Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка програмного забезпечення управління відеоконференцією»

Студента 2 курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення», спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Васильцова Дмитра Дмитровича

Науковий керівник кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Цензура Микола Олександрович

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Криворучко Олена Володимирівна

КИЇВ – 2020

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Васильцову Дмитру Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка програмного забезпечення управління відеоконференцією»

Затверджена наказом ректора від "13" грудня 2019 р. № 4304

2. Строк здачі студентом закінченої проекту 01 грудня 2020

3. Цільова установка та вихідні дані до проекту

Мета проекту полягає у створенні програмного продукту для проведення відеоконференцій.

Об'єкт дослідження проведення відеоконференцій

Предмет дослідження є методи та алгоритми створення програмного продукту для забезпечення відеоконференцій.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОВЕДЕННЯ ВІДЕОКОНФЕРЕНЦІЙ

1.1. ВИЗНАЧЕННЯ ВІДЕОКОНФЕРЕНЦІЇ

1.2. СТАНДАРТИ СТИСНЕННЯ

1.3. ПРОБЛЕМАТИКА ПРОВЕДЕННЯ ВІДЕОКОНФЕРЕЦІЙ

1.4. ВИДИ ВІДЕОКОНФЕРЕНЦІЙ

1.5.ВИСНОВКИ ДО РОЗДІЛУ

РОЗДІЛ 2

ВИБІР ПРОГРАМНИХ ЗАСОБІВ

2.1. МОВА ПРОГРАМУВАННЯ JAVASCRIPT

2.2. NODE JS

2.3. БІБЛІОТЕКА REACT

2.4. БІБЛІОТЕКА SOCKET.IO

2.4. ФРЕЙМВОРК EXPRESS

2.5. ФРЕЙМВОРК ELECTRON

2.6. ВИСНОВКИ ДО РОЗДІЛУ

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ПРОВЕДЕННЯ ВІДЕОКОНФЕРЕНЦІЙ

3.1. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

3.2. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

3.3. ВИСНОВКИ ДО РОЗДІЛУ

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ ДО РОЗРОБКИ ДОДАТКУ

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Дослідження діяльності керування проектами*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Аналіз інструментів розробки*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Програмна реалізація системи*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*08.12.2020-09.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Цензура М.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О.В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Васильцов Д.Д.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту

_____

(підпис, дата)

Відмітка про попередній захист _____Цензура М.О._____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента _____Васильцова Д.Д._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Криворучко О. В._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження проект призначений аналізу методів та засобів проведення відеоконференції та розробці настільного додатку за допомогою засобів веб-розробки.

В результаті аналізу визначено вагомі переваги та недоліки деяких методів управління відеоконференціями, на основі на аналізу обрано найбільш ефективні методи.

Для розробки клієнтської частини застосовано фреймворк Electron та бібліотека для розробки інтерфейсів React.

Ключові слова: Electron, розробка настільного додатку за допомогою засобів веб-розробки, відеоконференція.

ABSTRACT

In accordance with the purpose of the study, the project is designed to analyze the methods and means of video conferencing and the development of a desktop application using web development tools.

As a result of the analysis the significant advantages and disadvantages of some methods of videoconferencing management are determined, on the basis of the analysis the most effective methods are chosen.

The Electron framework and the React interface development library were used to develop the client part.

Keywords: Electron, desktop application development using web development tools, video conferencing .

ВСТУП

Проведення відеоконференцій сьогодні є одним з ефективних засобів дистанційного навчання, спілкування, максимально наближеного за своїми параметрами до реального, яке використовується в самих різних областях діяльності. Дистанційне обговорення актуальних питань фахівцями в режимі реального часу в умовах швидкого розвитку теле-, радіо-, супутникового зв'язку стає більш доступним. В галузі освіти відеоконференція стає звичайним, все частіше використовуються засобом при перепідготовці та підвищенні кваліфікації фахівців, для профільної школи, для навчання людей з обмеженими можливостями здоров'я, самоосвіті. Підготовка та організація відеоконференції залежить від мети її проведення, аудиторії, для якої вона призначена. Якщо в інтерактивній конференції беруть участь вчителі, викладачі вузів може бути запропонований один сценарій, якщо ж на зв'язку учні старших класів, відповідно - інший. Участь у відеоконференції видатних учених, методистів сприяє залученню широкої уваги вчителів, працівників освіти, викладачів вузів з регіонів, які користуються їхніми розробками і підручниками. Спілкування з професіоналами, авторитетними фахівцями дуже дорога для вчителя з регіону, що не має можливості сьогодні приїхати на курси підвищення, але слід підкреслити, що процес цікавий для обох сторін. У сучасному світі з розвитком інформаційних технологій поняття

"провінції" втрачає свій сенс. Вчителі та методисти можуть запропонувати на відеоконференцію свої власні розробки і обмінятися думками з колегами. Співпраця науки і практики, активна взаємодія необхідно, актуально на сучасному етапі, взаємно збагачує. Постійна участь в мережевих заходах: відеоконференціях, форумах, чат-сесіях, електронних журналах, - створює умови для професійного зростання і самоосвіти вчителів, сприяє самореалізації.

Актуальність Важко переоцінити актуальність проведення відеоконференцій під час пандемії. В даний час відеоконференція використовується не тільки для забезпечення віддаленого взаємодії співробітників компаній, але і для вирішення маси інших завдань, в тому числі дистанційного навчання (проведення відеолекцій, семінарів та консультацій) і телемедицини.

Об'єктом дослідження є проведення відеоконференцій.

Предметом дослідження є методи та алгоритми створення програмного продукту для забезпечення відеоконференцій.

Мета роботи полягає у створенні програмного продукту для проведення відеоконференцій.

Результатом роботи є створення програмного продукту – для проведення відеоконференцій.

Завдання дослідження:

- Визначити вимоги та рекомендації до виконання проекту;
- Проаналізувати методи та засоби управління вілеоконференцією;
- Розглянути готові програмні рішення управління відеоконференцією;

- Спроектувати та розробити серверне забезпечення;
- Спроектувати та розробити клієнтську частину.

РОЗДІЛ 1
Дослідження проведення відеоконференцій

1.1. Визначення відеоконференції

Відеоконференція - це технологія, яка дозволяє людям бачити і чути один одного, обмінюватися даними і спільно обробляти їх в інтерактивному режимі, використовуючи можливості звичного всім комп'ютера, максимально наближаючи спілкування на відстані до реального живого спілкування. Області застосування відеоконференції величезні. На сьогоднішній день практично не залишилося області життєдіяльності, в якій не використовують відеоконференцзв'язок.

Відеоконференція знаходить застосування всюди, де необхідні оперативність в аналізі ситуації і ухваленні рішень, консультація фахівця або спільна робота в режимі віддаленого доступу над проектами і рішеннями і т.д. Розглянемо сфери, в яких відеоконференції проводяться найчастіше.

