Київський національний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Проєктування web-орієнтованої системи консультативної медичної допомоги»

Студентки 2 курсу, 2м групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення»

_____
підпис студента

Сарнавської Марії Романівни

Науковий керівник кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки

_____
підпис керівника

Савченко Тетяна Віталіївна

Гарант освітньої програми доктор економічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки

_____
підпис гаранта

Токар Володимир Володимирович

КИЇВ - 2021

Київський національний торговельно-економічний університет

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.
«10» листопада 2020 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Сарнавській Марії Романівні
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Проєктування web-орієнтованої системи консультативної медичної допомоги»

Затверджена наказом ректора від «30» листопада 2020 р. № 3224

2. Строк здачі студентом закінченого проєкту 25 листопада 2021 р.

3. Цільова установка та вихідні дані до проєкту

Мета проєкту: аналіз побудови та безпосередньо розробка web-орієнтованої системи консультативної медичної допомоги

Об'єкт дослідження: процес створення web-орієнтованої системи

Предмет дослідження: технологія створення Web-додтаку із використанням мови програмування JavaScript

4. Консультанти проєкту із зазначенням розділів, які консультують:

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

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

ВСТУП

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПОБУДОВИ WEB-ДОДАТКУ ТА ВИБІР ПРОГРАМНИХ ЗАСОБІВ

1.1. Поняття Web-додатку та його основних моделей розробки

1.2. Аналіз підходів до створення веб-додатку

1.3. Вибір мови програмування та його фреймворку

1.4. Висновок до розділу 1

РОЗДІЛ 2. ПРОЕКТУВАННЯ ЛОГІКИ РОБОТИ WEB-ДОДАТКУ ТА СТВОРЕННЯ ЙОГО ДИЗАЙНУ

2.1. Побудова схеми роботи відеочату

2.2. Побудова схеми роботи чату з авторизацією через Google

2.3. Побудова користувацького інтерфейсу

2.4. Висновок до розділу 2

РОЗДІЛ 3. РОЗРОБКА WEB-ОРІЄНТОВАНОЇ СИСТЕМИ КОНСУЛЬТАТИВНОЇ МЕДИЧНОЇ ДОПОМОГИ

3.1. Розробка real-time відеочату

3.2. Розробка real-time чату

3.3. Стилізація web-додатку

3.4. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ

КЕРІВНИЦТВО КОРИСТУВАЧА

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2020*	*21.09.2020*
2.	*Розробка та затвердження завдання на проєкт магістра (стац/заоч)*	*06.11.2020*	*22.12.2020*
3.	*Вступ та перелік літературних джерел*	*27.02.2021*	*27.02.2021*
4.	*Розробка технічного завдання*	*20.03.2021*	*20.03.2021*
5.	*Розділ 1. (Теоретичні аспекти побудови web-додатку та вибір програмних засобів)*	*16.04.2021*	*16.04.2021*
6.	*Розділ 2.* (*Проектування логіки роботи web-додатку та створення його дизайну*)	*24.05.2021*	*24.05.2021*
7.	*Розділ 3.* (*Розробка web-орієнтованої системи консультативної медичної допомоги*)	*21.06.2021*	*21.06.2021*
8.	*Розділ 4. (Об'єднання розділів 3 та 4)*	*20.09.2021*	*20.09.2021*
9.	*Розробка програми та методики тестування*	*18.10.2021*	*18.10.2021*
10.	*Написання наукової статті*	*22.05.2021*	*22.05.2021*
11.	*Керівництво користувача*	*21.10.2021*	*21.10.2021*
12.	*Висновки та пропозиції*	*01.11.2021*	*01.11.2021*
13.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*03.11.2021*	*03.11.2021*
14.	*Підготовка автореферату та презентації доповіді*	*03.11.2021*	*03.11.2021*
15.	*Попередній захист випускного кваліфікаційного проєкту*	*22.11.2021 – 25.11.2021*	*22.11.2021*
16.	*Здача зброшурованої випускного кваліфікаційного проєкту*	*25.11.2021*	*25.11.2021*
17.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*26.11.2021*	*26.11.2021*
18.	*Підготовка до публічного захисту випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «10» листопада 2020 р.

8. Науковий керівник випускного кваліфікаційного проєкту Савченко Т.В.

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Токар В.В.

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Сарнавська М. Р.

(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

11. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента Сарнавської М. Р.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Токар В.В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«____»_____20____ р.

АНОТАЦІЯ

Даний випускний кваліфікаційний проєкт присвячений дослідженню важливості та актуальності веб-додатків, їх моделей розробки - характеристика, переваги та недоліки; вивченню, аналізу існуючих способів створення веб-додатку та їх відмінностей. Проведено ознайомлення з різними мовами програмування для побудови web-додатків, визначено їх особливості, обрано найбільш оптимальну мову та її фреймворк для конкретної задачі. Спроектовано схеми з логікою роботи окремих компонентів додатку, створено макет користувацького інтерфейсу. Розроблена система була успішно перевірено на правильність функціонування.

Ключові слова: Web-додаток, SPA, MPA, JavaScript, React, WebRTC, Firebase, Npm.

ABSTRACT

This final qualification project is devoted to the study of the importance and relevance of web applications, their development models - characteristics, advantages and disadvantages; study, analysis of existing ways to create a web application and their differences. Acquaintance with different programming languages for construction of web-applications is carried out, their features are defined, the most optimum language and its framework for a concrete task is chosen. Schemes with logic of work of separate components of application are designed, the layout of the user interface is created. The developed system was successfully tested for proper operation.

Keywords: Web application, SPA, MPA, JavaScript, React, WebRTC, Firebase, Npm.

ЗМІСТ

ВСТУП 3

РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ ПОБУДОВИ WEB-ДОДАТКУ ТА ВИБІР ПРОГРАМНИХ ЗАСОБІВ 5

1.1. Поняття Web-додатку та його основних моделей розробки 5

1.2. Аналіз підходів до створення веб-додатку 9

1.3. Вибір мови програмування та його фреймворку 17

1.4. Висновки до розділу 1 17

РОЗДІЛ 2 ПРОЕКТУВАННЯ ЛОГІКИ РОБОТИ WEB-ДОДАТКУ ТА СТВОРЕННЯ ЙОГО ДИЗАЙНУ 19

2.1. Побудова схеми роботи відеочату 19

2.2. Побудова схеми роботи чату з авторизацією через Google 27

2.3. Побудова користувацького інтерфейсу 25

2.4. Висновки до розділу 2 27

РОЗДІЛ 3 РОЗРОБКА WEB-ОРІЄНТОВАНОЇ СИСТЕМИ КОНСУЛЬТАТИВНОЇ МЕДИЧНОЇ ДОПОМОГИ 28

3.1. Розробка real-time відеочату 28

3.2. Розробка real-time чату 37

3.4. Стилізація web-додатку 49

3.4. Висновки до розділу 3 47

ВИСНОВКИ ТА ПРОПОЗИЦІЇ 48

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50

ТЕХНІЧНЕ ЗАВДАННЯ 53

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ 60

КЕРІВНИЦТВО КОРИСТУВАЧА 66

ДОДАТКИ 67

					КНТЕУ 121 02-18.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*22.12.20*	Проєктування web-орієнтованої системи консультативної медичної допомоги	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Савченко Т.В.		*22.12.20*		*3*	*2*	*52*
Гарант		Токар В.В.		*22.12.20*		*Факультет інформаційних технологій 2 курс, 2м група*		
Розробив		Сарнавська М.Р.		*22.12.20*	*Зміст*			

ВСТУП

Нині Інтернет – це потужний інструмент комунікації, який не має територіальних меж – дозволяє обмінюватися різноманітними видами інформації. Існує безліч Web-сайтів для спілкування та різних розваг, але в контексті медицини таких ресурсів достатньо мало. Мова йдеться про офіційні телемедичні платформи, на яких користувачі мають змогу отримати консультації від лікарів, не виходячи з дому.

Телемедицина є сукупністю дій, технологій і заходів, що використовуються при наданні лікарської допомоги з впровадженням засобів дистанційного зв'язку за допомогою обміну електронними повідомленнями. На сьогодні телемедицина займає провідне місце в системі охорони здоров’я багатьох розвинених країн світу.

Актуальність роботи визначається у створенні зручного та зрозумілого Web-додатку для тих, хто потребує отримати лікарську допомогу дистанційно. У деяких випадках (наприклад, коли відстань і час є критичними чинниками) такий вид консультацій може бути єдиним способом зв’язатися зі спеціалістом та одержати рекомендації щодо лікування. Також варто підкреслити, що такий вид спілкування з лікарем не наражає оточуючих на небезпеку перехопити можливе захворювання будь-якої складності – особливо в наш час глобальних захворювань.

Мета дослідження: аналіз побудови та безпосередньо розробка web-орієнтованої системи консультативної медичної допомоги.

Об’єкт дослідження: процес створення web-орієнтованої системи.

Предмет дослідження: технологія створення Web-додатку із використанням мови програмування JavaScript.

					КНТЕУ 121 02-18.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*27.02.21*	Проєктування web-орієнтованої системи консультативної медичної допомоги	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Савченко Т.В.		*27.02.21*		*В*	*3*	*52*
Гарант		Токар В.В.		*27.02.21*		*Факультет інформаційних технологій 2 курс, 2м група*		
Розробив		Сарнавська М.Р.		*27.02.21*	*Вступ*			

Для досягнення поставленої мети були сформульовані і вирішені наступні *завдання*:

- вивчення предметної області;
- огляд та вибір мови програмування для створення Web-додатку;
- побудова логіки роботи компонентів системи;
- практична розробка додатку;
- перевірка виконаної роботи на коректність та адекватність.

Методи дослідження. Під час вивчення предметної області - поняття Web-додатку, його моделей розробки та підходів до його створення – використовувався метод аналізу та синтезу. При виконанні даної роботи використовувався метод інформаційного моделювання предметної області. Для якісного вирішення поставлених завдань при аналізі мов програмування для побудови Web-додатку був використаний системний та процесний підхід. При дослідженні джерел з даної проблематики та предметної області використовувався метод системного аналізу. При розгляді прийнятих системних термінів з теми дослідження використовувався термінологічний підхід до визначення понять. Інформаційною базою для дослідження та вивчення предметної області, для розробки сайту є мережа Інтернет та відповідна література.

Наукова новизна дослідження полягяє у тому, що робота спрямована на створення веб-орієнтованої системи для онлайн консультування користувачів з питань медицини як за допомогою звичайного чату так і відеочату, аналогів якої в країні немає.

Практична значимість дослідження визначається тим, що її результати будуть корисними для застосування під час різних пандемій (або інших складних становищ), оскільки клієнти матимуть змогу отримати професійну допомогу, не виходячи з дому і не наражаючи себе та оточуючих на небезпеку.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ПОБУДОВИ WEB-ДОДАТКУ ТА ВИБІР ПРОГРАМНИХ ЗАСОБІВ

1.1. Поняття Web-додатку та його основних моделей розробки

Існує багато різних типів програм. Бондаренко у своїй роботі [1] поділяє програмне забезпечення на веб, мобільні та гібридні програми (Рис. 1.1). Зазначається, що завжди необхідно зважити всі за і проти кожного виду, коли приймається рішення, що найкраще відповідає конкретним потребам, оскільки кожен додаток слугує різним цілям.

Рис. 1.1. Основні типи програм

Мобільні програми - це програмні додатки, розроблені для використання на певній платформі чи пристрої, наприклад, Android або IoS.

Гібридні програми - поєднують елементи мобільних та веб-програм. На перший погляд, гібридні програми виглядають так само, як будь-які мобільні програми. Вони встановлюються на мобільний пристрій та побудовані за допомогою таких самих мов програмування, як і мобільні додатки. Однак всередині вони, по суті, є веб-програмами з інформаційною панеллю.

Веб-додаток - це комп'ютерна програма, яка використовує веб-браузери та веб-технології для виконання завдань через Інтернет.

Веб-програми використовують комбінацію сценаріїв на стороні сервера (PHP та ASP) для обробки та зберігання інформації, а також сценарії на стороні клієнта (JavaScript та HTML) для подання інформації користувачам.

Це дозволяє користувачам взаємодіяти з компанією, використовуючи онлайн-форми, системи управління контентом, візки для покупок тощо. Крім того, програми дозволяють працівникам створювати документи, обмінюватися інформацією, співпрацювати над проектами та працювати над загальними документами незалежно від місця розташування та пристрою.

Опираючись на працю Шевкуна Б. О. «Програмне забезпечення браузера для перегляду веб-сайтів» [2], пропоную наступне пояснення принципу роботи веб–додатку. Веб-додатки зазвичай кодуються мовами, підтримуваними браузером, такими як JavaScript та HTML, оскільки ці мови залежать від браузера для виконання програми. Деякі програми є динамічними і потребують обробки на стороні сервера (Рис. 1.2). Інші повністю статичні і не вимагають обробки на сервері.

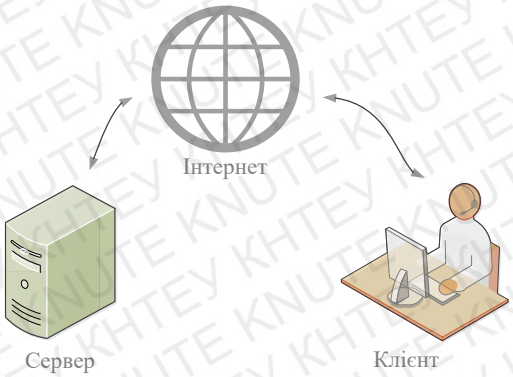


Рис. 1.2. Проста схема роботи динамічних програм

Веб-програма вимагає, щоб веб-сервер керував запитами клієнта, сервер додатків виконував запитані завдання, а іноді й базу даних для зберігання інформації. Технологія сервера додатків варіюється від ASP.NET, ASP та ColdFusion до PHP та JSP.

Ось як виглядає типовий потік веб-додатків [3]:

- Користувач запускає запит до веб-сервера через Інтернет або через веб-браузер, або через інтерфейс користувача програми;
- Веб-сервер пересилає цей запит на відповідний сервер веб–додатків;
- Сервер веб-додатків виконує запитуване завдання, наприклад запит до бази даних або обробку даних, а потім генерує результати запитуваних даних;

- Сервер веб-додатків надсилає результати на веб-сервер із запитаною інформацією або обробленими даними;
- Веб-сервер відповідає клієнту із запитаною інформацією, яка потім з'являється на дисплеї користувача

Веб-програми включають онлайн-форми, візки для покупок, текстові процесори, електронні таблиці, редагування відео та фотографій, перетворення файлів, сканування файлів та програми електронної пошти, такі як Gmail, Yahoo та AOL. Серед популярних програм - Google Apps і Microsoft 365. У Google Apps for Work є Gmail, Документи Google, Таблиці Google, Презентації Google, Інтернет-сховище тощо. Інші функції включають обмін документами та календарями в Інтернеті. Це дозволяє всім членам команди одночасно отримувати доступ до однієї і тієї ж версії документа.

Науковець Захарченко А. П. зауважує, що вибір правильної архітектури компонентів веб-додатків покращує якість роботи майбутнього веб-додатка [4]. Розглянемо плюси та мінуси можливих моделей.

Таблиця 1.1.

ПЕРЕВАГИ ТА НЕДОЛІКИ РІЗНИХ МОДЕЛЕЙ ВЕБ-ДОДАТКІВ

	Переваги	Недоліки
Один веб-сервер (з базою даних)	Швидкість розробки; підходить для невеликих проектів	Висока ймовірність відмов роботи
Два+ веб-сервери, одна база даних	Наявність взаємозамінних серверів	Відсутність додаткового, екстреного сховища даних
Два+ веб-сервери, дві+ бази даних	Відмовостійка; надійно зберігаються дані	Складність розробки
Мікросервіси та без сервера	Достатньо висока продуктивність роботи додатку	Уразливі для атак на безпеку

Один веб-сервер (з базою даних) - це найпростіша та найризикованіша модель, де одна база даних є частиною єдиного сервера веб-програми. Якщо сервер знижується, веб-додаток також знижується. Рекомендується обирати дану модель тільки якщо веб-додаток є тестовим проектом або приватною практикою.

