекст, який буде містить засоби, які будуть змінювати стан застосунку та взаємодіяти з серверною частиною вебзастосунку.

Лістинг 3.5 React контекст:

```
export const SocketContext = createContext();
```

Провайдер контексту має містити базовий стан застосунку та функції, які його змінюють.

Лістинг 3.6 Базовий стан застосунку:

```
const [callAccepted, setCallAccepted] = useState(false);
  const [callEnded, setCallEnded] = useState(false);
  const [stream, setStream] = useState();
```

```
const [name, setName] = useState('');
const [callInfo, setCallInfo] = useState({});
const [currentUserId, setCurrentUserId] = useState('');
const currentUserVideoStream = useRef();
const callerVideoStream = useRef();
const refToConnection = useRef();
```

Необхідно додати основний функціонал вебзастосунку, який буде відповідати за обробку дій зв’язаних зі створенням відеодзінків. Створимо функцію, яка буде обробляти вхідний дзвінок.

Лістинг 3.7 Код, який оброблює вхідний дзвінок, та встановлює відеозв’язок:

```
const answer = () => {
  setCallAccepted(true);
  const peer = new Peer({ initiator: false, trickle: false, stream });
  peer.on('signal', (data) => { socket.emit('answer', { signal: data, to: callInfo.from }); });
  peer.on('stream', (currentStream) => {
    callerVideoStream.current.srcObject = currentStream; });
  }
}
```

Створимо функцію, яка буде здійснювати новий дзвінок, та передавати відеоінформацію між користувачами.

Лістинг 3.8 Код, який здійснює вхідний дзвінок:

```
const startUserCall = (id) => {
  const peer = new Peer({ initiator: true, trickle: false, stream });
  peer.on('stream', (currentStream) => {
    callerVideoStream.current.srcObject = currentStream;
  });
    peer.on('signal', (data) => {
     socket.emit('startUserCall', { userToCall: id, signalData: data, from: currentUserId, name });
     });
    socket.on('callAccepted', (signal) => { setCallAccepted(true); peer.signal(signal); });
    refToConnection.current = peer;
  };
```

Створимо функцію, яка буде закінчувати дзвінок та зміняти стан вебзастосунку.

Лістинг 3.9 Код, який закінчує дзвінок:

```
const endCall = () => {
```

```
    setCallEnded(true);

    refToConnection.current.destroy();

    window.location.reload();

}
```

3.3 Програмна реалізація серверної частини

Для того, щоб розпочати програмну реалізацію серверної частини необхідно заздалегідь визначити необхідні модулі для створення додатка за допомогою фреймфорку Express та бібліотеки Socket.IO.

Під час створення додатку необхідно створити файл з назвою package.json (рис. 3.3), який містить у собі бібліотеки для ініціалізації серверної частини.

```
{
    "name": "videochat",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    ▷ Debug
    "scripts": {
        "start": "node index.js",
        "dev": "nodemon index.js"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "dependencies": {
        "cors": "^2.8.5",
        "express": "^4.17.1",
        "nodemon": "^2.0.7",
        "serve": "^11.3.2",
        "socket.io": "^4.0.0"
    }
}
```

Рисунок 3.3 – Файл package.json, який містить бібліотеки

Створимо файлову структуру клієнтської частини (рис. 3.4), яка містить:

– файл index.js, який містить функціональну частину серверної частини застосунку;

– файл package.json, який містить перелік та версії бібліотек та модулів використані застосунком;

– файл Readme.md, який містить опис клієнтської частини вебзастосунку;

– папку node_modules, яка містить бібліотеки використані застосунком;

– файл package-lock.json, який містить залежності бібліотек.

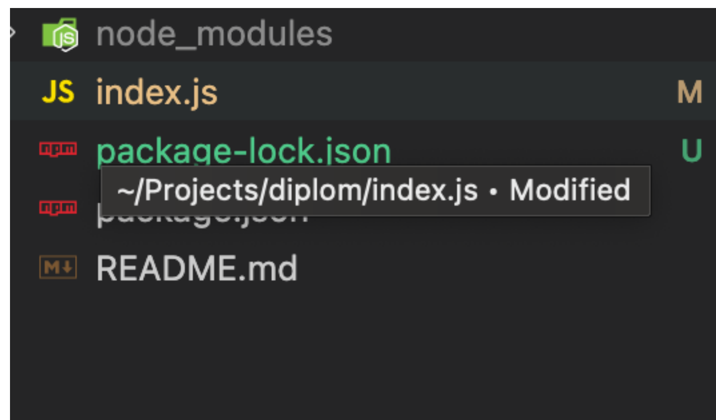

Рисунок 3.4 – Файлова структура серверної частини

Створимо сервер за допомогою фреймворку Express.

Лістинг 3.10 Ініціалізація серверу:

```
const app = require("express")();
const server = require("http").createServer(app);
```

Для визначення порта, на якому сервер буде очікувати запити від клієнтської частини, використаємо змінну середовища, яка має бути визначена заздалегідь в окремому файлі, або під час запуску серверної частини. Якщо такої змінної не буде знайдено, то використаємо порт 3000. Також необхідно зазначити для сервера, що необхідно очікувати запити на цьому порті.

Лістинг 3.11 Створення змінної порта та додавання слухача на цей порт:

```
const PORT = process.env.PORT || 3000;
server.listen(PORT, () => console.log(`Server is running on port ${PORT}`));
```

Необхідно додати модуль Cors, для того, щоб була можливість обміну запитів та відповідей між серверною та клієнтською частиною вебзастосунка. Зазначимо, що клієнтська частина буде мати можливість робити тільки GET та POST запити та ініціалізуємо бібліотеку Socket.IO.