Після з'єднання Ви бачите свого співрозмовника на екрані комп'ютера так, як якщо б він сидів в 3-4 метрах від Вас. Не встаючи з місця, Ви можете вести нормальну живу бесіду, навіть якщо аппонент знаходиться на відстані тисяч кілометрів від Вас. Кількість учасників розмови відеоконференції може бути два або більше (відеоконференцзв'язок). Це дає можливість проводити відеонаради декількох учасників, які перебувають в різних містах, країнах і навіть на різних континентах

1.2. Стандарти стиснення

Використання сучасних рішень в області високих технологій дозволяє розвиватися одному з найперспективніших напрямків міжособистісних комунікацій – відеоконференції. Останнім часом спостерігається справжній переворот в даній області, пов'язаний з появою нового стандарту стиснення - H.265, який став гідною заміною кодеку H.264 при потокової передачі відео, а з його впровадженням вимоги до пропускної здатності каналів передачі даних знижується практично вдвічі. Крім того, на даний момент для відеозв'язку характерно інтеграція соціальних сервісів і хмарних рішень.

1.3. Проблематика проведення відеоконфереції

В даний час області застосування відеотехнологій можуть бути розділені на чотири групи. Впершу входять додатки і сервіси, що забезпечують інтерактивну взаємодію віддалених користувачів в режимі реального часу. До цієї групи належать відеозв'язок і різні його похідні, включаючи багатопоточну взаємодію, тобто відео конференції з кількістю учасників більше трьох. При функціонуванні таких додатків відео по мережі одночасно передається в обидва боки, при цьому висуваються жорсткі вимоги до затримки трафіку і її варіації. Для відеозв'язку високої чіткості необхідні канали з шириною смуги пропускання в кілька десятків Мбіт / с, а для транскодування відео - потужні обчислювальні платформи.

Додатки трьох груп, є менш вимогливі до характеристик передачі трафіку, так як в них передача відеоданих здійснюється тільки в одну сторону. До даних трьох груп відносяться відеоспостереження, різні категорії IP-TV та інформаційні дисплеї, в яких відображення інформації

здійснюється з електронних цифрових носіїв. Як правило, подібні інформаційні дисплеї утворюють територіально розподілену систему і встановлюються в громадських місцях. Ключове перевага таких систем - їх виняткова універсальність і гнучкість, що дозволяє демонструвати на дисплеях найрізноманітнішу інформацію.

В даний час відеоконференція використовується не тільки для забезпечення віддаленого взаємодії співробітників компаній, але і для вирішення маси інших завдань, в тому числі дистанційного навчання (проведення відеолекцій, семінарів та консультацій) і телемедицини.

Головною технічною проблемою впровадження відеоконференцій є характеристики каналів зв'язку. Рішенням даної проблеми може служити виділення вищого пріоритету трафіку відеоданих при їх передачі. Виробники систем відеоконференцій намагаються знизити ефективну ширину смуги пропускання, наприклад, впровадженням технології H.264 High Profile.

На другому місці в ряду технічних складнощів - проблеми інтеграції відеоконференції з іншими корпоративними системами (телефонією, бізнес-додатками і ін.). Відеоконференція довгий час існувала сама по собі, однак цінність цього виду комунікацій істотно підвищується при його включенні в загальну середу уніфікованих (об'єднаних) комунікацій або систему підтримки спільної роботи. Серед пріоритетів в частині інтеграції - забезпечення взаємодії з телефонною системою і програмними додатками

. Наприклад, взаємодія систем відеоконференцій і відеоспостережень дозволить в контролювати процес відвантаження важливого товару або проведення відновлювальні роботи на місці аварії.

Крім того, в даний час в системах відеоконференцій реалізується інтеграція з мобільними пристроями, що викликано зростанням продуктивності і збільшенням енергоефективності мобільних процесорів. Провідні постачальники систем відеоконференцій вже реалізували в своїх рішеннях функції з підтримки основних мобільних платформ (Android, iOS, Windows). При цьому за наявності підключення через Wi-Fi якість відеозв'язку відповідає рекомендаціям MCE. У той же час, можливості мобільного широкосмугового доступу обмежуються, перш за все, структурою і параметрами мереж стільникового зв'язку третього і четвертого покоління, тому що дані мережі не призначені для роботи двобічної відеозв'язку в реальному часі.

Більшість компаній використовують класичну схему ВКС, що включає використання апаратних терміналів і сервера MCU. До таких систем відносяться, перш за все, апаратні комплекси підприємств Polycom, Cisco / Tandberg, LifeSize, Sony, Radvision, VCON. Серед виробників програмних рішень можна відзначити таку компанію, як TrueConf. Найбільший розробник корпоративних та індивідуальних продуктів і обладнання для відеоконференцій в Східній Європі. Рішення TrueConf дозволяють за 15 хвилин розгорнути захищену корпоративну систему об'єднаних комунікації з підтримкою відеоконференції UltraHD якості в масштабах організації будь-якого розміру.

Головним стримуючим фактором у поширенні хмарних сервісів відеоконференцій вважається можливість несанкціонованого доступу до інформації. Недовіра до постачальників хмарних сервісів, а також відсутність пропозицій з прийнятним співвідношенням ціна / функціональність. Серед інших перешкод можна назвати високу вартість каналів зв'язку, потенційні проблеми з оперативним відновленням сеансів

ВКС, а також наявність спеціальних вимог щодо захисту інформації в державних структур.

1.4. Види відеоконференцій

Групові відеоконференції діляться, на симетричні і асиметричні. Відеоконференція є симетричною, якщо всі її учасники можуть повністю взаємодіяти між собою: вони бачать, чують один одного, а також мають можливість спілкуватися. Відеоконференції, в яких відсутній зворотний зв'язок між будь-якими учасниками, називаються асиметричними.

Існує три види асиметричних відеоконференцій: селекторні наради, відеомовлення (відеосемінари) і відеовещанія (трансляція).

Існує два основних типи відеоконференцій - персональна і групова. Персональна відеоконференція передбачає сеанс відеозв'язку, в якому бере участь всього дві особи. Під груповими відеоконференціями маються на увазі всі інші види відеоконференцій.

1.5. Висновки до розділу

Досліджено проблематику проведення відеоконференцій, а також деякі технічні питання проекту.

Проаналізовано відомі методології проведення відеоконференцій, та готові технічні рішення.

Складено технічне завдання по розробці програмного рішення даного проекту, вказані важливі аспекти щодо структури серверної і клієнтської частини.

РОЗДІЛ 2 Вибір програмних засобів

2.1. Мова програмування JavaScript

JavaScript - мультіпарадігменна мова програмування. Підтримує об'єктно-орієнтована, імперативний і функціональний стилі. Є реалізацією стандарту ECMAScript (стандарт ECMA-262.)

JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожим на Java. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє його від ряду мов програмування, використовуваних в веб-розробці.

JavaScript це мова, яка дозволяє Вам застосовувати складні речі на web сторінці - кожен раз, коли на web сторінці відбувається щось більше, ніж просто її статичне відображення - відображення періодично

оновлюваного контенту, або інтерактивних карт, або анімація 2D / 3D графіки, або прокрутка відео в програвачі, і т.д. - можете бути впевнені, що швидше за все, не обійшлося без JavaScript. Це третій шар листкового пирога стандартних web технологій, два з яких (HTML і CSS)

2.2. Node js

Node або Node.js - програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованого мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js і espruino). В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням / висновком.