Два+ веб-сервери, одна база даних. Ідея цієї моделі полягає в тому, що веб-сервер не повинен зберігати жодних даних: навіть коли він отримує інформацію від клієнта, веб-сервер обробляє її, записує дані в базу даних (на фізично окремому комп'ютері) і забуває про це. Маючи принаймні два веб-сервери, значно зменшується ризик збоїв. Навіть якщо один із веб-серверів коли-небудь вийде з ладу, інший негайно бере роботу на себе - усі запити автоматично переадресовуються на новий сервер, і веб-додаток продовжує працювати. Однак, лише з однією базою даних все ще існує певний ризик: якщо вона вийде з ладу, вийде з ладу і вся система.

Два+ веб-сервери, дві+ бази даних. Цю модель можна вважати найбільш відмовостійкою: ні веб-сервери, ні бази даних не мають єдиних точок відмови.

У такому разі є два варіанти - зберігати однакові дані на кожному з комп'ютерів баз даних, або рівномірно розподілити дані між базами даних. Незважаючи на очевидну перевагу економії місця для зберігання, цей параметр створює ризик тимчасового недоступності деяких даних у разі збою бази даних.

Мікросервіси та безсерверна архітектура були винайдені для того, щоб підвищити спритність веб-додатків, спростивши оновлення та масштабування. В обох цих моделях веб-сервери розбиті на менші компоненти: "послуги" в мікросервісах і "функції" у безсерверних. Кожен з цих невеликих компонентів існує в окремому контейнері і обробляється незалежно, що полегшує його зміну або масштабування.

Мікросервіси дешевші в обслуговуванні та дозволяють швидше виходити на ринок. Однак, через посилену взаємодію між кількома компонентами, мікросервіси та веб-програми без серверів можуть запропонувати нижчу продуктивність та становити загрозу безпеці при неправильній реалізації.

1.2. Аналіз підходів до створення веб-додатку

Джош Пауелл та Майкл Міковскі вважають, що для того, щоб веб-додаток працював без небажаних взаємодій, його має підтримувати відповідна технологія, що гарантує високу продуктивність та швидкість [7]. Існує два способи розробки веб-програм: односторінкові програми та багатосторінкові програми. Обидва мають плюси і мінуси.

Односторінковий додаток (SPA) - це більш сучасний підхід до розробки додатків. Він використовувався Google, Facebook, Twitter тощо. SPA - це програма, яка працює всередині браузера і не вимагає перезавантаження сторінок під час використання.

З іншого боку, багатосторінкова програма вважається більш класичним підходом до розробки додатків. Шаблон багатосторінкового дизайну вимагає перезавантаження сторінки щоразу, коли змінюється вміст (Рис. 1.3). Це кращий варіант для великих компаній з великим портфелем продуктів, таких як підприємства електронної комерції.

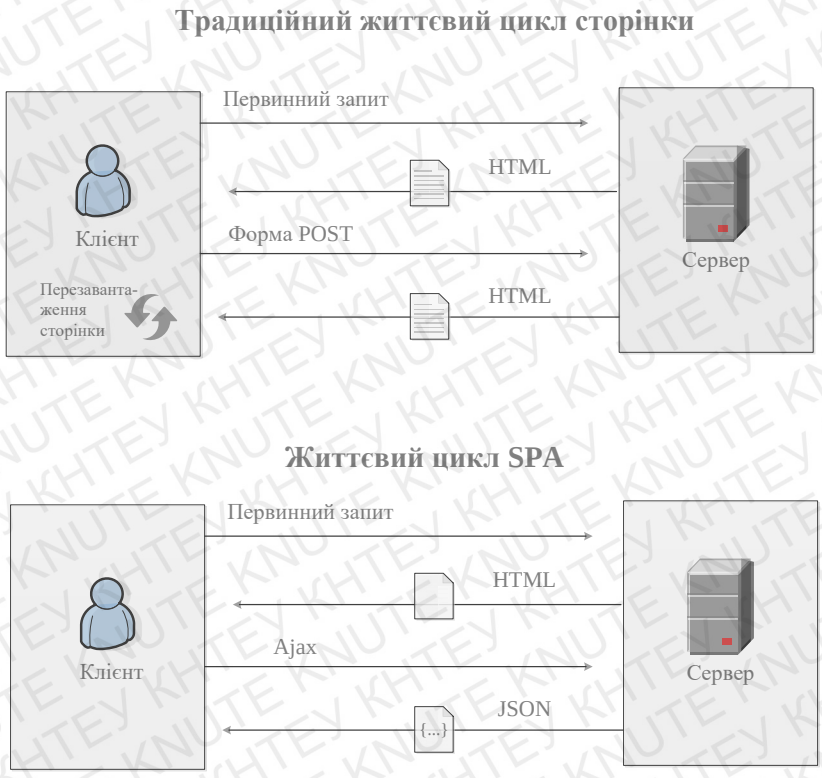


Рис. 1.3. Різниця в функціонуванні односторінкового та багатосторінкового додатку

І SPA, і MPA базуються на протоколі HTTP. У традиційній архітектурі клієнта/сервера MPA кожен клік користувача викликає HTTP - запит до сервера. Результатом цього нового запиту є повне оновлення сторінки, навіть якщо частина вмісту залишається незмінною.

З іншого боку, ядро SPA базується на Ajax, наборі методів розробки, що дозволяє клієнту асинхронно (у фоновому режимі) надсилати та отримувати дані з сервера, не заважаючи відображенню та поведінці веб-сторінки. Ajax дозволяє веб-сторінкам і, відповідно, веб-програмам, динамічно змінювати вміст без необхідності перезавантажувати всю сторінку. Для цього SPA значною мірою покладається на JavaScript. Фреймворки JavaScript, такі як React, Vue, Angular та Ember, відповідають за обробку складного вантажу на стороні клієнта.

SPA працює за таким принципом [8]:

- Клієнт спочатку підключається до сервера і отримує вміст сторінки, який в основному відповідає HTML-коду, CSS та пакету JavaScript, що містить усі сценарії JavaScript, необхідні для запуску логіки програми;
- Дія користувача запускає виконання відповідних JavaScript-ів, які, у свою чергу, запитують дані на сервер за допомогою викликів Ajax. Дані зазвичай подаються у форматі JSON і не запускають повне оновлення веб-сторінки.

Основні переваги та недоліки підходів SPA та MPA наведені в таблиці [9]:

Табл. 1.2.

ВІДМІННОСТІ SPA ТА MPA АРХІТЕКТУР

	SPA	MPA
Швидкість	Висока	Низька
Зчеплення	Розділені компоненти	З’єднані компоненти
Пошукова оптимізація	Низька	Висока
Досвід користувача	Залежить від інтерфейсу	Залежить від інтерфейсу
Безпека	Низька	Висока

Продовження таблиці 1.2.

Процес розробки	Відносно швидкий	Повільний
Залежність від JavaScript	Залежний	Не залежний

Швидкість є важливим фактором - тривалість уваги людей стає все коротшою. SPA завантажується швидше, оскільки він завантажує більшість ресурсів програми лише один раз. MPA повільніше, оскільки браузер повинен перезавантажити всю сторінку з нуля, коли користувач хоче отримати доступ до нових даних або перейти в іншу частину веб-сайту. Оптимальний час завантаження веб-сайту - 0,4 секунди.

Зчеплення. SPA сильно роз'єднаний, що означає, що інтерфейс і бекенд розділені. Односторінкові програми використовують API, розроблені на стороні сервера, для читання та відображення даних. У MPA інтерфейс та бекенд більш взаємозалежні. Все кодування зазвичай розміщується в одному проекті.

Пошукова оптимізація. Більшість односторінкових додатків працюють на JavaScript, який більшість пошукових систем не підтримує. MPA дозволяє краще позиціонувати веб-сайт, оскільки кожна сторінка може бути оптимізована для різних ключових слів. Крім того, мета-теги можна включати на кожну сторінку - це позитивно впливає на рейтинг Google.

Досвід користувача. SPA більш зручні для мобільних пристроїв, і про це варто пам’ятати, оскільки велика кількість трафіку надходить з мобільних пристроїв. Навіть Google почав надавати пріоритет мобільному досвіду над настільним ПК. Фреймворки, що застосовуються при розробці SPA, дозволяють розробляти мобільні додатки. MPA, з іншого боку, забезпечують кращу архітектуру інформації. Можна створити стільки сторінок, скільки потрібно, і можна включити на сторінку стільки інформації, скільки потрібно, без будь-яких обмежень. Навігація зрозуміла, тому користувач може легко орієнтуватися на веб-сайті, що позитивно впливає на їхній досвід.

Безпека. Чим більший веб-сайт, тим більше зусиль потрібно для його захисту. У MPA доведеться захищати кожну веб-сторінку, а в SPA всього одну сторінку. Однак SPA більш схильні до хакерських атак, оскільки вони працюють на JavaScript, який не виконує компіляцію коду, що робить його більш вразливим до шкідливого програмного забезпечення.

Процес розробки. Однією з найбільших переваг SPA є бекенд-код багаторазового використання. Можна застосувати до власного мобільного додатку той самий код, який використовувався у своєму веб-додатку. Це важливо, оскільки програми та веб-сайти часто відкриваються на мобільних пристроях для економії часу. Також завдяки чіткому розподілу між френтендом та бекендом, обидві частини можна створювати одночасно, що прискорює весь процес розробки. Розробка MPA займає більше часу, оскільки в більшості випадків сторона сервера має бути закодована з самого початку.

Залежність від JavaScript. SPA повністю залежить від JavaScript, що може бути проблематичним. Якщо додаток працює у веб-переглядачі з вимкненим JavaScript, це може спричинити проблеми з функціональністю програми, що може призвести до вищих показників відмов та зниження конверсії. Належність JavaScript також сприяє його проблемам із оптимізацією SEO та проблемами безпеки. MPA можна створити без будь -якої залежності від JavaScript.

1.3. Вибір мови програмування та його фреймворку

Існує приказка про «правильний інструмент для правильної роботи», і це дуже стосується мови програмування. Не всі мови програмування спроектовані однаково, і тому вони не однаково хороші для будь-яких завдань. Наприклад, Java найкраще підходить для створення бекендів, JavaScript добре підходить як для інтерфейсу, так і для бекенду, і не дивно, що це найкраща мова програмування для веб-розробки, яку використовують 67,7% респондентів сайту stackoverflow [10] (Рис. 1.4):

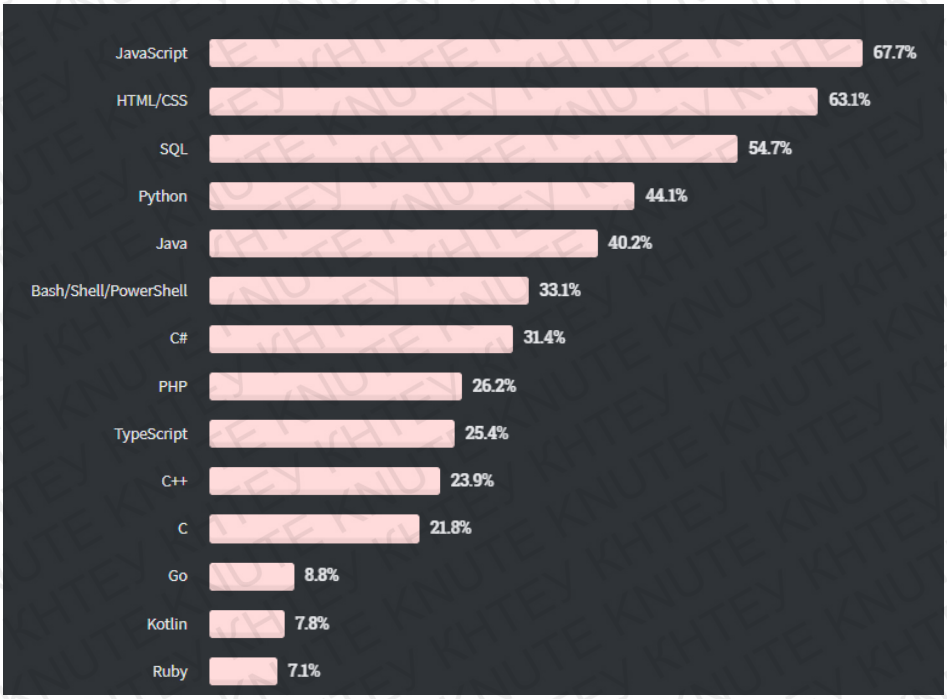


Рис. 1.4. Рейтинг найбільш популярних мов програмування/технологій за 2020 рік

Саме для розробки веб-додатку на основі графіку та праці Нікітченка М. С. можна сформувати такий топ мов програмування саме для розробки веб-додатків [11]:

1. JavaScript. Безсумнівно, що JavaScript - це найпопулярніша мова серед веб-розробників. Це також єдина мова, яка дозволяє створювати веб-додатки, як інтерфейсні, так і серверні, а також мобільні (React Native). Сильна сторона Javascript полягає не тільки в тому, що він може працювати у браузері та на сервері за допомогою Nodejs, а й у чудових фреймворках та бібліотеках, які він має для веб-розробки та розробки додатків. Наприклад, можна використовувати React.js та Angular для зовнішнього інтерфейсу, Nodejs для бекенда та React Native для створення кроссплатформенних мобільних додатків.

2. Python - мабуть, найбільш універсальна мова програмування на даний момент. Python можна використовувати для веб-розробки, створення сценаріїв та автоматизації. Python також має багато корисних фреймворків, бібліотек та інструментів, які можуть допомогти у швидкому створенні веб-програми. Наприклад, Django – гарний вибір для створення повномасштабних веб-програм.

3. TypeScript – доволі сучасна мова програмування для веб -розробки. Перевага TypeScript полягає в тому, що вона додає безпеку типів у коді JavaScript, що означає, що можна виявити неприємні помилки, пов'язані з типом JavaScript, на етапі розробки. Це також спрощує розробку об'єктно-орієнтованого коду для JavaScript, а кілька вбудованих засобів налагодження TypeScript роблять веб-розробку дуже легкою.

4. PHP також є однією з найкращих мов програмування, коли йдеться про створення веб -додатків. Це динамічна мова сценаріїв на стороні сервера, що робить дійсно простим створення повністю функціональних веб-додатків. WordPress - найпопулярніше програмне забезпечення для веб-додатків, створене на PHP.

5. Ruby - це ще одна мова програмування, яка чудово підходить для веб -розробки. Подібно до PHP та Python, Ruby також простий у вивченні та доступний для початківців. Що робить Ruby особливим для веб-розробки - це платформа Ruby on Rails, яка підтримує такі веб-сайти, як Github, Shopify, Airbnb, Groupon, GoodReads та Kickstarter.

Отже, JavaScript – ідеальний варіант для створення веб-додатку, оскільки він має низку переваг: його код простий і гнучкий; його можна використовувати всередині сценаріїв, написаних іншими мовами програмування; може перетворювати статичні веб-сайти у веб-програми, дозволяючи розробникам додавати функціональні можливості меню, анімації та взаємодію з курсором.

Більшість фреймворків JavaScript мають детальну документацію з повним описом функцій, які збільшують швидкість розробки веб-додатків. Angular підтримується Google, а фреймворк React JavaScript підтримується Facebook.

Кілька років тому розробники JavaScript в основному використовували React та Angular. Але оскільки у спільноти розробників зростає інтерес до Vue, усі три фреймворки JavaScript зайняли міцні позиції на ринку розробки програмного забезпечення (Рис. 1.5) [12].