Лістинг 3.12 Додавання модуля Cors та ініціалізація Socket.IO:

```
const cors = require("cors");

const io = require("socket.io")(server, {
	cors: {
		origin: "*",
		methods: [ "GET", "POST" ]
	}
});

app.use(cors());
```

Створимо GET ендпоінт, на який можна буде зробити запит та дізнатися про стан вебзастосунку, також додамо логування запитів для того, щоб мати історію запитів.

Лістинг 3.13 GET ендпоінт:

```
app.get('/', (req, res) => {
	console.log('Server is up');
	res.send('Running');
});
```

Створимо обробку запитів через протокол WebSocket за допомогою бібліотеки Socket.IO, які будуть транслювати повідомлення про новий дзвінок, про те що, користувач відповів на дзвінок, про закінчення дзвінка.

Лістинг 3.14 Обробка запитів через протокол WebSocket:

```
io.on("connection", (socket) => {
```

```
        socket.emit("currentUserId", socket.id);
        socket.on("disconnect", () => {
            socket.broadcast.emit("callEnded")
        });
        socket.on("startUserCall", ({ userToCall, signalData, from, name }) => {
            io.to(userToCall).emit("startUserCall", { signal: signalData, from, name });
        });
        socket.on("answer", (data) => {
            io.to(data.to).emit("callAccepted", data.signal)
        });
    });
```

3.4 Перспективи подальшої роботи

Наразі наявні наступні перспективи розвитку:

- покращити дизайн сторінок вебзастосунку;
- розширити функціонал застосунку;
- додати можливість створювати групові зустрічі для користувачів;
- додати можливість вести текстову комунікацію;
- додати можливість створювати сторінку користувача.

ВИСНОВКИ

У рамках кваліфікаційної роботи було систему комунікації в реальному часі з використанням протоколу WebSocket.

Дана система дозволяє передавати аудіо та відеоінформацію.

В результаті виконання роботи були досліджені різні технології для взаємодії між користувачами в Інтернеті. За результатами дослідження можна сказати, що технології комунікації в режимі реального часу будуть набувати більшої необхідності, тому, що разом зі швидким розвитком Інтернету, технології комунікації мають розвиватись також, для забезпечення стабільної роботи застосунків для користувачів.

Систему було протестовано, за результатами чого було визначено перелік питань для подальшого покращення системи.

За підсумками розробки мета була повністю досягнута. І, як результат, кінцевим продуктом стало повноцінний вебзастосунок для комунікації в реальному часі, який має високу швидкість відгуку і наглядну візуалізацію процесу комунікації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Loreto S., Romano S. P. (2014). Real-Time Communication with WebRTC: Peer-to-Peer in the Browser. "O'Reilly Media, Inc.".
2. Comer D. (2013). Internetworking with TCP/IP Volume One.
3. Cadenhead T. (2015). Socket.IO Cookbook.
4. Banks, A., & Porcello, E. (2017). Learning React: functional web development with React and Redux. "O'Reilly Media, Inc.".
5. Banks A., Porcello E. (2020). Learning React: Modern Patterns for Developing React Apps.
6. Duckett, J. (2011). HTML & CSS: design and build websites.
7. Haverbeke M. (2018) .Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming. No Starch Press.
8. Duckett J. (2014). JavaScript and jQuery: Interactive Front-End Web Development. Wiley.
9. Boduch, A. (2019). React Material-UI Cookbook: Build captivating user experiences using React and Material-UI. Packt Publishing Ltd.
10. Richardson L., Amundsen M., Ruby S. (2013). RESTful Web APIs: Services for a Changing World "O'Reilly Media, Inc.".
11. Doglio, F. (2018). Rest Api development with node.js: Manage and understand the full capabilities of successful rest development. Apress.
12. Powers S. (2016). Learning Node "O'Reilly Media, Inc.".
13. Teixeira P. (2012). Professional Node.js: Building Javascript Based Scalable Software.
14. Hahn E. (2016). Express in Action: Writing, building, and testing Node.js applications.
15. Hunter T. II (2020). Distributed Systems with Node.js: Building Enterprise-Ready Backend Services "O'Reilly Media, Inc.".

16. Lombardi A. (2015). WebSocket: Lightweight Client-Server Communications "O'Reilly Media, Inc.".

17. Burns B. (2018). Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. "O'Reilly Media, Inc.".

18. References Fowler, S. J. (2017). Production-ready Microservices : building standardized systems across an engineering organization. "O’reilly Media, Inc.".

19. Newman S. (2015). Building Microservices: Designing Fine-Grained Systems. "O'Reilly Media, Inc.".

20. Gamma E., Helm R., Johnson R., Vlissides J., Booch G., (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

21. Westerveld D. (2021). API Testing and Development with Postman: A practical guide to creating, testing, and managing APIs for automated software testing.

22. Beck K. (2002). Test Driven Development: By Example. Addison-Wesley Professional.

23. Tvoroshenko, I., & Andrieieva, A. (2021). Development of web applications for remote learning of English.

24. Biehl M. (2016). RESTful API Design. CreateSpace Independent Publishing Platform.

25. Grigorik I. (2013). High Performance Browser Networking. "O'Reilly Media, Inc.".

26. Sergiienko A. (2015). WebRTC Cookbook. Packt Publishing

27. Grigorik I. (2013). High Performance Browser Networking. "O'Reilly Media, Inc.".

28. Загальні відомості про Visual Studio Code URL: https://en.wikipedia.org/wiki/Visual_Studio_Code (дата звернення 15.05.2022).

29. Загальні відомості про Real-time communication URL: https://en.wikipedia.org/wiki/Real-time_communication (дата звернення 13.05.2022).

30. Загальні відомості про WebRTC URL: https://webrtc.org/ (дата звернення 14 .05.2022)