2.3. Бібліотека react

React - це бібліотека JavaScript, яка використовується для створення призначеного для користувача інтерфейсу. React був створений компанією Facebook, а перший реліз бібліотеки побачив світ у березні 2013 року. Поточної версій на даний момент (жовтень 2020 року) є версія React v17.0.1.

Спочатку React призначався для створення веб-сайтів, проте пізніше з'явилася платформа React Native, яка вже призначалася для мобільних пристроїв.

React представляється ідеальний інструмент для створення масштабованих веб-додатків (в даному випадку мова йде про фронтенді), особливо в тих ситуаціях, коли додаток являє SPA (односторінкове додаток).

React відносно простий в освоєнні, має зрозумілий та лаконічний синтаксис.

Віртуальний DOM

Вся структура веб-сторінки може бути представлена за допомогою DOM (Document Object Model) - організація елементів html, якими ми можемо маніпулювати, змінювати, видаляти або додавати нові. Для взаємодії з DOM застосовується мова JavaScript. Однак коли ми намагаємося маніпулювати html-елементами за допомогою JavaScript, то ми можемо зіткнутися зі зниженням продуктивності, особливо при зміні великої кількості елементів. А операції над елементами можуть зайняти деякий час, що неминуче позначиться на призначеному для користувача досвід. Однак якби ми працювали з коду js з об'єктами JavaScript, то операції проводилися б швидше.

Для вирішення проблеми продуктивності якраз і з'явилася концепція віртуального DOM.

Віртуальний DOM представляє легковажну копію звичайного DOM. І відмінною рисою React є те, що дана бібліотека працює саме з віртуальним DOM, а не звичайним.

Якщо з додатком потрібно дізнатися інформацію про стан елементів, то відбувається звернення до віртуального DOM.

Якщо необхідно змінити елементи веб-сторінки, то зміни спочатку вносяться в віртуальний DOM. Потім новий стан віртуального DOM порівнюється з поточним станом. І якщо ці стани розрізняються, то React знаходить мінімальну кількість маніпуляцій, які необхідні до поновлення реального DOM до нового стану і виробляє їх.

У підсумку така схема взаємодії з елементами веб-сторінки працює набагато швидше і ефективніше, ніж якби ми працювали з JavaScript з DOM безпосередньо.

2.4. Бібліотека Socket.IO

Socket.IO - JavaScript-бібліотека для веб-додатків і обміну даними в реальному часі. Складається з двох частин: клієнтської, яка запускається в браузері і серверної для node.js. Обидва компоненти мають схожу API. Подібно node.js, Socket.IO подієво-орієнтована.

Socket.IO головним чином використовує протокол WebSocket, але якщо потрібно, використовує інші технології, наприклад Flash Socket, AJAX Long Polling, AJAX Multipart Stream, надаючи той же самий інтерфейс. Крім того, що Socket.IO може бути використана як оболонка для WebSocket, вона містить багато інших функцій, включаючи мовлення на кілька гнізд, зберігання даних, пов'язаних з кожним клієнтом, і асинхронний ввід / вивід.

Може бути встановлена через npm (node package manager).

2.4. Фреймворк Express

Express.js, або просто Express, фреймворк web-додатків для Node.js, реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроектований для створення веб-додатків і API. Де-факто є стандартним каркасом для Node.js. Автор фреймворка, TJ Holowaychuk, описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи на увазі, що він мінімалістичний і включає велику кількість додаткових плагінів. Express може бути backend'ом для програмного стека MEAN, разом з базою даних MongoDB і каркасом Vue.js, React або AngularJS для frontend'a.

2.5. Фреймворк Electron

Electron - це фреймворк для розробки настільних додатків з впровадженням HTML, CSS і JavaScript. Такі додатки можуть працювати на різних платформах. Серед них - Windows, Mac і Linux.

В основі Electron лежать проекти Chromium і Node.js, Об'єднаний в єдине середовище, що забезпечує роботу додатків. Це дає можливість застосовувати веб-технології при розробці настільних програм.

Розробка настільних додатків з графічним інтерфейсом користувача трудової роботи і потребує спеціальних знань щодо операційної системи (ОС), що приводить до додаткових затрат і тимчасових затрат, а також до збільшення технічної складності проекту. Для таких проектів забезпечення якості, функціональності та безпеки в декількох операційних системах є важкою задачею. Electron пропонує вирішення цієї проблеми, надаючи основу для розробки кроссплатформенних нативних настільних додатків із використанням

веб-технологій, таких як JavaScript, HTML та CSS. Це означає, що досвідчений веб-розробник може розширювати такі безкоштовні додатки, не володіючи загальними навичками роботи з ОС.

Підготовка, створена за допомогою Electron, складається з двох типових процесів: основного процесу та декількох процесорів рендеринга, які наступні шаблони головного-підчиненого. Основним процесом додатків є головний процес, кілька процесів рендеринга - підчинені, які обмінюються даними за допомогою міжпроцесорного взаємодії (МПК)

Зокрема, Electron вивчає початковий запис у маніфесті package.json, включеному до проекту, щоб визначити точку входу програми, яка працює як основний процес.

Основний процес: Основний процес відповідає за реагування на події життєвого циклу програми, такі як запуск, вихід із програми, підготовка до виходу, перехід у фоновий режим, вихід на передній план тощо. Основний процес також відповідає за зв'язок із власними API операційної системи. Наприклад, для відображення діалогового вікна для відкриття або збереження файлу. Основний процес також може створювати та знищувати процеси візуалізації за допомогою модуля BrowserWindow від Electron.

Процеси візуалізації: процеси візуалізації можуть завантажувати веб-сторінки для відображення графічного інтерфейсу. Кожен процес використовує багатопроцесорну архітектуру Chromium і працює на своєму потоці. На відміну від звичайних веб-сторінок, у процесах візуалізації є доступ до всіх API Node, що дозволяє розробникам використовувати рідні модулі та системні взаємодії нижчого рівня.

2.6. Висновки до розділу

В даному розділі було розглянуто обрані програмні засоби, їх переваги та недоліки.

Проаналізовано архітектуру і роботу фреймворка Electron, що надає можливість використовувати веб-інструменти для реалізації клієнт частини.

РОЗДІЛ 3 Реалізація програмного продукту для проведення відеоконференцій

3.1. Розробка серверної частини

Для розробки серверної частини використовується мова програмування JavaScript, платформа Node і фреймворк Express. Головним завданням є створити швидке забезпечення для отримання і обробки запитів з клієнта.

Для ініціалізації проекту потрібно завантажити і встановити платформу Node з офіційного сайту. Наступним етапом має бути встановити всі необхідні бібліотеки і фреймворк Express за допомогою пакетного менеджера npm, який встановлюється разом з Node.

Установка бібліотек зазвичай починається з допоміжних бібліотек для роботи над проектом, а саме:

- eslint - для підсвічування синтаксису;
- prettier - для форматування коду;
- nodemon - для безперервної роботи з сервером на стадії розробки;
- cross-env - для виконання npm скриптів в будь-якому середовищі розробки.