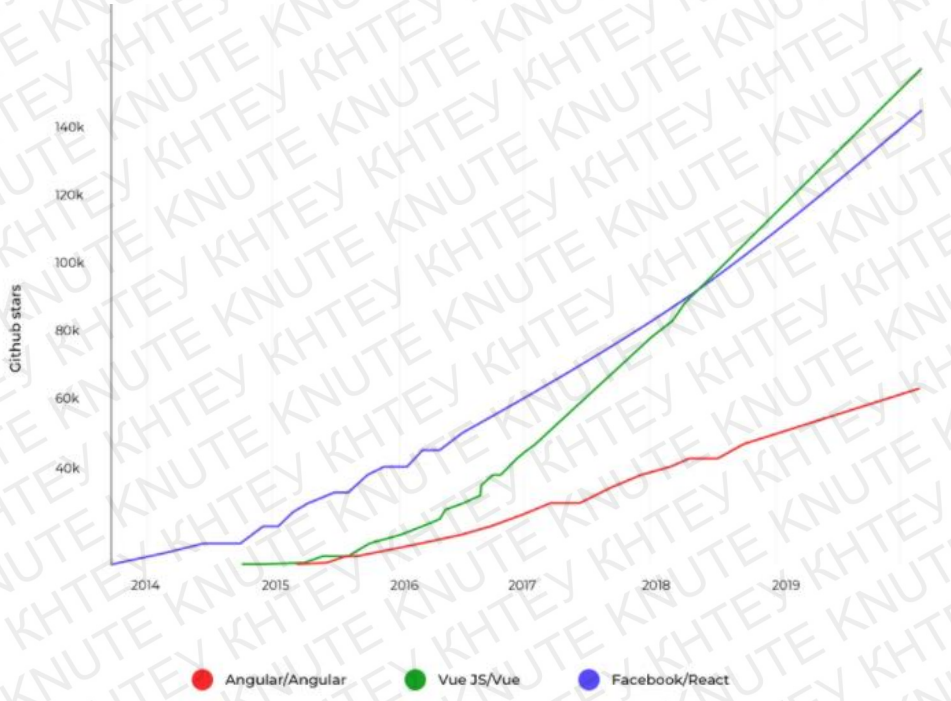


Рис. 1.5. Зміна популярності фреймворків Angular, Vue та React впродовж 2014-2019х років

Щоб охопити всі аспекти сучасної розробки веб-додатків, більшість фреймворків працюють з додатковими інструментами. Однак Angular може виконати роботу самостійно. Ця структура забезпечує якісні готові шаблони та компоненти. Деякі з найкращих веб-програм, таких як Google Play Store, веб - програма Microsoft Xbox, Office 365 для Інтернету, Netflix, YouTube та PayPal, базуються на Angular [12]. Angular має модульну структуру, що дозволяє розробникам організовувати канали, послуги, директиви та інші компоненти в окремі сегменти. Ці незалежні компоненти легко замінити або масштабувати. Крім того, Angular часто оновлюється, і кожне нове оновлення покращує продуктивність.

React більше схожий на функціональну бібліотеку інтерфейсу користувача, а не на повноцінну структуру. Це дозволяє розробникам створювати користувальницькі інтерфейси з компонентами, які можна змінювати з часом без необхідності переписувати хитрий код. Більш того, хоча більшість фреймворків JavaScript погано поводять себе з пошуковими системами, веб-сайти React відображаються на стороні сервера, що робить їх швидшими та покращує SEO.

Vue вважається прогресивною основою для розробки односторінкових додатків (SPA). Ідея Vue полягала в тому, щоб кодувати лише кілька рядків, таким чином досягаючи хороших результатів з мінімальними зусиллями. Сьогодні ця рамка JavaScript невелика і перевершує громіздкі Angular та React. Кожен може швидко завантажити Vue і почати його використовувати. Vue містить такі функції, як спостерігачі, обчислювані властивості та директиви, які спрощують трудомісткий процес розробки. Крім того, його можна інтегрувати в інші веб-програми, вбудовані в JavaScript.

Табл. 1.3.

ПОРІВНЯННЯ ФРЕЙМВОРКІВ ANGULAR, VUE ТА REACT

	React	Angular	Vue.js
Випадки використання	Динамічні та односторінкові програми, а також мобільні програми за допомогою React Native.	Прогресивні веб-програми, гібридні мобільні програми та веб-програми корпоративного рівня.	Ідеально підходить для динамічних та односторінкових додатків.
Функціональність	Завдяки великій популярності бібліотеки існує багато сторонніх сервісів для React.	Монолітний каркас із широкими функціональними можливостями, що робить додатки Angular більш важкими.	Обмежений основний функціонал, який можна легко розширити за допомогою сторонніх рішень.
Мова програмування	React використовує JSX - дещо змінену версію JavaScript. Це дозволяє	TypeScript для кращої продуктивності	Vue.js використовує чистий JavaScript. Це одна з причин

Продовження таблиці 1.3.

	ефективніше програмувати, але потребує деякого часу для ознайомлення.	або чистий JavaScript.	такої популярності бібліотеки серед розробників - початківців.
Технічна документація	React має ідеально складену документацію англійською мовою. Також є гідні підручники іншими мовами.	Відмінна детальна офіційна документація, десятки вичерпних навчальних посібників.	Документація в наявності. Все більше користувачів створюють власні підручники та посібники для Vue.js різними мовами

1.4. Висновки до розділу 1

Отже, веб-додатки є оптимальним, економічним варіантом для будь-якої організації. Їх не потрібно їх завантажувати чи встановлювати, тому що це програми, доступ до яких здійснюється через веб-браузер. У порівнянні з настільними програмами, веб-програми легше підтримувати, оскільки вони використовують один і той самий код у всьому додатку. Проблем із сумісністю немає. Також на відміну від веб-сайту веб-програма призначена для взаємодії з кінцевим користувачем, він складається із динамічного контенту (SPA), що значно прискорює всю роботу системи.

Для розробки веб-додатку було обрано мову JavaScript, оскільки його код легко пишеться та налагоджується. Також замість того, щоб виконуватися на сервері веб-сайту, JavaScript виконується на пристрої користувача.

Це зводить до мінімуму запити сервера та покращує роботу користувача. Для розширення можливостей JavaScript використовуватиметься фреймворк React, який добре підходить для створення динамічниї односторінкових програм та має низку зрозумілої документації.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ЛОГІКИ РОБОТИ WEB-ДОДАТКУ ТА СТВОРЕННЯ ЙОГО ДИЗАЙНУ

2.1. Побудова схеми роботи відеочату

Перед розробкою схеми роботи відеочату, розрахованого на багато користувачів, з використанням фреймворку React JavaScript розберемося з поняттям WebRTC, оскільки в даному проекті буде застосовуватися саме ця технологія. WebRTC (Web Real Time Communications) - це стандарт, який описує пряму передачу потокових даних між браузерами в реальному часі [13].

Маємо на меті створити відео чат між двома клієнтами - браузером №1 і браузером №2 - тобто по суті нам потрібно встановити пряме WebRTC-з'єднання між двома клієнтами (його також називають p2p з'єднанням). Для зв'язку цих двох клієнтів нам знадобиться допоміжний сервер. У стандарті WebRTC описано, що ми можемо реалізувати його самі - тобто на зручних нам технологіях [13].

Для того щоб встановити пряме з'єднання між клієнтами нам потрібно знати IP-адреси цих клієнтів, але у нас вони є не завжди, оскільки не всі клієнти володіють публічною статичною IP-адресою [14]. Більшість пристроїв, які виходять в Інтернет, знаходиться за NAT - тобто вони підключаються до роутерів, роутер володіє статичною публічною IP-адресою і він транслює нашу локальну внутрішню адресу пристрою у зовнішню публічну адресу.

Перша задача - з нашого клієнта дізнатися нашу публічну адресу. Для цього є технологія, яка називається STAN. Це мережевий протокол, який дозволяє клієнту дізнатися свою публічну адресу. STAN сервер є у Google, їх є безліч безкоштовних, з яких ми й оберемо відповідно до нашого проекту.

Після того як ми дізналися IP-адресу, порт і всі інші метадані про нашого

клієнта, ми починаємо захоплювати медіа-контент (відео, аудіо). Далі нам потрібно їх відправити через сервер сигналізації. Працює це так [15]: клієнт відправляє наші медіа дані (або ж запит на підключення), а також ICE-кандидат (- це мета-дані, які ми зібрали - тобто наш IP, порт) через сервер сигналізації іншому клієнту, до якого він хоче підключитися; другий клієнт в свою чергу робить те ж саме - він захоплює свої медіа дані (відео, аудіо), а також свій ICE-кандидат (свою IP-адресу, порт, які він також отримав на STAN сервері) і відправляє їх у відповідь - тільки вже він відправлятися не offer, а answer. І після того як answer прийшов другому клієнту, встановлюється пряме WebRTC-з'єднання (Рис. 2.1).

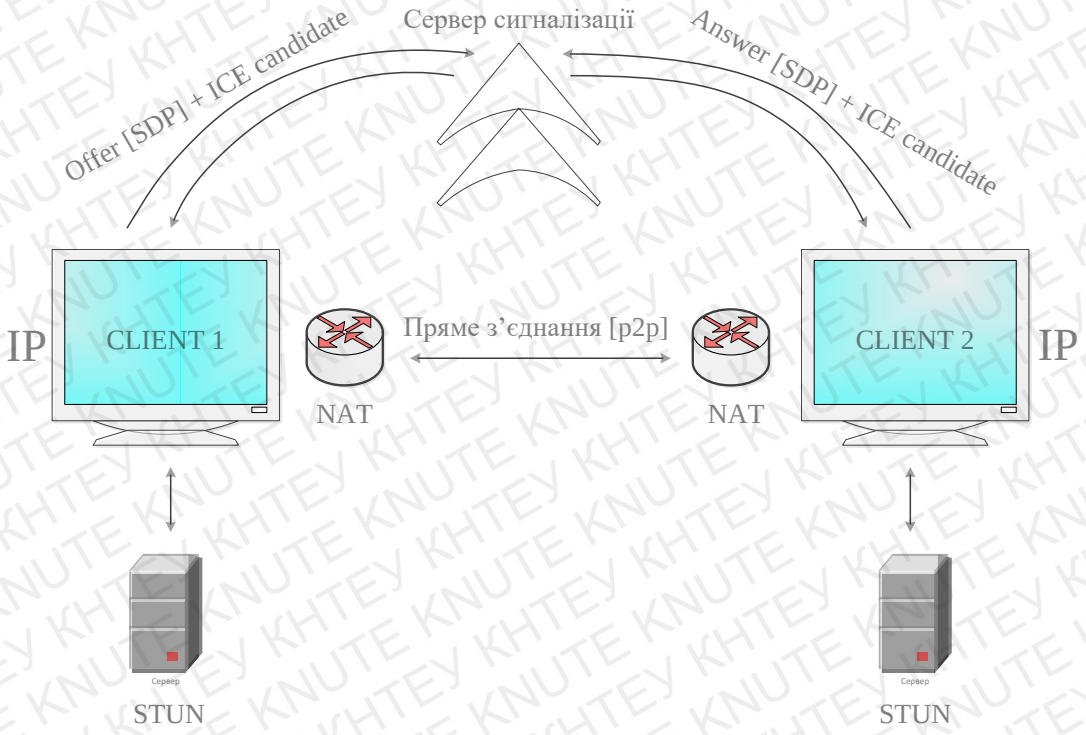


Рис. 2.1. Схема встановлення прямого з'єднання між клієнтами

Бувають випадки, коли WebRTC-з'єднання встановити неможливо, наприклад при використанні 3G. В такому випадку додається ще одна технологія, яка називається TURN-сервер. Він використовується в крайньому випадку як посередник, перетворюючи p2p-з'єднання в клієнт-серверний зв'язок - тобто ми ходимо через сервер між клієнтами. Але такі випадки потрібно уникати, оскільки з TURN-сервером все працює повільніше і якість зображення буде гірше (Рис. 2.2):

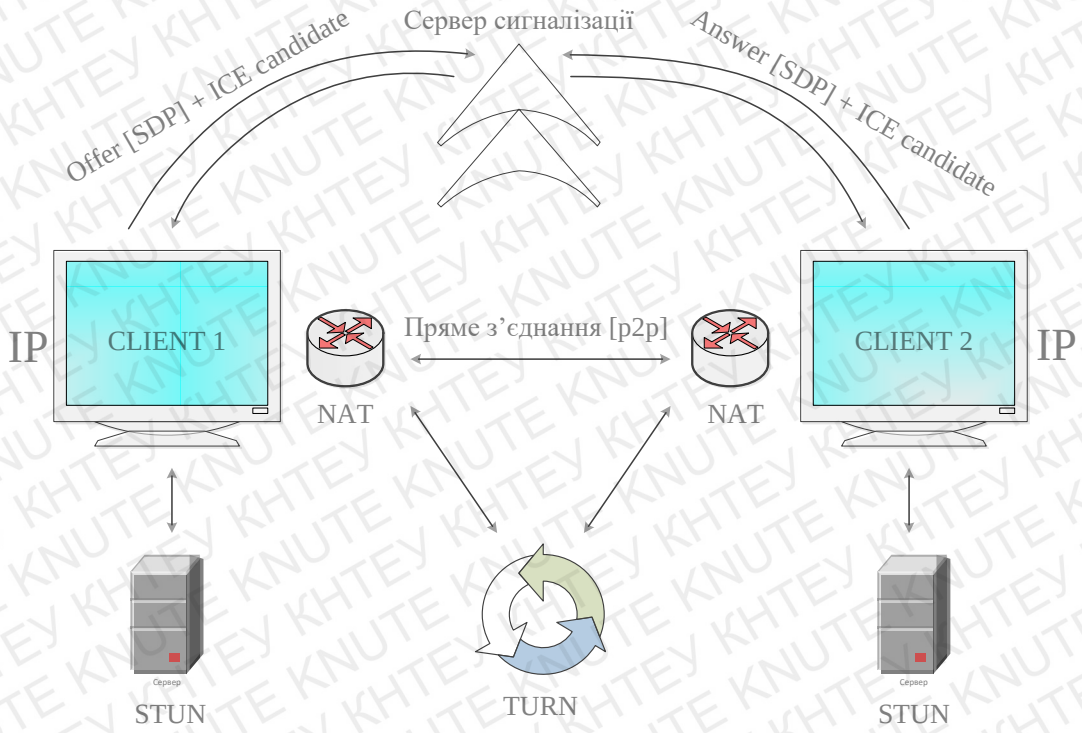


Рис. 2.2. Схема встановлення прямого з'єднання між клієнтами для більш складних випадків

Конкретно в контексті JavaScript розберемо, як буде влаштований додаток. В клієнті №1 створимо instance RTCPeerConnection, якому передамо config ice-серверів - тобто це JSON, в якому є url-адреси STUN і TURN-серверів. Після цього нам потрібно буде захопити наш медіаконтент - тобто відео та аудіо - та встановити його як local description цьому RTCPeerConnection, тобто це той контент, який ми хочемо передавати. При встановленні local description спрацьовує метод onIceCandidate, після цього ми створюємо offer на підключення, відправляємо наш Ice Candidate - тобто наші метадані про клієнта, а також offer іншому клієнту.

У свою чергу клієнт №2 також у себе створює RTCPeerConnection. І в той момент, коли він отримує offer, він його собі встановлює через метод setRemoteDescription. Потім він бере Ice Candidate, який він отримав, і додає його до себе - тобто ми вказуємо, що з цим айс кандидатом ми будемо встановлювати зв'язок. Після цього клієнт №2 захоплює свій локальний контент (відео, аудіо) і як і перший клієнт створює відповідь. У нього також спрацьовує метод onIceCandidate - тобто він дізнається свої метадані, свій IP - і все це

відправляється клієнту №1. А клієнт №1 робить те ж саме - він додає собі айс кандидатом клієнта №2 і викликає setRemoteDescription з відповіддю. Тобто ми встановили локальний контент і віддалений контент обом клієнтам, і після цього у нас починається пряма передача потокових даних (Рис. 2.3):

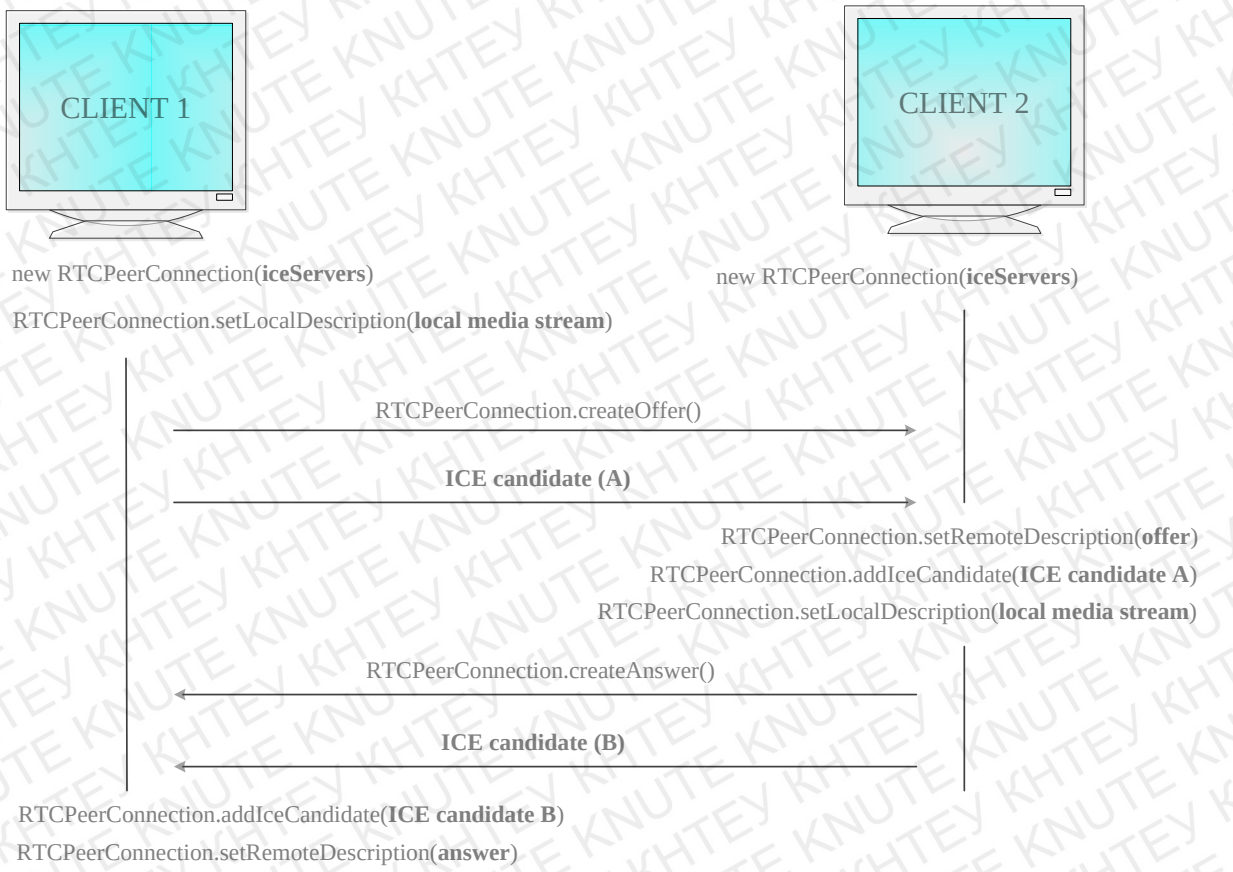


Рис. 2.3. Схема встановлення зв'язку між двома клієнтами в контексті JavaScript

2.2. Побудова схеми роботи чату з авторизацією через Google

У нашому проекті real-time чат буде реалізовуватися за допогомою платформи Firebase. Ця платформа призначена для розробки мобільних та веб-додатків і вона надає розробникам безліч інструментів та послуг, які допомагають їм розробляти високоякісні програми, розширювати базу користувачів та отримувати більший прибуток [16].

База даних Firebase Realtime - це хмарна база даних NoSQL, яка дозволяє зберігати та синхронізувати дані між користувачами в режимі реального часу. У цьому випадку база даних реального часу - це один великий об’єкт JSON,

яким розробники можуть керувати в режимі реального часу [17]. За допомогою лише одного API база даних Firebase надає додатку як поточне значення даних, так і будь-які оновлення цих даних.

Однією з переваг бази даних реального часу полягає в тому, що вона поставляється з мобільними та веб-пакетами SDK, що дозволяє створювати свої програми без необхідності серверів.

База даних реального часу також може інтегруватися з автентифікацією Firebase, щоб забезпечити простий та інтуїтивно зрозумілий процес автентифікації. Є можливість автентифікувати користувачів свого додатка такими способами - Електронна адреса та пароль, Номери телефонів, Google, Facebook, Twitter і т.д.

База даних у реальному часі є базою даних NoSQL і, як така, має різні оптимізації та функціональні можливості порівняно з реляційною базою даних [18]. API баз даних у реальному часі призначений лише для операцій, які можна виконувати швидко (Рис. 2.4). Це дозволяє створювати чудовий досвід роботи в режимі реального часу, який може обслуговувати мільйони користувачів без шкоди для чуйності. Через це важливо подумати про те, як користувачам потрібно отримати доступ до ваших даних, а потім структурувати їх відповідно.

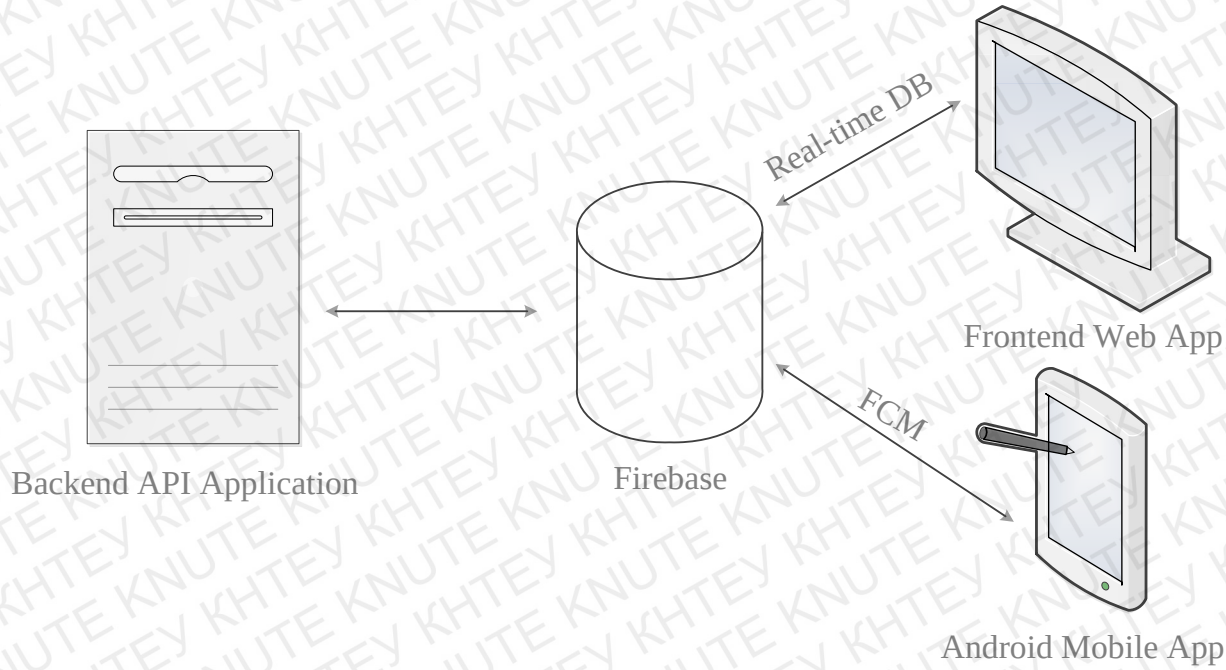


Рис. 2.4. Схема роботи Firebase Realtime Database

Розберемося, як працюватиме Realtime чат за допомогою схеми, наведеної нижче (Рис. 2.5). Для початку створюється об’єкт auth, за допомогою якого ми будемо авторизовуватися. Далі ми, використовуючи функцію signInWithPopup(), отримуємо провайдер авторизації, куди клієнт вводить свої логін та пароль від акаунту Google. Другий користувач проводить ті ж маніпуляції зі свого пристрою.

Після цього обом відкривається доступ до чату. Як для відправлення повідомлень, так і для отримання колекції всіх повідомлень застосовуватиметься об’єкт firestore, оскільки ми використовуємо базу даних Firebase – тобто ми кожного разу будемо до неї звертатися. У випадку з отриманням повідомлень в функцію useCollectionData() буде передаватися цей об’єкт, а у разі відправлення - об’єкт firestore викликатиме функцію add(), в якому передаватиме необхідні дані: ID користувача, ім’я, фотографія, саме повідомлення, час відкравки. Після цього можна вважати, що ми налаштували realtime чат – користувачі можуть обмінюватися повідомленнями.

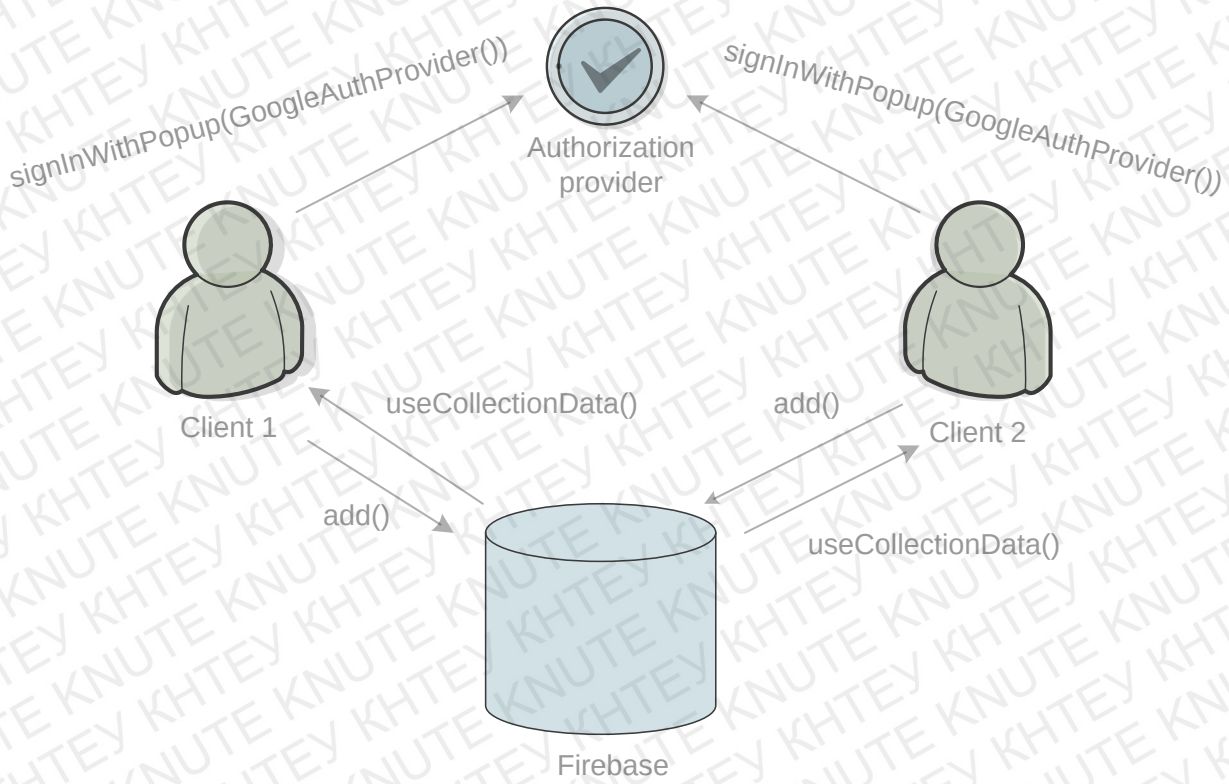


Рис. 2.5. Схема збереження та отримання повідомлень з Firebase Realtime Database

2.3. Побудова користувацького інтерфейсу

Для розробки дизайну web-додатка будемо використовувати Figma. Figma - це інструмент веб-дизайну інтерфейсу, який складається з потужних функцій для веб-дизайну. Цей інструмент забезпечує спільну та економічно ефективну платформу для створення гідних веб-дизайнів [19]. Його можна використовувати для виконання різних завдань, таких як векторні ілюстрації, дизайн інтерфейсу користувача, дизайн додатків та створення прототипів.

Ця програма дозволяє користувачам працювати над своїми проектами в режимі онлайн та офлайн за допомогою настільної програми. Файли можна змінювати без перебування користувача в мережі. Figma має функцію синхронізації, яка інтегрує зміни, зроблені в настільному додатку, з його веб-програмою [20]. Ця функція працює, коли користувач виходить в Інтернет.

Для створення дизайну в Figma знадобиться увійти в обліковий запис (або створити його), додати фрейм – основне поле, де будуть розміщуватися компоненти - та дати йому назву. Після цього треба буде лише додавати різні фігури (прямокутник, коло, лінія), текстові поля на даний фрейм і змінювати стилі цих елементів – колір і розмір тексту, фону і так далі.

Отже, перша задача – спроектувати початковий екран додатка, тобто який з’явиться відразу після завантаження сторінки. Із самого верху розмістимо шапку сайту, де знаходитимуться пункти меню та кнопка «Увійти - Вийти», аби користувач мав змогу авторизуватися, або вийти зі свого облікового запису в будь-який момент. Дана функція необхідна для того, щоб в подальшому мати змогу приєднатися до real time чату. Вхід у систему буде відбуватися через Google акаунт, тому відразу під шапкою слід розмістити кнопку «Увійти за допомогою Google», після натискання на яку відкриється вікно для входу в Google акаунт (Рис. 2.6):

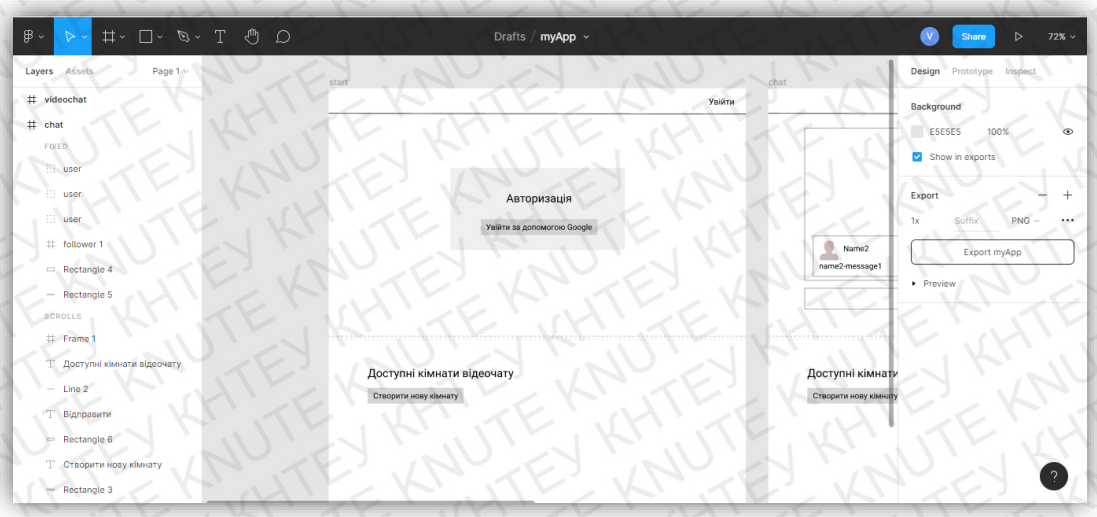


Рис. 2.6. Дизайн кнопки авторизації

Для real time відеочату нам знадобиться тільки одна кнопка «Створити нову кімнату», аби в подальшому отримати посилання на сам відеочат, щоб потім до нього змогли приєднатися.