З основних бібліотек:

- express - простий і гнучкий фреймворк для розробки веб та мобільних додатків, або ж звичайних API-серверів;
- socket.io - JavaScript-бібліотека для веб-додатків і обміну даними в реальному часі;

- xss – бібліотека для уникнення XSS-атаки.

Всі встановлені бібліотеки додаються в файл package.json на рис. 3.1. (Файл package.json – список встановлених бібліотек), який містить інформацію про проект, скрипти виконання і два розділи під бібліотеки.

package.json

C: > Users > crow9 > Desktop > project > {} package.json > ...

```
{
  "name": "video",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@material-ui/core": "^4.9.7",
    "@material-ui/icons": "^4.9.1",
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.3.2",
    "@testing-library/user-event": "^7.1.2",
    "antd": "^4.1.4",
    "bootstrap": "^4.4.1",
    "concurrently": "^5.1.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "faker": "^4.1.0",
    "nodemon": "^2.0.2",
    "react": "^16.13.1",
    "react-bootstrap": "^1.0.0",
    "react-css-grid": "^2.0.0-0",
    "react-dom": "^16.13.1",
    "react-flexbox-grid": "^2.1.2",
    "react-grid-layout": "^0.18.3",
    "react-grid-system": "^6.3.0",
    "react-native": "^0.62.1",
    "react-router": "^5.1.2",
    "react-router-dom": "^5.1.2",
    "react-scripts": "3.4.1",
    "reactstrap": "^8.4.1",
    "socket.io": "^2.3.0",
    "socket.io-client": "^2.3.0",
    "standard": "^14.3.3",
    "xss": "^1.0.8"
  },
```

Рис.3.1. Файл package.json – список встановлених бібліотек

Застосовуючи встановлені бібліотеки створюється вхідний файл в якому відбувається ініціалізації серверу за допомогою фреймворка express та бібліотеки HTTP.

Для ініціалізації прямого підключення між сервером і клієнтом використовується бібліотека socket.io.

Головний функціонал сервера має такі фунції: (див. додаток А)

- Підключення до конференції рис. 3.2.(запит на підключення до конференції);
- Чат рис. 3.3.(функціонал чату);
- Дісконект рис.3.4.(запит на відключення від конференції) .

```
socket.on('join-call', (path) => {
    if(connections[path] === undefined){
        connections[path] = []
    }
    connections[path].push(socket.id)

    timeOnline[socket.id] = new Date()

    for(let a = 0; a < connections[path].length; ++a){
        io.to(connections[path][a]).emit("user-joined", socket.id, connections[path])
    }

    if(messages[path] !== undefined){
        for(let a = 0; a < messages[path].length; ++a){
            io.to(socket.id).emit("chat-message", messages[path][a]['data'],
                messages[path][a]['sender'], messages[path][a]['socket-id-sender'])
        }
    }
```

Рис. 3.2. Запит на підключення до конференції

```
socket.on('chat-message', (data, sender) => {
    data = sanitizeString(data)
    sender = sanitizeString(sender)

    var key
    var ok = false
    for (const [k, v] of Object.entries(connections)) {
        for(let a = 0; a < v.length; ++a){
            if(v[a] === socket.id){
                key = k
                ok = true
            }
        }
    }

    if(ok === true){
        if(messages[key] === undefined){
            messages[key] = []
        }
        messages[key].push({"sender": sender, "data": data, "socket-id-sender": socket.id})


        for(let a = 0; a < connections[key].length; ++a){
            io.to(connections[key][a]).emit("chat-message", data, sender, socket.id)
        }
    }
```

Рис. 3.3. Функціонал чату

```
socket.on('disconnect', () => {
    var diffTime = Math.abs(timeOnline[socket.id] - new Date())
    var key
    for (const [k, v] of JSON.parse(JSON.stringify(Object.entries(connections)))
        for(let a = 0; a < v.length; ++a){
            if(v[a] === socket.id){
                key = k

                for(let a = 0; a < connections[key].length; ++a){
                    io.to(connections[key][a]).emit("user-left", socket.id)
                }

                var index = connections[key].indexOf(socket.id)
                connections[key].splice(index, 1)


                if(connections[key].length === 0){
                    delete connections[key]
                }
```

Рис.3.4. Запит на відключення від конференції

3.2 Розробка клієнтської частини

Для початку роботи над додатком необхідно встановити фреймворк Electron. Після установки даного фреймворка створюється вхідний файл «app.js», який містить надбудови для запуска веб-додатків у вигляді настільних додатків. Цього достатньо для роботи настільного додатку в середовищі розробки.

Наступним етапом є розробка інтерфейсу за допомогою бібліотеки React. Для початку необхідно створити вхідний файл ініціалізації додатку рис. 3.5. (Вхідний файл «index.js»), який містить розбиту на компоненти реалізацію всього інтерфейсу.

src > JS index.js

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import App from './App';
4   import * as serviceWorker from './serviceWorker';
5
6   ReactDOM.render(
7       <App />,
8       document.getElementById('root')
9   );
```

Рис. 3.5. Вхідний файл «index.js»

Для поділу логіки та інтерфейсу використовується методика поділу компонентів на:

- компоненти-уявлення – які містять тільки структуру та дизайн компонента;
- компоненти-контейнери – містять в собі компоненти-уявлення і будь-які логічні взаємодії з даними, наприклад, створення запитів до серверу.

Після ініціалізації програми можна перейти до проектування та розробки сторінки проекту рис.3.6. (Інтерфейс сторінки проекту), яка

надає можливість створення власної конференції чи підключення до існуючої за допомогою URL.

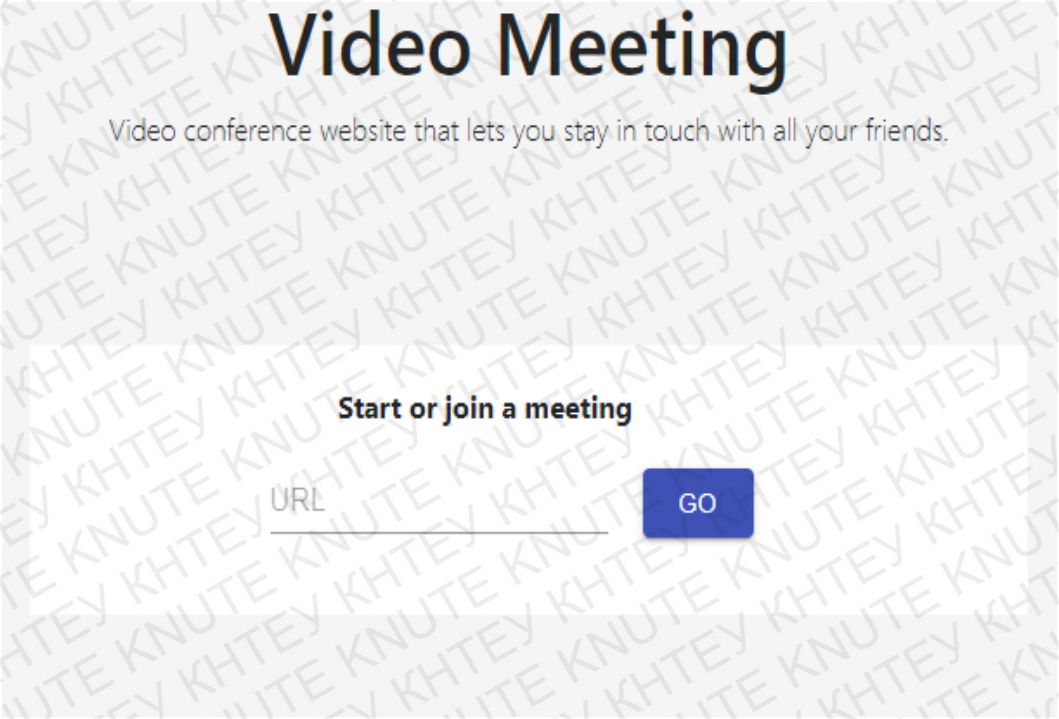

Рис.3.6. Інтерфейс сторінки проекту

Наступний крок проектування та розробка сторінки яка відповідає за підключення до готової відео конференції рис.3.7. (Інтерфейс підключення до відеоконференції), яка дозволяє вибрати ім'я яке буде відображатися під час відеоконференції та містить зображення веб камери.

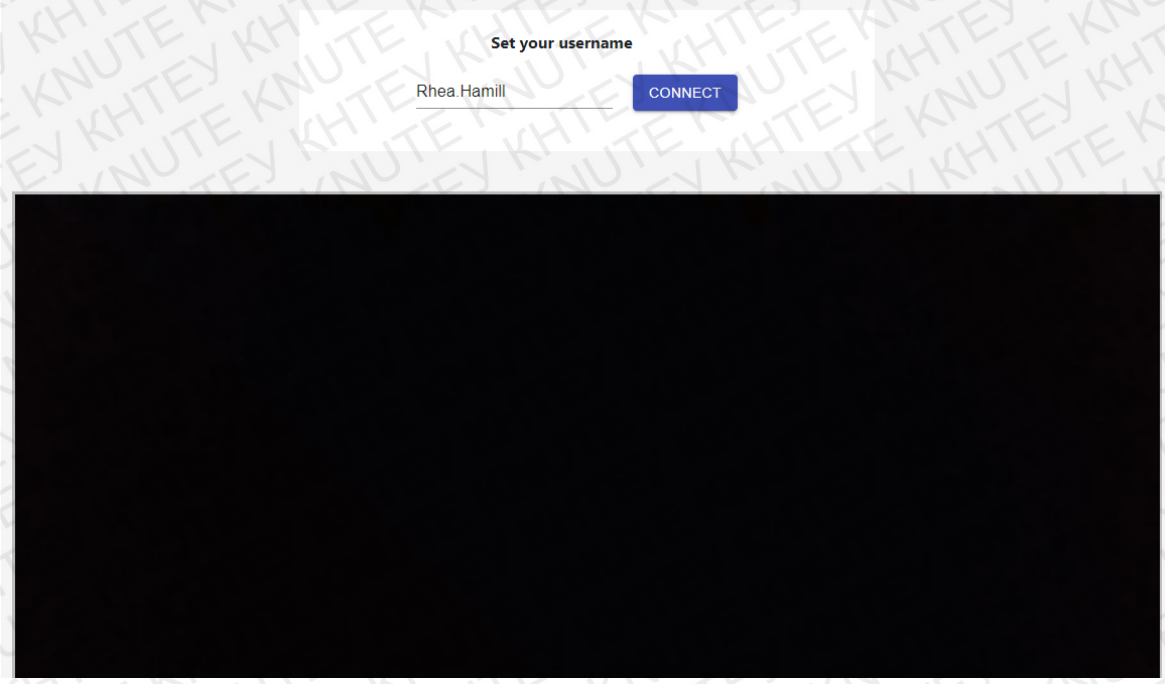

Рис.3.7. Інтерфейс підключення до відеоконференції

Останній крок проектування та розробка сторінки яка відповідає за трансляцію відео, звуку, можливість чату та демонстрацію екрану рис.3.8. (Інтерфейс під час проведення відуоконференції). (див. додаток Б)

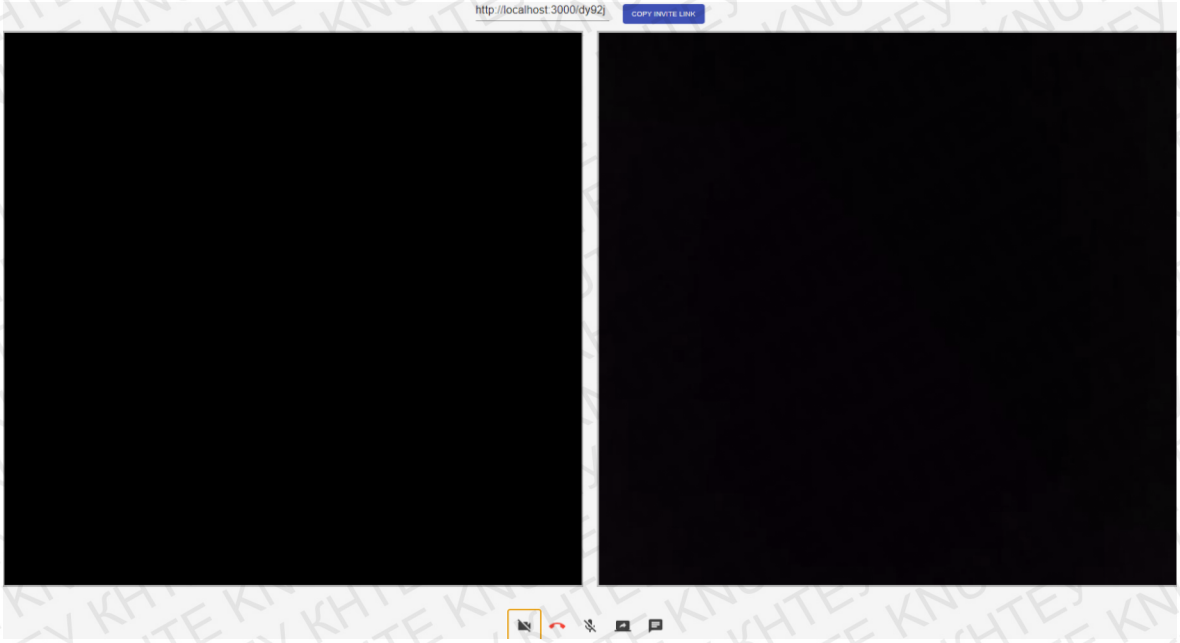

Рис.3.8. Інтерфейс під час проведення відуоконференції

3.3. Висновки до розділу

В даному розділі було спроектовано і розроблено клієнтську і серверну частину настільного додатку за допомогою сучасних засобів розробки, таких як: Electron, React, NodeJS, Express.

Клієнт частина розроблена як звичайний веб-додаток за допомогою бібліотеки React, застосовуючи всі ті ж підходи та інструменти, що і в веб-розробці. Настільні додатки є результатом впровадження фреймворка Electron, який має можливість відкривати веб-додатки всередині вікна поточної операційної системи.

Electron, який має можливість відкривати веб-додатки всередині вікна поточної операційної системи.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В ході виконання випускного кваліфікаційного проекту було розглянуто проектування системи управління відеоконференцією. Було проаналізовано метода та засоби управління відеоконференцією.

Проаналізовано засоби розробки настільних додатків за допомогою інструментів для веб-розробки. Розглянуто фреймворк Electron, який надає можливості впровадження веб-засобів до настільних додатків.

Спроектовано та розроблено структуру настільного додатку дотримуючись принципів простого і зрозумілого інтерфейсу. Клієнтську частину розроблено, як багатосторінковий додаток за допомогою бібліотеки для розробки інтерфейсів React.

Розроблене серверне забезпечення використовує пряме підключення між сервером і клієнтом, та дозволяє транслювати відео в режимі реального часу. Планується подальший розвиток проекту оснований на відгуках і потребах користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Видання іноземною мовою:

1. Simpson K. You Don't Know JS: Async & Performance / Kyle Simpson. – O'Reilly Media, Inc, 2015 – 247 p.
2. Brown E. Web Development with Node and Express, 2nd Edition / Ethan Brown – O'Reilly Media, 2019 – 409 p.

Електронні ресурси:

3. Документація бібліотеки для розробки інтерфейсів React [Електронний ресурс]. – [Режим доступу]: https://reactjs.org/docs.
4. Документація фреймворку Electron [Електронний ресурс]. – [Режим доступу]: https://www.electronjs.org/docs.
5. Документація Node.js [Електронний ресурс]. – [Режим доступу]: https://nodejs.org/uk/docs/

ТЕХНІЧНЕ ЗАВДАННЯ ДО РОЗРОБКИ ДОДАТКУ

1. Загальні вимоги

Необхідно створити простий та зрозумілий за інтерфейсом настільний додаток, який надає можливості керування проектами інформаційних систем.

1.1 Вимоги до інструментів розробки

1.1.1. Інструменти для розробки серверної частини

- мова програмування JavaScript;
- програмна платформа Node та фреймворк Express;

1.1.2. Інструменти для розробки клієнтської частини

- мова програмування JavaScript;
- бібліотека для розробки інтерфейсів React;
- біблітека стилізації styled-components;
- біблітека socket.io
- webpack – інструмент для складання модульних файлів JavaScript;
- babel – компілятор сучасного JavaScript-коду для підтримки старих браузерів.

2. Вимоги до розробки серверного забезпечення.

2.1. Вимоги до структури.

Серверне забезпечення має дотримуватися послідовної структури, а саме:

- controllers – функції які використовуються в роутері;
- routes – функції для обробки REST-запитів.

3. Вимоги до розробки клієнтської частини.

3.1. Вимоги до оформлення сторінок.

Кожна сторінка, включає в себе заголовок, опис і найголовніше певний функціонал.

3.2. Вимоги до оформлення інтерфейсу.

Візуалізація повинна забезпечувати зручний для користувача інтерфейс, який відповідає наступним вимогам:

- наявність локалізованого інтерфейсу користувача;
- меню навігації.

ДОДАТКИ

Додаток А

Код файлу створення серверу