На даному зображенні (Рис. 2.7) розглянемо інтерфейс нашого чату для обміну повідомленнями. Як зазначалося, отримати таку сторінку можна лише після того, як користувач авторизувався за допомогою Google. Тож на екрані розміщений блок, в якому саме будуть відображатися всі повідомлення, а також ім’я та фотографія юзера, які підтягнуться в систему із Google акаунта. Нижче знаходиться поле для вводу повідомлень та кнопка «Відправити».

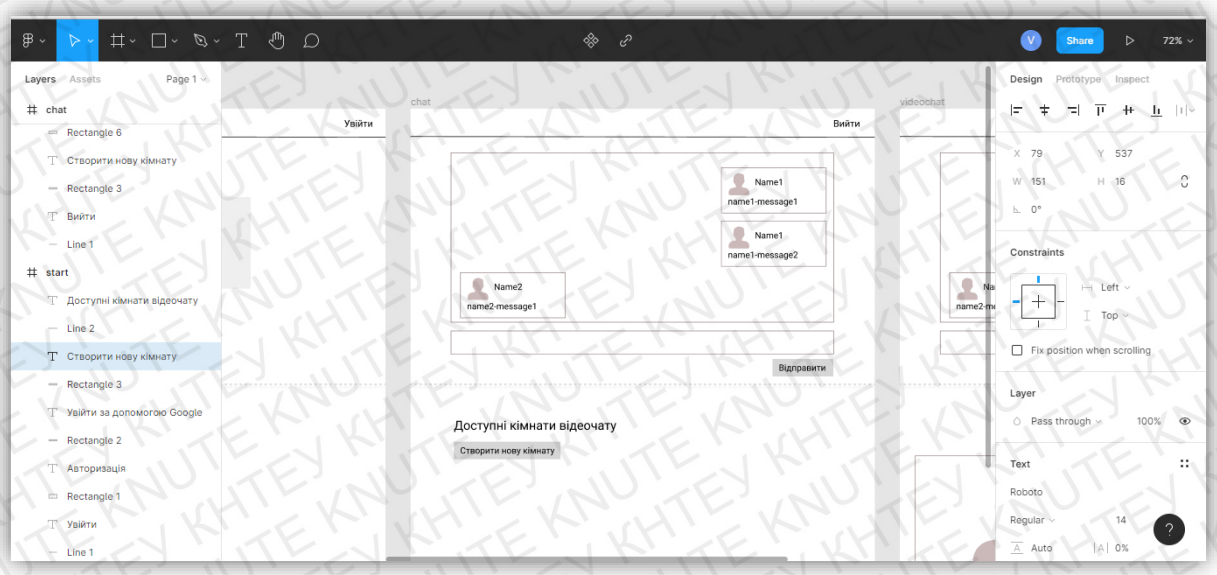


Рис. 2.7. Дизайн real-time чату

Для відеочату після натиснення на кнопку «Створити нову кімнату» відразу на сторінці з’явиться блок із відео-трансляцією першого користувача. Після того, як будь-який інший користувач натисне на кнопку «Приєднатися до кімнати», він буде перенесений на сторінку із першим клієнтом, де тепер розміщатимуться два блоки із відео-трансляцією обох користувачів (Рис. 2.8):

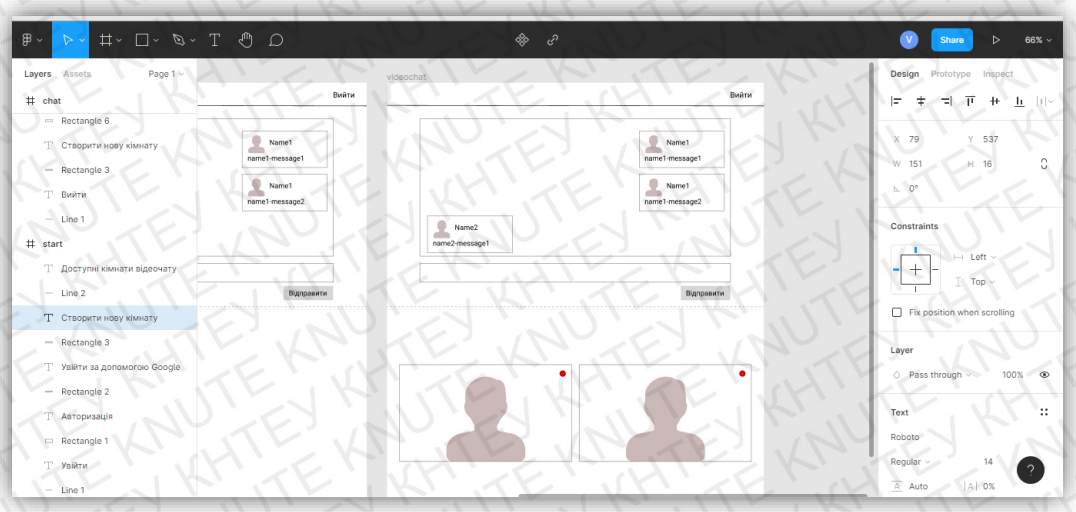


Рис. 2.8. Дизайн відеочату

Розглянуто лише найважливіші компоненти, які знадобляться для правильної роботи додатка. Стилі сторінки та інші елементи будуть додані під час розробки самого веб-додатка.

2.4. Висновки до розділу 2

Отже, в основі створення відеозв’язку лежатиме технологія WebRTC та STAN. Перша відповідає за пряму передачу потокових даних між браузерами в реальному часі, а друга - дозволяє дізнаватися власну публічну адресу для подальшого з’єднання. Текстовий чат працюватиме на базі Firebase Realtime, яка дозволяє зберігати та синхронізувати дані - об’єкти JSON - між користувачами в режимі реального часу. Для підключення до чату необхідна авторизація через Google, що також забезпечує сервіс Firebase. Користувацький інтерфейс веб-додатку, розроблений з використанням Figma, має максимально простий та зрозумілий для клієнта дизайн.

РОЗДІЛ 3
РОЗРОБКА WEB-ОРІЄНТОВАНОЇ СИСТЕМИ КОНСУЛЬТАТИВНОЇ МЕДИЧНОЇ ДОПОМОГИ

3.1. Розробка real-time відеочату

Перейдемо до написання нашого веб-застосування. Для цього будемо використовувати середовище Visual Studio Code [21], оскільки дана програма має зручний і зрозумілий інтерфейс, а також – консоль, яка неодноразово буде необхідна. Після створення нового проекту, через термінал встановимо всі потрібні залежності. По-перше, створимо react-додаток за допомогою команди npm create-react-app [22] (Рис.3.1):

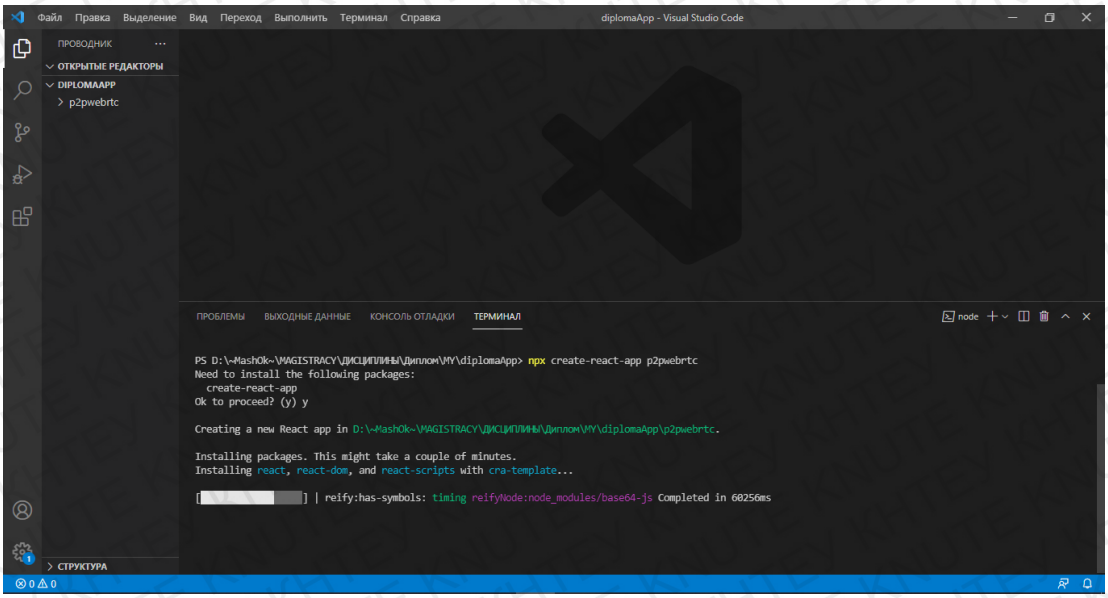


Рис. 3.1. Створення React-додатку

Далі ми маємо встановити всі бібліотеки, які будуть необхідні в процесі розробки real-time відеочату [23]: веб-сервер express для написання обробників для запитів з різними HTTP-методами в різних URL-адресах; веб-сокети - socket.io і веб-сокет на клієнті socket.io-client для встановлення зв'язку між клієнтом і сервером; react-router і react-router-dom для маршрутизації на стороні клієнта (необхідна для синхронізації додатка з URL браузера); бібліотека uuid

для генерування унікальних ідентифікаторів і пакет, який називається freeice для отримання випадкового серверу STUN або TURN WebRTC-додатка [24] (Рис. 3.2).

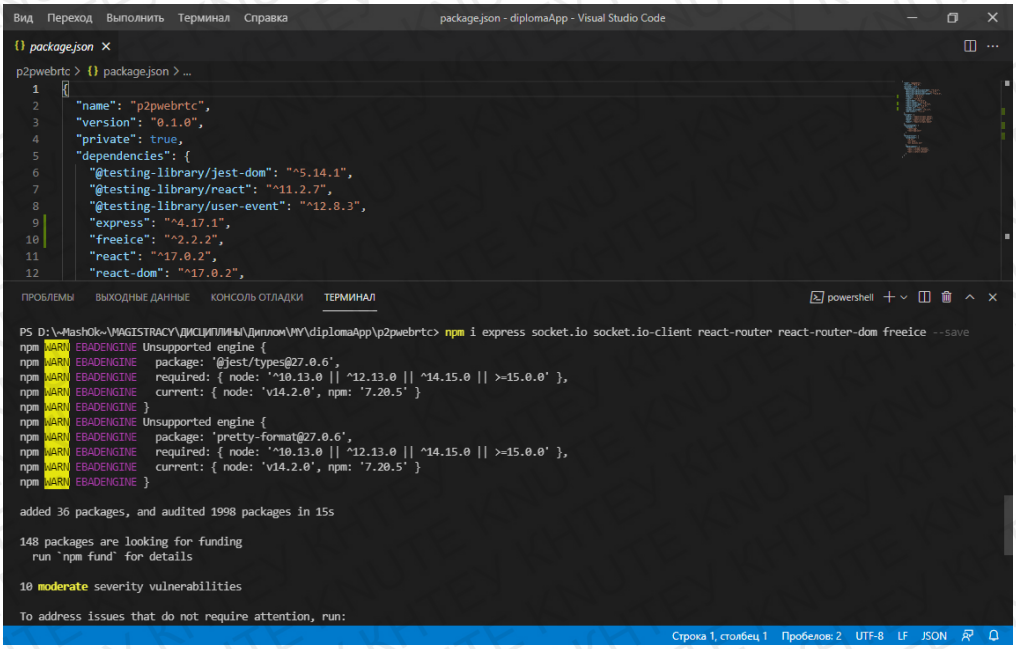


Рис. 3.2. Установлення необхідних пакетів в проекті

Після цього можна приступити до написання коду програми. В файлі App.js (додаток А) налаштуємо роутер нашого застосування - для цього нам знадобиться компонент <BrowserRouter> знадобиться <Switch> [25], оскільки ми матимемо декілька роутів (посилань, «сторінок»). Перший роут – головна сторінка, другий – відповідатиме за всі сторінки, які не існують і третій – сторінка з кімнатою з відеозв'язком. Повна вкладеність даних компонентів:

```
<BrowserRouter>
    <Switch>
        <Route exact path="/room/:id" component={Room} />
        <Route exact path="/" component={Main} />
        <Route component={NotFound404} />
    </Switch>
</BrowserRouter>,
```

де path – посилання на сторінку, component – сама сторінка (окремий js-файл).

Тепер ми маємо створити ці три компоненти (сторінки) і прописати для них відповідний вміст.

Переходимо до створення нашого веб сервера. Для цього ми створимо файл server.j і в ньому додамо потрібні пакети: пакет path, який відповідає за шляхи в додатку (const path = require("path");) [26]; викличимо веб-сервер (express const express = require("express"); const app = express()); далі створимо сам сервер - для цього імпортуємо модуль http і викличемо метод CreateServer і передамо туди сервер express (const server = require("http").createServer(app)) [27]; і також нам потрібно буде підключити сокети і тому ми вкажемо що нам потрібен сокет io та передамо відразу ж туди наш сервер (const io = require("socket.io")(server)). Визначимо порт – або опціональний порт (тобто щоб ми могли його передати) або порт 3001, якщо такого немає. І за допомогою функції listen() вкажемо, що сервер повинен бути запущений на цьому порту (додаток Б).

Тепер створимо підключення на клієнті. Для цього додамо нову директорію та файл index.js. В ньому імпортуємо io з socket.io-client, створимо об’єкт з налаштуваннями для нашого веб-сокету [28]:

```
const options = {
  "force new connection": true,  // повторно використовувати з’єднання
  reconnectionAttempts: "Infinity",  // перепідключення серверу
  timeout: 10000,  // тайм-аут 10 секунд
  transports: ["websocket"]   // з'єднання сокету з сервером за допомогою WebSocket }
```

Далі створимо сам сокет - const socket = io("http://localhost:3001", options). Ми будемо входити на наш http://localhost:3001 з тими описаними налаштуваннями (options) [29]. І експортуємо цей socket - export default socket;

Тепер перевіримо, що ми можемо зв'язати наш клієнт і сервер через websocket-з'єднання. Для цього на сервері (server.js) додамо функцію і в разі правильного підключення просто виведемо текстову інформацію в консоль.

Оскільки ми створюємо відеочат, то у нас будуть кімнати, в яких можуть спілкуватися клієнти. Тому необхідно створити функцію, яка буде повертати нам список всіх кімнат, які існують. Отримати ми їх можемо з io.sockets.adapter

і повернути їх у вигляді масиву:

```
function getClientRooms() {
  const {rooms} = io.sockets.adapter;
  return Array.from(rooms.keys()); }
```

При вході на сторінку користувач буде отримувати список всіх доступних кімнат, аби мати змогу підключитися в будь-яку, тому нам необхідна функція, яка буде відправляти інформацію про створення нової кімнати всім користувачам (сокетам) [30]. Для цього використаємо метод io.emit(), куди ми і передамо нову кімнату:

```
function shareRoomsInfo() {
  io.emit('ACTIONS.SHARE_ROOM', {
rooms: getClientRooms() })  }
```

Наступним кроком варто створити новий файл – actions.js – в якому опишемо всі можливі події, з якими ми будемо працювати як на сервері, так і на клієнті. Це будуть такі події як JOIN - коли ми будемо приєднуватися в кімнату, LEAVE – коли ми будемо залишати кімнату, SHARE_ROOMS – поділитися кімнатами, ADD_PEER/ REMOVE_PEER – коли ми будемо створювати/розривати зв’язки між клієнтами, RELAY_SDP – коли ми будемо передавати наші медіа-дані (відео та аудіо)

В server.js опишемо процес приєднання до кімнати – ACTION.JOIN. По-перше, ми маємо перевірити, чи клієнт приєднаний до даної кімнати чи ні. У випадку підключенні отримаємо попередження - return console.warn('Already joined to ${roomID}'); У іншому випадку – ми отримуємо всіх підключених клієнтів цієї кімнати. Після цього потрібно встановити зв’язки між всіма клієнтами – передати кожному свій особистий ID, а новий клієнт ще повинен кожному дати пропозицію на підключення зв’язку з ними. Код даної процедури наведено нижче:

```
clients.forEach(clientID => {
        io.to(clientID).emit(ACTIONS.ADD_PEER, {   // для всіх клієнтів
          peerID: socket.id,
```

```
            createOffer: false,
        });
    socket.emit(ACTIONS.ADD_PEER, {   // для нового клієнта
            peerID: clientID,
            createOffer: true,  });  });
```

Логіка виходу із кімнати (ACTION. LEAVE) буде мати подібний до входу вигляд– потрібно перевірити всі кімнати та всіх клієнтів, до яких ми приєднані, і кожному відправити запит на від'єднання (тобто відправити свій ID), щоб вони видалили зв'язок з нами і навпаки.

Саме на клієнті логіка підключення/відключення до кімнати розміщена в файлі Main.js (додаток В). Відразу при вході на сторінку ми повинні бачити всі доступні кімнати та кнопки «Створити нову кімнату» і «Приєднатися до кімнати». Перша кнопка буде генерувати унікальний ID для нової кімнати. Це зробиться за допомогою функції v4, яка міститься у вже встановленому пакеті uuid [31]. Отже, при натисненні на кнопку «Створити нову кімнату» ми будемо переходити на сторінку кімнат (Room), а функція згенерує для цієї кімнати ідентифікатор, за допомогою якого в подальшого будуть приєднуватися інші користувачі. Кнопка «Приєднатися до кімнати» відрізняється лише тим, що в ній саме буде використовуватися ID відповідної кімнати замість генерування нової - history.push(`/room/${roomID}`);

Далі напишемо логіку створення кімнати всередині нашої компоненти Room в файлі Room.js (додаток Г). Після того, як ми потрапляємо в цю компоненту (на цю сторінку) нам треба отримати ID поточної кімнати для подальших маніпуляцій – для виклику функції з файлу useWebRTC.js. Вона і буде виконувати головні дії, а саме – зберігати всіх клієнтів, всі зв'язки між клієнтами (const peerConnections = useRef({});), посилання на наш медіаелемент, тобто який буде транслюватися з нашої відеокамери (const localMediaStream = useRef(null);), та медіадані всіх користувачів - const peerMediaElements = useRef({}) [15];

Також в ній описуватимуться дії, які відбуватимуться саме при вході на сторінку (при вході в кімнату) – буде починатися захват екрану (відео та аудіо) та безпосереднє приєдання до цієї кімнати (ACTION.JOIN).

Спочатку іде запит у користувача на дозвіл транслювати відео та аудіо:

```
useEffect(() => {
    async function startCapture() {
        localMediaStream.current=await navigator.mediaDevices.
getUserMedia({
            audio: true,
            video: {
              width: 1200,
              height: 720,
            } });}});
```

У разі згоди відбувається приєднання до кімнати, а в другому випадку отримаємо помилку в консолі. Тільки після цього можна безпосередньо вивести медіадані нового користувача на сторінку і додати рядки, які вимкнуть наше аудіо (щоб ми не чули самих себе), але відео щоб демонструвалося. Функція useWebRTC() (додаток Д), окрім вищезазначеного, перевірятиме, чи приєднаний клієнт до даної кімнати, аби не приєднатися вдруге:

```
const addNewClient = useCallback((newClient, cb) => {
      if (!clients.includes(newClient)) {
        updateClients(list => [...list, newClient], cb);
      }
  }, [clients, updateClients]);
```

Для рендерингу відео, тобто саме для відображення відео елемента на сторінку напишемо наступний код в Room.js:

```
return (
    <div>
        {clients.map((clientID) => {   // перебираються всі клієнти
            return (
```

```
<div key={clientID}>
        <video                          // відображається відео
            autoPlay                    //  автозапуск відео
            playsInline                 //  автоматичний запуск на
        деяких мобільних пристроях
            muted={clientID === LOCAL_VIDEO}  // вимкнути
        звук нашого відео />
    </div> ) })}
</div> );
```

Наступним кроком опишемо логіку виходу з кімнати. Отже, у разі, якщо ми змінюємо кімнату, або просто виходимо з неї – відео та аудіо трансляція всіх користувачів і нас повинні припинятися. Після цього запускається подія – вихід з кімнати (ACTION. LEAVE):

```
return () => {
    localMediaStream.current.getTracks().forEach(track => track.stop());
    socket.emit(ACTION.LEAVE); }
```

Як зазначалось раніше, при вході в кімнату між всіма клієнтами повинні утворитися зв’язки (peers), щоб кожен міг бачити та чути інших і навпаки. Тому варто описати цей процес утворення зв’язків. Для обробки наша функція приймає два значення – ID клієнта та true/false (необхідно чи ні користувачеві давати пропозицію на підключення зв’язку з ним). Спочатку перевіряємо, чи знаходиться ID в наших поточних зв'язках. Якщо так – виводимо помилку, якщо ні – створюємо новий зв’язок за допомогою функції RTCPeerConnection().

Пакет freeice() надає набір адрес безкоштовних stan-серверів, які необхідні для визначення клієнтом своєї зовнішньої IP-адреси, способу трансляції адреси і порту в зовнішній мережі [31].

Після цього треба отримати медіадані від нового користувача і почати їх транслювати. І оскільки ми повинні отримати і аудіо і відео, варто перевірити, чи дозволив користувач транслювати саме ці два елементи. Для цього створимо змінну і зробимо перевірку:

```
let tracksNumber = 0;
peerConnections.current[peerID].ontrack = ({ streams: [remoteStream] }) => {
        tracksNumber++;    // додається +1, коли отримується дозвіл на
відео та аудіо
        if (tracksNumber === 2) {
          addNewClient(peerID, () => {
            peerMediaElements.current[peerID].srcObject= remoteStream;
// вивід медіа-елементів}); } }
```

Тільки після цих маніпуляцій слідує відгалудження: якщо користувач дає пропозицію на підключення зв'язку (if (createOffer) - тобто це новий клієнт), то нам треба створити цю пропозицію - const offer = await peerConnections.current[peerID]. createOffer(); і передати цей offer разом зі всіма нашими SDP-даними (медіа-алементами) на серверну обробку події ACTIONS.RELAY_SDP:

```
socket.emit(ACTIONS.RELAY_SDP, {
        peerID,
        sessionDescription: offer, });
```

В файлі server.js опишемо серверну логіку подій RELAY_ICE і RELAY_SDP. Коли нам приходять SDP-дані, ми отримуємо peerID (тобто ID клієнта, його зв'язок) та offer на підключення зв'язку [31]. Після цього буде відбуватися подія SESSION_DESCRIPTION, куди ми і передамо ті два параметри.

На клієнті подія SESSION_DESCRIPTION буде описана таким чином – встановлюємо setRemoteDescription нашому peerID, де ми зможемо отримувати як offer так і answer, а потім перевіряємо, який тип має наш sessionDescription/ remoteDescription. Якщо "offer", то ми створюємо відповідь і пересилаємо її назад на подію RELAY_SDP.

Залишилося описати на клієнті подію видалення зв'язків (REMOVE_PEER) між клієнтами при виході з кімнати. В функції ми перевіряємо, чи маємо ми зв'язок з конкретним користувачем в нашому списку всіх зв'язків. Якщо маємо,

то потрібно закрити та видалити із цього списку. А також необхідно видалити всі медіа-елементи того, хто хоче розірвати зв'язок. Після цього викликаємо метод для оновлення всіх наявних клієнтів:

```
useEffect(() => {
    const handleRemovePeer = ({ peerID }) => {
      if (peerConnections.current[peerID]) {
        peerConnections.current[peerID].close();
      }
      delete peerConnections.current[peerID];   // видалення зв'язку
      delete peerMediaElements.current[peerID];  // видалення медіа
      updateClients(list => list.filter(c => c !== peerID));
    };
    socket.on(ACTIONS.REMOVE_PEER, handleRemovePeer);
  }, []);
```

Для того аби відео-трансляції всіх користувачів зручно розміщувалися на сторінці (щоб було видно абсолютно всіх клієнтів), профільтруємо їх – виставимо по два в ряд. Для цього в функції створимо масив з кількістю користувачів і розділимо їх на пари масивів. Після цього йде перевірка: якщо ми маємо парну кількість – то попарно їх і запишемо в загальний масив:

```
const pairs = Array.from({ length: clientsNumber }).reduce((acc, next, index,
arr) => {
    if (index % 2 === 0) {
      acc.push(arr.slice(index, index + 2)); }
    return acc;
  }, []);
```

У разі, якщо в кінці залишається один клієнт (без пари), то даного користувача розмістимо його контейнер на всю ширину сторінки, а висоту розрахуємо за формулою height = `${100 / rowsNumber}%`, де rowsNumber – кількість пар масивів:

```
return pairs.map((row, index, arr) => {
```

```
if (index === arr.length - 1 && row.length === 1) {
        return [{
          width: '100%',
          height,
}]; }
      return row.map(() => ({       // для пар елементів ширина = ½ сторінки
        width: '50%',
        height,  }));
  }).flat();
```

Мінімально кастомізуємо стилями наш ряд, де будуть розміщені відео-трансляції. Задамо гнучкий тип відображення, вирівняємо по центру весь вміст, дозволимо переносити блоки на новий ряд та встановимо висоту рівною висоті екрану користувача [32] за допомогою стилів.

3.2. Розробка real-time чату

Для написання real-time чату першим кроком створимо декілька окремих компонентів (файлів .js). Додамо Navbar.js – навігаційна панель (меню) зверху сторінки, Login.js – сторінка, де ми будемо авторизуватися через Google і Chat.js – компонент для сторінки безпосередньо з чатом.

За допомогою терміналу встановимо всі додаткові необхідні для розробки чату залежності: firebase – для взаємодії з хмарною базою даних, react-firebase-hooks - модуль, який надає функції для зручної взаємодії з авторизацією, з базою даних [16] (Рис. 3.3):

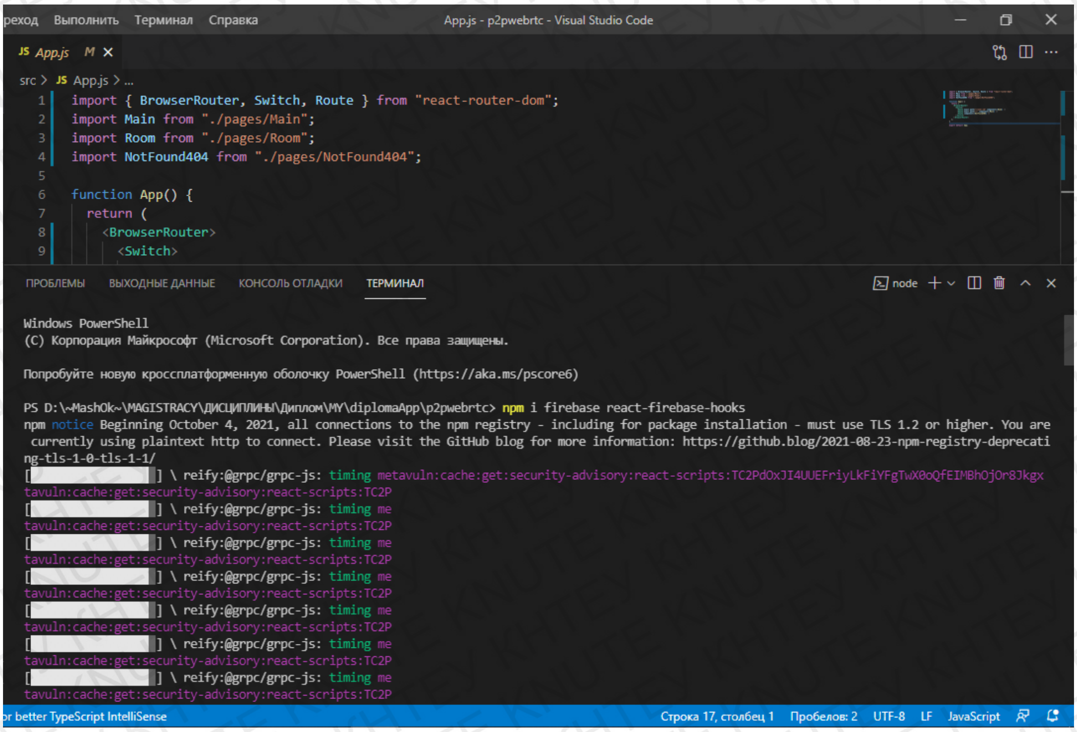


Рис. 3.3. Установлення додаткових пакетів для розробки чату

По-перше створимо файл routes.js, де будуть знаходитися посилання на наші сторінки з авторизацією та з чатом. Але обов’язково ці посилання повинні перебувати в різних константах в залежності від того, чи увійшов користувач в свій обліковий запис. Тобто для них буде відображатися сторінка Chat, а для всіх інших – Login:

```
export const publicRoutes = [ {
    path: LOGIN_ROUTE,
    Component: Login } ]
export const privateRoutes = [ {
    path: CHAT_ROUTE,
    Component: Chat } ]
```

Новостворений файл AppRouter.js фільтруватиме дані щодо авторизації користувача і в залежності від його значення буде перенаправляти клієнта на відповідну сторінку Chat, або Login. Для цього тимчасово додамо константу user, яка буде приймати значення true або false:

```
const AppRouter = () => {
  const user = false;
  return user ?
```

```
( <Switch>
        {privateRoutes.map(({ path, Component }) =>
          <Route key={path} path={path} component={Component}
exact={true} /> )}
        <Redirect to={CHAT_ROUTE} />
      </Switch>
    ) : (
      <Switch>
        {privateRoutes.map(({ path, Component }) =>
          <Route key={path} path={path} component={Component}
exact={true} /> )}
        <Redirect to={LOGIN_ROUTE} />
      </Switch> ); };
```

Оскільки в нашому додатку буде навігаційна панель (Navbar.js), додамо її верстку за допомогою графічного фреймворку Material-UI. Для початку потрібно його встановити, використовуючи команду npm install @material-ui/core, яка прописана на офіційному сайті [33]. Там знаходимо необхідну панель, копіюємо верстку та видаляємо зайве (Рис. 3.4):

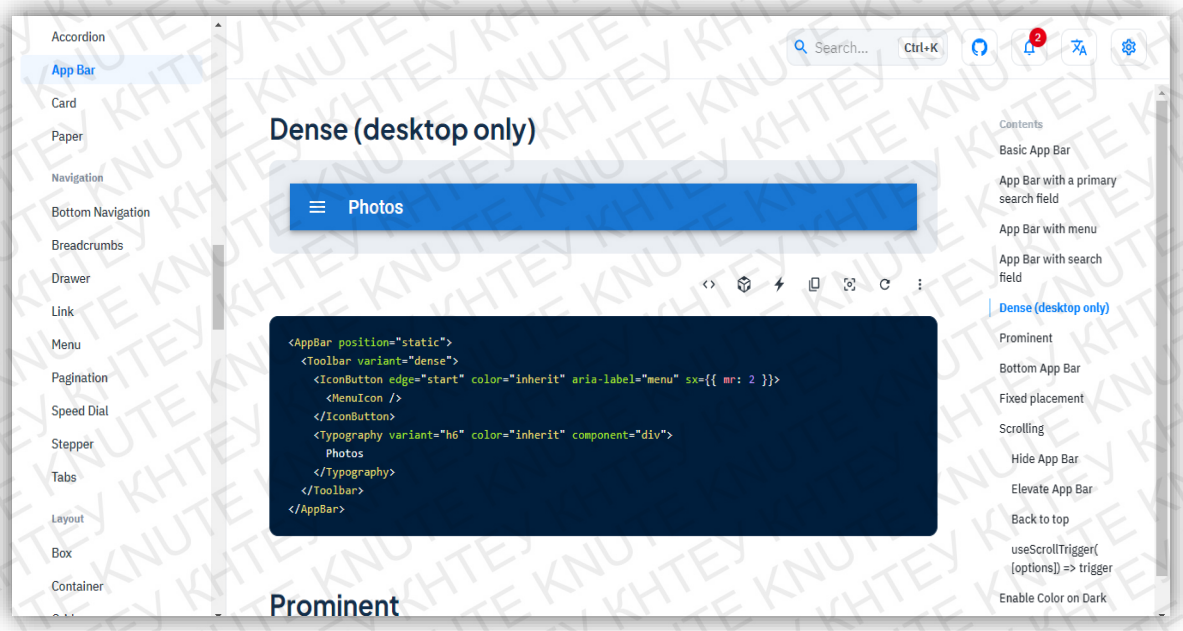


Рис. 3.4. Під'єднання навігаційної панелі фреймворку Material-UI

Поки що в нашій Navbar (додаток Е) будуть міститися лише дві кнопки – «Увійти» в обліковий запис та «Вийти» з нього. Знову йде розгалудження: якщо користувач не авторизований, то в нього буде відображатися перша кнопка, якщо навпаки – друга кнопка, яка буде перенаправляти на сторінку Login:

```
{user ?
  <Button variant={"outlined"}>
    Log In</Button> :
      <NavLink to={LOGIN_ROUTE}>
        <Button variant={"outlined"}>Log Out</Button>
      </NavLink> }
```

Після цього можна переходити до Firebase. Заходимо на його офіційний сайт під власним Google-акаунтом і створюємо новий проект [16]. Далі для підключення сервісу Firebase до нашого додатку переходимо на відповідну вкладку і слідуємо вказівкам – копіюємо код (Рис. 3.5) і вставляємо в наш файл index.js (додаток Ж).