```
const express = require('express')
const http = require('http')
var cors = require('cors')
const app = express()
const bodyParser = require('body-parser')
const path = require("path")
var xss = require("xss")

var server = http.createServer(app)
var io = require('socket.io')(server)

app.use(cors())
app.use(bodyParser.json())

if(process.env.NODE_ENV==='production'){
    app.use(express.static(__dirname+"/build"))
    app.get("*", (req, res) => {
        res.sendFile(path.join(__dirname+"/build/index.html"))
    })
}
app.set('port', (process.env.PORT || 4001))

sanitizeString = (str) => {
    return xss(str)
}

connections = {}
messages = {}
timeOnline = {}
```

Додаток Б

Код інтерфейс головної сторінки

```
import React, { Component } from 'react'
import io from 'socket.io-client'
import faker from "faker"

import {IconButton, Badge, Input, Button} from '@material-ui/core'
import VideocamIcon from '@material-ui/icons/Videocam'
import VideocamOffIcon from '@material-ui/icons/VideocamOff'
import MicIcon from '@material-ui/icons/Mic'
import MicOffIcon from '@material-ui/icons/MicOff'
import ScreenShareIcon from '@material-ui/icons/ScreenShare'
import StopScreenShareIcon from '@material-ui/icons/StopScreenShare'
import CallEndIcon from '@material-ui/icons/CallEnd'
import ChatIcon from '@material-ui/icons/Chat'

import { message } from 'antd'
import 'antd/dist/antd.css'

import { Row } from 'reactstrap'
import Modal from 'react-bootstrap/Modal'
import 'bootstrap/dist/css/bootstrap.css'
import "./Video.css"

const server_url = "http://localhost:4001"

var connections = {}
const peerConnectionConfig = {
    'iceServers': [
        { 'urls': 'stun:stun.l.google.com:19302' },
    ]
}
var socket = null
var socketId = null
var elms = 0

class Video extends Component {
    constructor(props) {
        super(props)

        this.localVideoref = React.createRef()

        this.videoAvailable = false
        this.audioAvailable = false

        this.state = {
            video: false,
            audio: false,
            screen: false,
            showModal: false,
            screenAvailable: false,
            messages: [],
            message: "",
            newmessages: 0,
            askForUsername: true,
```

```
      username: faker.internet.userName(),
    }
    connections = {}

    this.getPermissions()
  }

  getPermissions = async () => {
    try{
      await navigator.mediaDevices.getUserMedia({ video: true })
        .then(() => this.videoAvailable = true)
        .catch(() => this.videoAvailable = false)

      await navigator.mediaDevices.getUserMedia({ audio: true })
        .then(() => this.audioAvailable = true)
        .catch(() => this.audioAvailable = false)

      if (navigator.mediaDevices.getDisplayMedia) {
        this.setState({ screenAvailable: true })
      } else {
        this.setState({ screenAvailable: false })
      }

      if (this.videoAvailable || this.audioAvailable) {
        navigator.mediaDevices.getUserMedia({ video: this.videoAvailable, audio: this.audioAvailable
})
          .then((stream) => {
            window.localStream = stream
            this.localVideoref.current.srcObject = stream
          })
          .then((stream) => {})
          .catch((e) => console.log(e))
      }
    } catch(e) { console.log(e) }
  }

  getMedia = () => {
    this.setState({
      video: this.videoAvailable,
      audio: this.audioAvailable
    }, () => {
      this.getUserMedia()
      this.connectToSocketServer()
    })
  }

  getUserMedia = () => {
    if ((this.state.video && this.videoAvailable) || (this.state.audio && this.audioAvailable)) {
      navigator.mediaDevices.getUserMedia({ video: this.state.video, audio: this.state.audio })
        .then(this.getUserMediaSuccess)
        .then((stream) => {})
        .catch((e) => console.log(e))
    } else {
      try {
        let tracks = this.localVideoref.current.srcObject.getTracks()
        tracks.forEach(track => track.stop())
      } catch (e) {}
```