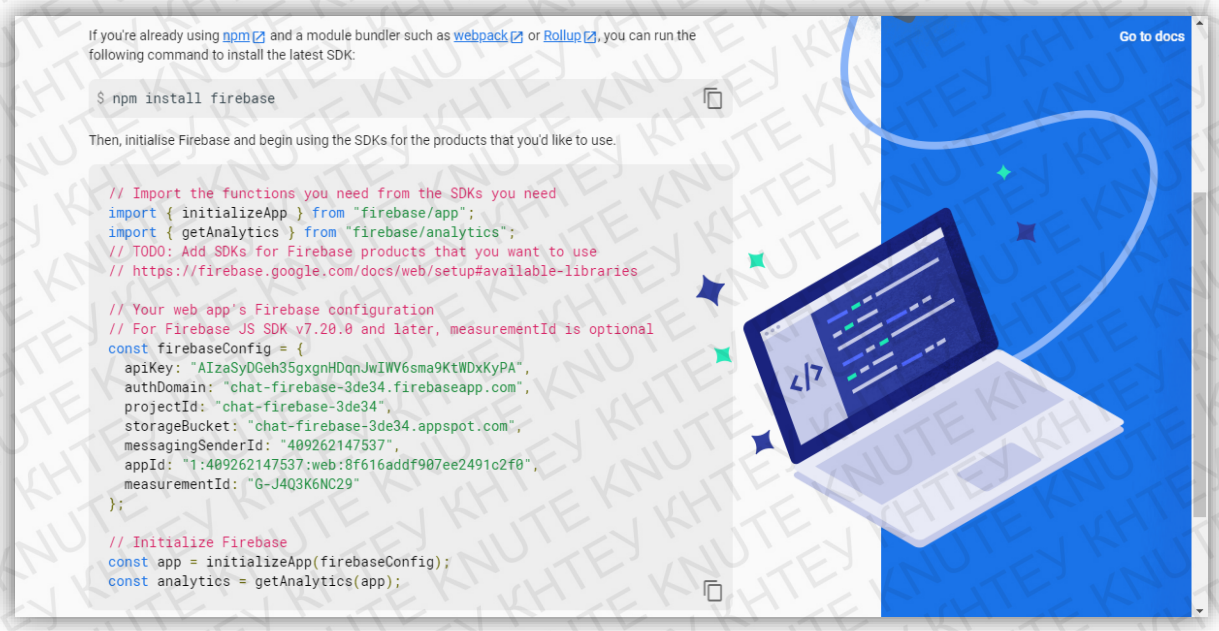


Рис. 3.5. Підключення Firebase до додатку

І для того, аби мати можливість авторизуватися через Google, на сайті вмикаємо даний функціонал. В index.js створимо об’єкт, за допомогою якого ми будемо заходити в свій обліковий запис - const auth = firebase.auth() (Рис. 3.6):

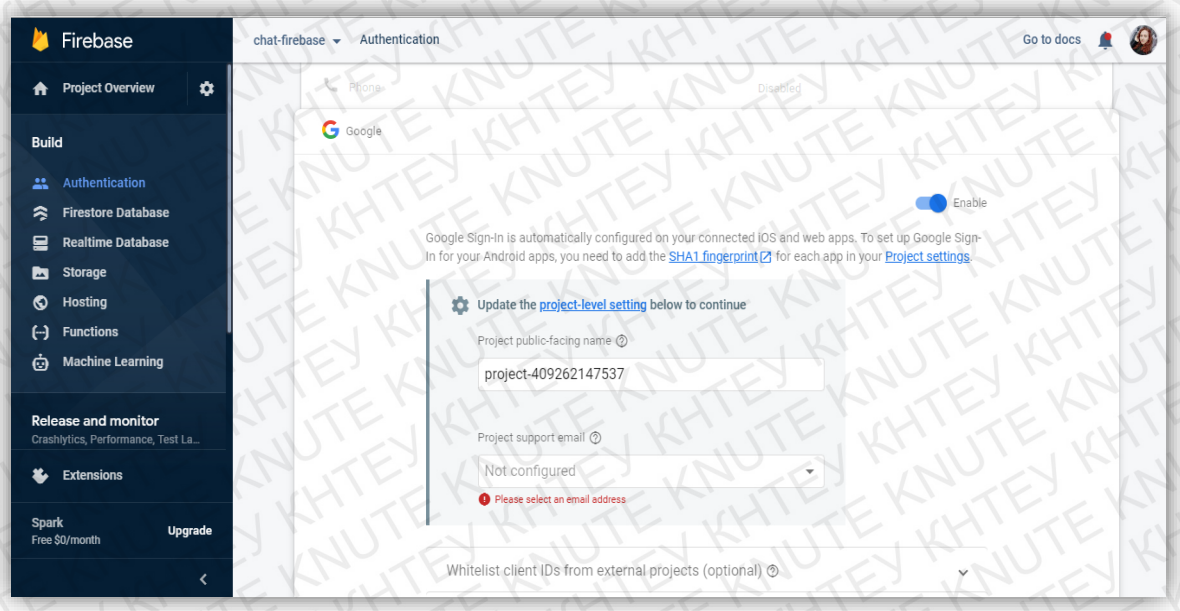


Рис. 3.6. Встановлення можливості авторизуватися через Google

На сторінці Login.js у нас саме і буде знаходитися ця кнопка – «Увійти за допомогою Google», при натисненні на яку викликатиметься функція login - <Button onClick={login} variant={"outlined"}>Login with Google</Button>. Опишемо функцію login (додаток И):

```
const login = async () => {
    const provider = new firebase.auth.GoogleAuthProvider();
        // викликаємо провайдер авторизації
    const { user } = await auth.signInWithPopup(provider);
        // отримуємо користувача після авторизації }
```

Після підключення цього можливості «Увійти» через Google змінимо константу user на - const [user] = useAuthState(auth); Тепер за допомогою функції useAuthState() ми будемо отримувати актуальну інформацію про те, чи авторизований наш користувач (будуть його дані) чи ні (null).

На сторінці чату (додаток К) у нас буде розміщуватися блок зі всіма повідомленнями, а також поле для вводу цих повідомлень з кнопкою відправки:

```
<TextField
    fullWidth  // ширина на весь екран
    rowsMax={2}    // кількість рядків – 2
```

```
        variant={"outlined"}   //  рамка
        value={value}          //  значення з поля
        onChange={e => setValue(e.target.value)}
                //  функція при зміні значення поля    />
    <Button onClick={sendMessage} variant={"outlined"}>Send</Button>
```

Для управління станом поля для вводу (при зміні тексту, який знаходиться всередині) створимо константу const [value, setValue] = useState(""). Вона буде реагувати на зміну вмісту поля, яка буде відбуватися завдяки функції setValue(). Кнопка відправки повідомлення запускає функцію sendMessage при настисненні, яка додає нашу інформацію в базу даних. Реалізуємо дану функцію:

```
  const sendMessage = async () => {
        firestore.collection("messages").add({
          uvid: user.id,                          // ID користувача
          displayName: user.displayName,          // ім’я
          photoURL: user.photoURL,                // посилання на фотографію
          text: value,                            // повідомлення
          createdAt: firebase.firestore.Firestore.serverTimestamp()
          //  дата відправки, яка буде отримуватися із серверу firebase })
       setValue("");  // очистка поля для вводу після відправки }
```

Отримувати повідомлення ми будемо за допомогою функції useCollectionData() в константу messages - const [messages] = useCollectionData(firestore.collection("messages").orderBy("createdBy")). Оскільки це NoSQL база даних, вона складається з колецій. В нашому випадку вона називається messages. OrderBy("createdBy") додасть сортування по полю створення повідомлення [16].

Тепер на сайті Firebase створюємо базу даних, в якій і буде знаходитися колекція messages з нашими повідомленнями та іншою інформацією (Рис. 3.7):

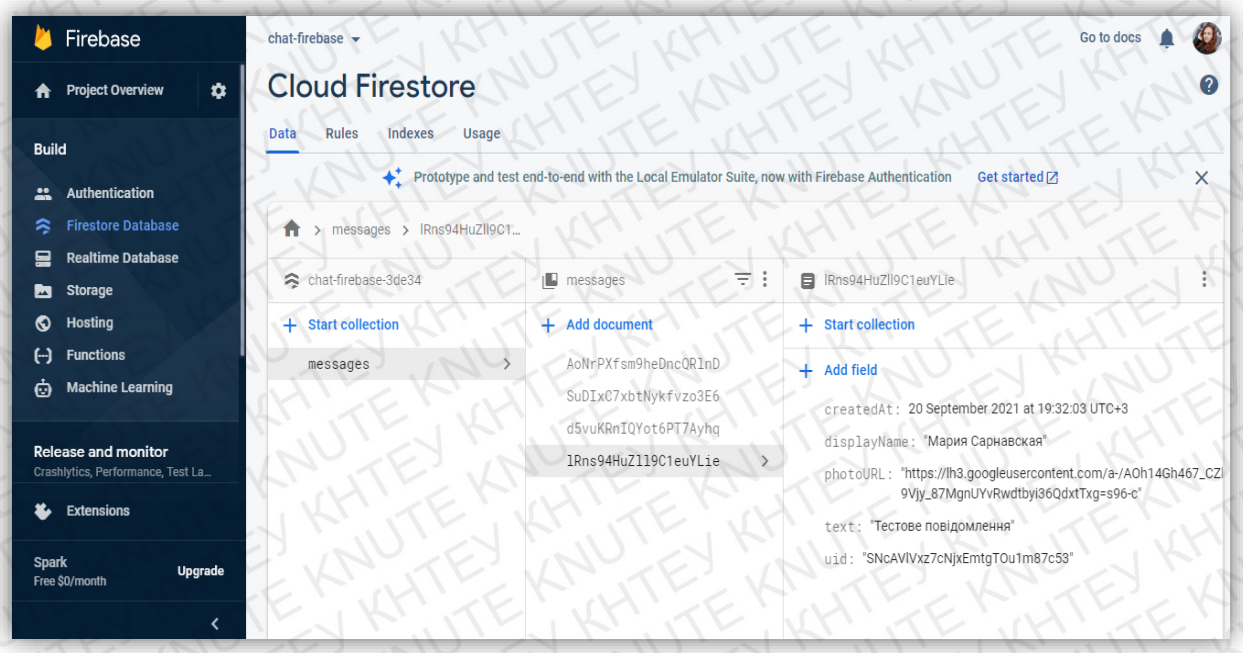


Рис. 3.7. Колекція повідомлень проекту в Cloud Firestore

Для відображення вмісту бази даних на сторінці використаємо функцію map(), що перебере всі наші повідомлення [24]. Для кожного ми будемо створювати блок, в якому розмістимо фотографію користувача, його ім’я і саме повідомлення:

```
{messages.map(message =>
        <div>
          <Grid container>
            <Avatar src={message.photoURL} />
            <div>{message.displayName}</div>
          </Grid>
          <div>{message.text}</div>
        </div>
      )}
```

3.4. Стилізація web-додатку

Функціонал веб-додатку розроблено, тепер варто звернути увагу на зовнішній вигляд програми. На даний момент додаток має наближений до макету стиль, покращимо його за допомогою CSS та декількох нових компонентів.

По-перше після навігаційної панелі додамо банер на всю ширину екрану (width: 100vw) з відповідним зображенням з теми медицини, яке ми імпортуємо з директорії в змінну BannerImg (import BannerImg from "../imgs/young-handsome-physician-medical-robe-with-stethoscope.jpg"), а потім встановлюємо як значення атрибуту src ( ). На ньому розмістимо заголовок і абзац з текстом. Все це буде знаходитися в новому компоненті Banner.js (Рис. 3.8):

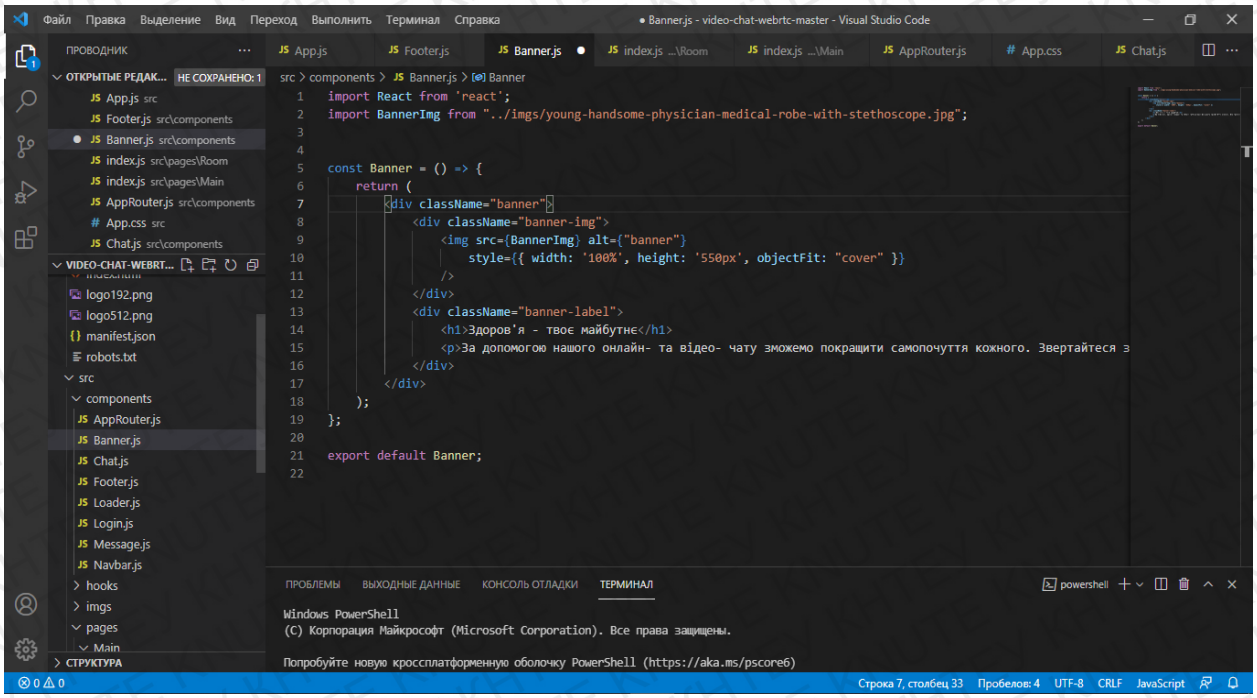


Рис. 3.8. Створення нового компоненту Banner.js

Перед онлайн-чатом та відеочатом додамо заголовки, розмістимо їх посередині. Приймемося стилізувати онлайн-чат. Зараз сам чат і всі повідомлення прикрашені лише рамкою. Фоном чату виставимо картинку з вирівнювання посередині, масштабуємо зображення зі збереженням пропорцій так, щоб його висота дорівнювала висоті блоку (cover) та встановимо внутрішні відступи (до повідомлень) 13 пікселів.

```
div.chat-block {
  background: url(../src/imgs/doctor-working-with-modern-touch-screen.jpg);
  background-size: cover;
  background-position: center;
  padding: 13px;
}
```