```
    }
  }

  getUserMediaSuccess = (stream) => {
    try {
      window.localStream.getTracks().forEach(track => track.stop())
    } catch(e) { console.log(e) }

    window.localStream = stream
    this.localVideoref.current.srcObject = stream

    for (let id in connections) {
      if (id === socketId) continue

      connections[id].addStream(window.localStream)

      connections[id].createOffer().then((description) => {
        connections[id].setLocalDescription(description)
          .then(() => {
            socket.emit('signal', id, JSON.stringify({ 'sdp': connections[id].localDescription }))
          })
          .catch(e => console.log(e))
      })
    }

    stream.getTracks().forEach(track => track.onended = () => {
      this.setState({
        video: false,
        audio: false,
      }, () => {
        try {
          let tracks = this.localVideoref.current.srcObject.getTracks()
          tracks.forEach(track => track.stop())
        } catch(e) { console.log(e) }

        let blackSilence = (...args) => new MediaStream([this.black(...args), this.silence()])
        window.localStream = blackSilence()
        this.localVideoref.current.srcObject = window.localStream

        for (let id in connections) {
          connections[id].addStream(window.localStream)

          connections[id].createOffer().then((description) => {
            connections[id].setLocalDescription(description)
              .then(() => {
                socket.emit('signal', id, JSON.stringify({ 'sdp': connections[id].localDescription }))
              })
              .catch(e => console.log(e))
          })
        }
      })
    })
  }

  getDislayMedia = () => {
    if (this.state.screen) {
      if (navigator.mediaDevices.getDisplayMedia) {
```

```
            navigator.mediaDevices.getDisplayMedia({ video: true, audio: true })
              .then(this.getDislayMediaSuccess)
              .then((stream) => {})
              .catch((e) => console.log(e))
          }
        }
      }

      getDislayMediaSuccess = (stream) => {
        try {
          window.localStream.getTracks().forEach(track => track.stop())
        } catch(e) { console.log(e) }

        window.localStream = stream
        this.localVideoref.current.srcObject = stream

        for (let id in connections) {
          if (id === socketId) continue

          connections[id].addStream(window.localStream)

          connections[id].createOffer().then((description) => {
            connections[id].setLocalDescription(description)
              .then(() => {
                socket.emit('signal', id, JSON.stringify({ 'sdp': connections[id].localDescription }))
              })
              .catch(e => console.log(e))
          })
        }

        stream.getTracks().forEach(track => track.onended = () => {
          this.setState({
            screen: false,
          }, () => {
            try {
              let tracks = this.localVideoref.current.srcObject.getTracks()
              tracks.forEach(track => track.stop())
            } catch(e) { console.log(e) }

            let blackSilence = (...args) => new MediaStream([this.black(...args), this.silence()])
            window.localStream = blackSilence()
            this.localVideoref.current.srcObject = window.localStream

            this.getUserMedia()
          })
        })
      }

      gotMessageFromServer = (fromId, message) => {
        var signal = JSON.parse(message)

        if (fromId !== socketId) {
          if (signal.sdp) {
            connections[fromId].setRemoteDescription(new RTCSessionDescription(signal.sdp)).then(() =
> {
              if (signal.sdp.type === 'offer') {
                connections[fromId].createAnswer().then((description) => {
```

```
                    connections[fromId].setLocalDescription(description).then(() => {
                        socket.emit('signal', fromId, JSON.stringify({ 'sdp': connections[fromId].localDescription }))
                    }).catch(e => console.log(e))
                }).catch(e => console.log(e))
            }
        }).catch(e => console.log(e))
    }

    if (signal.ice) {
        connections[fromId].addIceCandidate(new RTCIceCandidate(signal.ice)).catch(e => console.log(e))
    }
  }
}

changeCssVideos = (main) => {
  let widthMain = main.offsetWidth
  let minWidth = "30%"
  if ((widthMain * 30 / 100) < 300) {
    minWidth = "300px"
  }
  let minHeight = "40%"

  let height = String(100 / elms) + "%"
  let width = ""
  if(elms === 0 || elms === 1) {
    width = "100%"
    height = "100%"
  } else if (elms === 2) {
    width = "45%"
    height = "100%"
  } else if (elms === 3 || elms === 4) {
    width = "35%"
    height = "50%"
  } else {
    width = String(100 / elms) + "%"
  }

  let videos = main.querySelectorAll("video")
  for (let a = 0; a < videos.length; ++a) {
    videos[a].style.minWidth = minWidth
    videos[a].style.minHeight = minHeight
    videos[a].style.setProperty("width", width)
    videos[a].style.setProperty("height", height)
  }

  return {minWidth, minHeight, width, height}
}

connectToSocketServer = () => {
  socket = io.connect(server_url, { secure: true })

  socket.on('signal', this.gotMessageFromServer)

  socket.on('connect', () => {
    socket.emit('join-call', window.location.href)
```

```
        socketId = socket.id

        socket.on('chat-message', this.addMessage)

        socket.on('user-left', (id) => {
            let video = document.querySelector(`[data-socket="${id}"]`)
            if (video !== null) {
                elms--
                video.parentNode.removeChild(video)

                let main = document.getElementById('main')
                this.changeCssVideos(main)
            }
        })

        socket.on('user-joined', (id, clients) => {
            clients.forEach((socketListId) => {
                connections[socketListId] = new RTCPeerConnection(peerConnectionConfig)
                connections[socketListId].onicecandidate = function (event) {
                    if (event.candidate != null) {
                        socket.emit('signal', socketListId, JSON.stringify({ 'ice': event.candidate }))
                    }
                }

                connections[socketListId].onaddstream = (event) => {
                    var searchVidep = document.querySelector(`[data-socket="${socketListId}"]`)
                    if (searchVidep !== null) {
                        searchVidep.srcObject = event.stream
                    } else {
                        elms = clients.length
                        let main = document.getElementById('main')
                        let cssMesure = this.changeCssVideos(main)

                        let video = document.createElement('video')

                        let css = {minWidth: cssMesure.minWidth, minHeight: cssMesure.minHeight, maxHeig
ht: "100%", margin: "10px",
                            borderStyle: "solid", borderColor: "#bdbdbd", objectFit: "fill"}
                        for(let i in css) video.style[i] = css[i]

                        video.style.setProperty("width", cssMesure.width)
                        video.style.setProperty("height", cssMesure.height)
                        video.setAttribute('data-socket', socketListId)
                        video.srcObject = event.stream
                        video.autoplay = true
                        video.playsinline = true

                        main.appendChild(video)
                    }
                }

                if (window.localStream !== undefined && window.localStream !== null) {
                    connections[socketListId].addStream(window.localStream)
                } else {
                    let blackSilence = (...args) => new MediaStream([this.black(...args), this.silence()])
                    window.localStream = blackSilence()
                    connections[socketListId].addStream(window.localStream)
```

```
            }
          })

          if (id === socketId) {
            for (let id2 in connections) {
              if (id2 === socketId) continue

              try {
                connections[id2].addStream(window.localStream)
              } catch(e) {}

              connections[id2].createOffer().then((description) => {
                connections[id2].setLocalDescription(description)
                  .then(() => {
                    socket.emit('signal', id2, JSON.stringify({ 'sdp': connections[id2].localDescription
}))
                  })
                  .catch(e => console.log(e))
              })
            }
          }
        })
      })
    }

    silence = () => {
      let ctx = new AudioContext()
      let oscillator = ctx.createOscillator()
      let dst = oscillator.connect(ctx.createMediaStreamDestination())
      oscillator.start()
      ctx.resume()
      return Object.assign(dst.stream.getAudioTracks()[0], { enabled: false })
    }
    black = ({ width = 640, height = 480 } = {}) => {
      let canvas = Object.assign(document.createElement("canvas"), { width, height })
      canvas.getContext('2d').fillRect(0, 0, width, height)
      let stream = canvas.captureStream()
      return Object.assign(stream.getVideoTracks()[0], { enabled: false })
    }

    handleVideo = () => this.setState({ video: !this.state.video }, () => this.getUserMedia())
    handleAudio = () => this.setState({ audio: !this.state.audio }, () => this.getUserMedia())
    handleScreen = () => this.setState({ screen: !this.state.screen }, () => this.getDislayMedia())

    handleEndCall = () => {
      try {
        let tracks = this.localVideoref.current.srcObject.getTracks()
        tracks.forEach(track => track.stop())
      } catch (e) {}
      window.location.href = "/"
    }

    openChat = () => this.setState({ showModal: true, newmessages: 0 })
    closeChat = () => this.setState({ showModal: false })
    handleMessage = (e) => this.setState({ message: e.target.value })

    addMessage = (data, sender, socketIdSender) => {
```

```
      this.setState(prevState => ({
        messages: [...prevState.messages, { "sender": sender, "data": data }],
      }))
      if (socketIdSender !== socketId) {
        this.setState({ newmessages: this.state.newmessages + 1 })
      }
    }

    handleUsername = (e) => this.setState({ username: e.target.value })

    sendMessage = () => {
      socket.emit('chat-message', this.state.message, this.state.username)
      this.setState({ message: "", sender: this.state.username })
    }

    copyUrl = () => {
      let text = window.location.href
      if (!navigator.clipboard) {
        let textArea = document.createElement("textarea")
        textArea.value = text
        document.body.appendChild(textArea)
        textArea.focus()
        textArea.select()
        try {
          document.execCommand('copy')
          message.success("Link copied to clipboard!")
        } catch (err) {
          message.error("Failed to copy")
        }
        document.body.removeChild(textArea)
        return
      }
      navigator.clipboard.writeText(text).then(function () {
        message.success("Link copied to clipboard!")
      }, () => {
        message.error("Failed to copy")
      })
    }

    connect = () => this.setState({ askForUsername: false }, () => this.getMedia())

    isChrome = function () {
      let userAgent = (navigator && (navigator.userAgent || "")).toLowerCase()
      let vendor = (navigator && (navigator.vendor || "")).toLowerCase()
      let matchChrome = /google inc/.test(vendor) ? userAgent.match(/(?:chrome|crios)\/(\d+)/) : null
      return matchChrome !== null
    }

    render() {
      if(this.isChrome() === false){
        return (
          <div style={{background: "white", width: "30%", height: "auto", padding: "20px", minWidth: "
400px",
                textAlign: "center", margin: "auto", marginTop: "50px", justifyContent: "center"}}>
            <h1>Sorry, this works only with Google Chrome</h1>
          </div>
        )
```

```
    }
    return (
      <div>
        {this.state.askForUsername === true ?
          <div>
            <div style={{background: "white", width: "30%", height: "auto", padding: "20px", minWi
dth: "400px",
                textAlign: "center", margin: "auto", marginTop: "50px", justifyContent: "center"}}>
              <p style={{ margin: 0, fontWeight: "bold", paddingRight: "50px" }}>Set your username
</p>
              <Input placeholder="Username" value={this.state.username} onChange={e => this.han
dleUsername(e)} />
              <Button variant="contained" color="primary" onClick={this.connect} style={{ margin:
"20px" }}>Connect</Button>
            </div>

            <div style={{ justifyContent: "center", textAlign: "center", paddingTop: "40px" }}>
              <video id="my-video" ref={this.localVideoref} autoPlay muted style={{
                borderStyle: "solid",borderColor: "#bdbdbd",objectFit: "fill",width: "60%",height: "3
0%"}}></video>
            </div>
          </div>
          :
          <div>
            <div className="btn-
down" style={{ backgroundColor: "whitesmoke", color: "whitesmoke", textAlign: "center" }}>
              <IconButton style={{ color: "#424242" }} onClick={this.handleVideo}>
                {(this.state.video === true) ? <VideocamIcon /> : <VideocamOffIcon />}
              </IconButton>

              <IconButton style={{ color: "#f44336" }} onClick={this.handleEndCall}>
                <CallEndIcon />
              </IconButton>

              <IconButton style={{ color: "#424242" }} onClick={this.handleAudio}>
                {this.state.audio === true ? <MicIcon /> : <MicOffIcon />}
              </IconButton>

              {this.state.screenAvailable === true ?
                <IconButton style={{ color: "#424242" }} onClick={this.handleScreen}>
                  {this.state.screen === true ? <ScreenShareIcon /> : <StopScreenShareIcon />}
                </IconButton>
                : null}

              <Badge badgeContent={this.state.newmessages} max={999} color="secondary" onClic
k={this.openChat}>
                <IconButton style={{ color: "#424242" }} onClick={this.openChat}>
                  <ChatIcon />
                </IconButton>
              </Badge>
            </div>

            <Modal show={this.state.showModal} onHide={this.closeChat} style={{ zIndex: "999999
" }}>
              <Modal.Header closeButton>
                <Modal.Title>Chat Room</Modal.Title>
              </Modal.Header>
```

```
                    <Modal.Body style={{ overflow: "auto", overflowY: "auto", height: "400px", textAlign:
 "left" }} >
                      {this.state.messages.length > 0 ? this.state.messages.map((item, index) => (
                        <div key={index} style={{textAlign: "left"}}>
                          <p style={{ wordBreak: "break-all" }}><b>{item.sender}</b>: {item.data}</p>
                        </div>
                      )) : <p>No message yet</p>}
                    </Modal.Body>
                    <Modal.Footer className="div-send-msg">
                      <Input placeholder="Message" value={this.state.message} onChange={e => this.han
dleMessage(e)} />
                      <Button variant="contained" color="primary" onClick={this.sendMessage}>Send</B
utton>
                    </Modal.Footer>
                  </Modal>

                  <div className="container">
                    <div style={{ paddingTop: "20px" }}>
                      <Input value={window.location.href} disable="true"></Input>
                      <Button style={{backgroundColor: "#3f51b5",color: "whitesmoke",marginLeft: "20p
x",
                        marginTop: "10px",width: "120px",fontSize: "10px"
                      }} onClick={this.copyUrl}>Copy invite link</Button>
                    </div>

                    <Row id="main" className="flex-container" style={{ margin: 0, padding: 0 }}>
                      <video id="my-video" ref={this.localVideoref} autoPlay muted style={{
                        borderStyle: "solid",borderColor: "#bdbdbd",margin: "10px",objectFit: "fill",
                        width: "100%",height: "100%"}}></video>
                    </Row>
                  </div>
                </div>
              }
            </div>
        )
    }
}

export default Video
```