Для окремних повідомлень задамо такі властивості – заокруглення рамки, змінимо колір рамки на синій і встановимо фоном майже прозорий білий колір. Для аватару (зображення користувача, яке завантажеться із Google акаунту) пропишемо таку ж рамку, а також задамо відступ праворуч, аби ім'я клієнта від'єдналося від його зображення [32]:

```
div.chat-one-message {
  border-radius: 4px;
  border: 1px solid #1f3049;
  background-color: #fafafa6e;
}
```

Для секції з відеочатом ми також додамо фонове зображення, розмістимо весь текст посередині, стилізуємо списки з ідентифікаторами існуючих кімнат та кнопки «Створити нову кімнату» і «Приєднатися до кімнати» [32]:

```
div.main-video button {
    padding: 8px 13px;          // внутрішні відступи
    background-color: #1f3049;  // синім колір кнопки
    color: white;               // білий текст
    border: 1px solid #1f3049;  // рамка синього кольору
    border-radius: 3px;         // заокруглення рамки
    letter-spacing: 0.5px;      // відступи між літерами
    margin-top: 20px;           // верхній відступ
    font-size: 14px;            // розмір шрифту
    text-transform: uppercase;  // верхній регістр
```

transition: background-color 250ms cubic-bezier(0.4, 0, 0.2, 1) 0ms,box-shadow 250ms cubic-bezier(0.4, 0, 0.2, 1) 0ms, border 250ms cubic-bezier(0.4, 0, 0.2, 1) 0ms; // анімація переходу від одного стану в інший

}

На сторінці з відеотрансляціями не вистачає кнопки «Вийти з кімнати», тому перейдемо в компонент Room.js і після нашої функції з перебором всіх користувачів (для виводу відео на сторінку) додамо кнопку. Стилі вона матиме такі ж, як і інші кнопки. При натисненні буде відбуватися перехід на початкову сторінку.

```
<div className="room-exit-button">
    <button onClick={() => { history.push('/'); }}>Вийти з кімнати</button>
</div>
```

Створимо ще один компонент Footer.js, який буде повертати секцію з двома блоками: в одному розмістимо пізнавальну інформацію про телемедицину, а в іншому – назву додатку (Рис. 3.9):

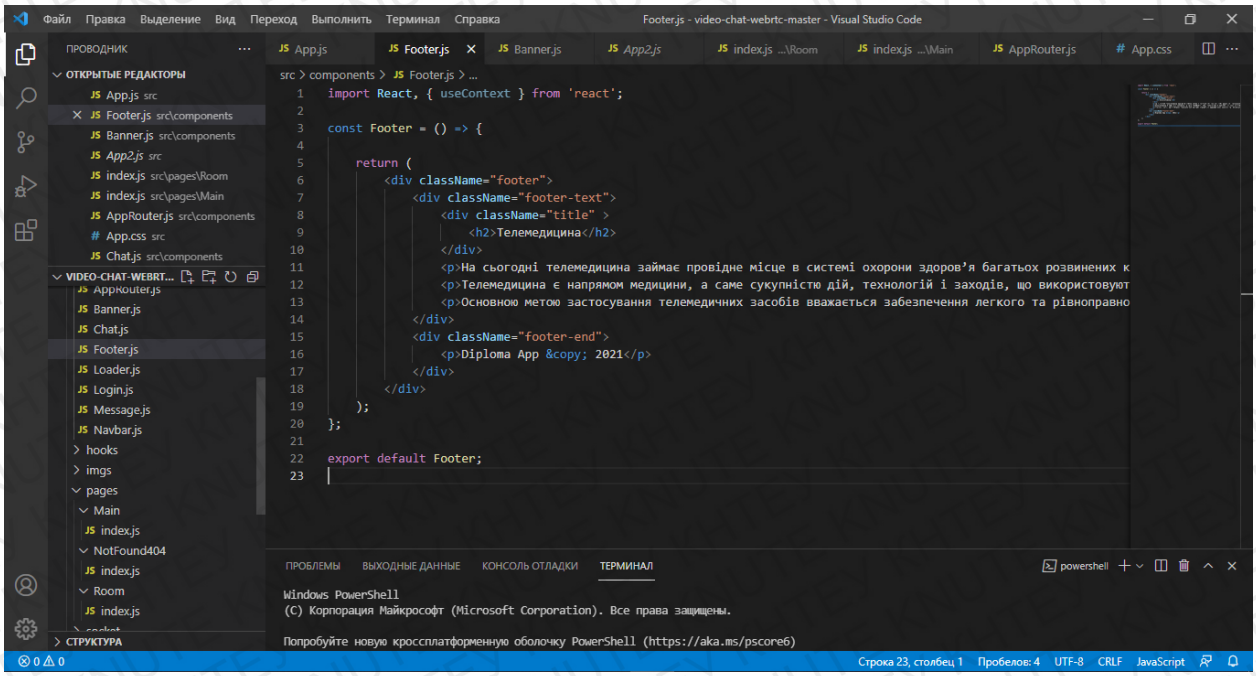


Рис. 3.9. Створення нового компоненту Footer.js

Для відображення цього компоненту необхідно його імпортувати в головний js-файл додатку та залишити після наших компонентів з головною сторінкою відеочату та з самими кімнатами.

```
import Footer from './components/Footer';
function App() {
 return (
        <BrowserRouter>
         <Switch>
       <Route exact path='/room/:id' component={Room} />
            <Route exact path='/' component={Main} />
            <Redirect to={"/"} />
          </Switch>
        <Footer />
        </BrowserRouter>
 );
}
export default App;
```

3.4. Висновки до розділу 3

Отже, логічне та послідовне поєднання описаних функцій та компонент фреймворку ReactJS у клієнтську та серверну частини, дозволили за допомогою технології WebRTC створити онлайн зв'язок з можливістю приймати та передати відео- та аудіо-дані між клієнтами. Даний відеочат у взаємодії з текстовим чатом, розробленим з використанням Firebase Realtime, створюють повноцінну web-орієнтовану систему консультативної медичної допомоги. Дизайн готового додатку був покращений завдяки додаванню нових компонент з використанням стилів CSS.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Проведено аналіз поняття веб-додатку та моделей їх розробки. З'ясувалося, що модель «один веб-сервер (з базою даних)» є найоптимальнішим варіантом для невеликих додатків, які є тестовим проектом або приватною практикою, оскільки вона є найпростішою та найшвидшою для розробки.
2. Було вивчено існуючі способи створення веб-додатку та їх особливості. SPA - це додаток, який працює всередині браузера і не вимагає перезавантаження сторінок під час використання. З іншого боку, MPA вимагає перезавантаження сторінки щоразу, коли змінюється її вміст. MPA це найкращий варіант для великих компаній, а для даного проєкту більш раціональним є вибір способу розробки SPA, який є швидким у завантаженні, що має оцінитися користувачем.
3. В якості мови написання додатку було обрано JavaScript, який порівняно з іншими мовами є достатньо простим і гнучким. Також перевагою JavaScript є те що він замість того, щоб виконуватися на сервері веб-сайту, виконується прямо на пристрої користувача, що зводить до мінімуму запити сервера та покращує роботу користувача. Серед популярних фреймворків (Angular, React та Vue) було обрано ReactJS, який є достатньо зрозумілим, простим та добре підходить для створення SPA-програм.
4. В процесі виконання проєкту було створено декілька схем, які повністю відображають логіку роботи компонентів додатку. Було з'ясовано, що задля встановлення прямого p2p-з'єднання між клієнтами знадобиться дізнатися публічну адресу, захопити медіа-контент (відео, аудіо) і відправити ці дані через сервер сигналізації іншому користувачу – offer, а інший у відповідь надсилає answer. Стосовно текстового real-time чату було спроектовано

логіку відправки/отримання повідомлень з бази даних Firebase, використовуючи відповідні технології та функції.

5. Розроблена система повністю задовольняє всі поставленні завдання: користувач має змогу в онлайн-режимі отримувати консультації лікаря; є можливість спілкування в текстовому варіанті; наявний відеозв'язок між клієнтами. Окрім очевидного факту того, що такий вид консультацій не наражає інших на небезпеку заразитися можливим захворюванням (особливо зараз, під час пандемії коронавірусу) даний додаток - гідний варіант для населення, що живе у віддалених сільських територіях з малою доступністю медичних послуг і браком медичних спеціалістів.

Незважаючи на те, що створена система є повною, працездатною та виконує головну необхідну задачу – надання консультацій онлайн – задля її покращення та доповнення можна сформулювати наступні пропозиції:

1. Створити можливість реєструватися на сайті, аби користувач мав особистий кабінет;
2. Додати функцію запису на особистий онлайн-прийом до лікаря (дата, час) – ці дані та висновки/поради лікаря мають відображатися в особистому кабінеті (передбачається, що дана інформація, а також дані про клієнта будуть зберігатися в базі даних);
3. Розробити секцію для публічних питань (форум), на які відповідатиме лікар.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бондаренко В. Мобільні застосунки як засіб комунікації в суспільстві знань: освітній аспект / В. Бондаренко // Наук. пр. Нац. б-ки України ім. В. І. Вернадського : зб. наук. пр. / НАН України, Нац. б-ка України ім. В. І. Вернадського, Асоц. б-к України. – Київ, 2017. – Вип. 48. – С. 576–590.
2. Шевкун, Б. О. Програмне забезпечення браузера для перегляду веб-сайтів / Б. О. Шевкун // Сучасні інформаційні технології та програмне забезпечення комп'ютерних систем : Всеукр. наук.-практ. конф. студентів та магістрантів / М-во освіти і науки, молоді та спорту України, Кіровоград. нац. техн. ун-т. — Кіровоград : КОД, 2015. — С. 132.
3. Розробка WEB – додатків для бізнесу [Електронний ресурс]. – URL: https://tqm.com.ua/ua/rozrobka-veb-dodatkiv-servisiv-web-application-development/rozrobka-web-dodatkiv-dlia-bizniesu
4. Захарченко А. П. Інтернет-медіа: навч. посіб. до курсу "Підтримка сайту" / Артем Захарченко ; Київ. нац. ун-т ім. Тараса Шевченка, Ін-т журналістики. - Київ : Марченко [вид.], 2015. - 141 с. : іл. - Бібліогр.: с. 139-141.
5. Інтернет-технології та ресурси : метод. вказ. до лаб. робіт / уклад. К. М. Марченко ; Центральноукраїн. нац. техн. ун-т, каф. кібербезпеки та прог. забезпеч. – Кропивницький : ЦНТУ, 2017. – 45 с.
6. Web-технології та Web-дизайн: лабораторний практ. для студ. напряму підготов. 6.050101 «Комп'ютерні науки» денної та заочної форми навч. / уклад. : Ю. П. Чаплінський, О. В. Субботіна ; Нац. ун-т харч. технол. — К. : НУХТ, 2015. — 137 с.
7. Зубенко В. В. Програмування: навчальний посібник (гриф МОН України) / В. В. Зубенко, Л. Л. Омельчук. — К. : ВПЦ «Київський університет», 2015. — 623 с.

8. Майкл Міковскі, Джош Пауелл. Розробка односторінкових веб-додатків. Single Page Web Applications: JavaScript End-to-end. - ДМК Пресс, 2014. - 512 с. - ISBN 978-5-457-83457-6.
9. Багатосторінкові та односторінкові додатки, їх переваги та недоліки [Електронний ресурс]. – URL: https://dou.ua/forums/topic/25444/
10. 2020 Developer Survey [Електронний ресурс]. – URL: https://insights.stackoverflow.com/survey/2020#technology
11. Нікітченко М. С. Теоретичні основи програмування: навчальний посібник / М.С Нікітченко — Ніжин: Видавництво НДУ імені Миколи Гоголя, 2018. — 121с.
12. What Are the Best Languages for Web Application Development in 2021? [Електронний ресурс]. – URL: https://steelkiwi.com/blog/best-languages-for-web-application-development/
13. Що таке WebRTC і чому розробники і компанії люблять цю мережеву технологію [Електронний ресурс]. – URL: https://senior.ua/articles/chto-takoe-webrtc-i-pochemu-razrabotchiki-i-kompanii-lyubyat-etu-setevuyu-tehnologiyu
14. Смірнов В.В. Технологія проектування програмних систем. Лекції / В.В. Смірнов, Н.В. Смірнова.- Кіровоград: КНТУ, 2015.-95 с.
15. WebRTC support overview [Електронний ресурс]. – URL: https://webrtc.org/support/overview
16. Firebase Documentation [Електронний ресурс]. – URL: https://firebase.google.com/docs
17. Firebase Realtime Database [Електронний ресурс]. – URL: https://ru.bmstu.wiki/Firebase_Realtime_Database#.D0.98.D1.81.D1.82.D0.BE.D1.87.D0.BD.D0.B8.D0.BA.D0.B8
18. Мартін Фаулер, Прамодкумар Дж. Садаладж. NoSQL: нова методологія розробки нереляційних баз даних - NoSQL Distilled. - М .: «Вільямс», 2016. - 192 с.

19. Figma Documentation [Електронний ресурс]. – URL: https://help.figma.com/hc/en-us
20. Хвостенко Т. М., Веліксар Д. С. Figma - перспективний інструмент сучасного веб-дизайнера // Вісник освітнього консорціуму. Інформаційні технології. - 2019. - № 2 (14). - С. 7-10.
21. Documentation for Visual Studio Code [Електронний ресурс]. – URL: https://code.visualstudio.com/docs
22. Npm Documentation [Електронний ресурс]. – URL: https://docs.npmjs.com/packages-and-modules
23. Комп'ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2014. — 256 с.
24. React Documentation [Електронний ресурс]. – URL: https://devdocs.io/react/
25. React Router: Declarative Routing for React.js [Електронний ресурс]. – URL: https://reactrouter.com/web/guides/quick-start
26. Фаулер, Мартін. Рефакторинг коду на JavaScript: поліпшення проекту існуючого коду, 2-е вид. - М .: «Діалектика», 2019. - 464 с.
27. Express. Керівництво [Електронний ресурс]. – URL: https://expressjs.com/ru/guide/routing.html
28. Socket.IO. Introduction [Електронний ресурс]. – URL: https://socket.io/docs/v4/
29. Emmit Scott. Spa Design and Architecture: Understanding Single Page Web Applications. — Manning Publications Company, 2015. – 156 с.
30. Майкл Міковскі, Джош Пауелл / Розробка односторінкових веб-додатків / Single Page Web Applications: JavaScript. - ДМК Пресс, 2014. - 512 с.
31. Npm Packages [Електронний ресурс]. – URL: https://www.npmjs.com/package/uuid
32. CSS documentation [Електронний ресурс]. – URL: https://devdocs.io/css/
33. Material-UI documentation [Електронний ресурс]. – URL: https://mui.com/getting-started/usage/

ТЕХНІЧНЕ ЗАВДАННЯ

1. Загальні відомості

1.1. Найменування системи

Технічне завдання на створення web-орієнтованої системи консультативної медичної допомоги на основі розробки real-time чату для текстового спілкування з лікарем, а також відеочату для можливості отримувати професійні поради в голосовому та відео-режимі.

1.1.1. Повне найменування системи: Web-орієнтована система консультативної медичної допомоги.

1.1.2. Скорочене найменування системи: СКМД, система, додаток.

1.2. Планові терміни початку та закінчення робіт

Дата початку робіт: 01 серпня 2021 року

Дата закінчення робіт: 18 жовтня 2021 року

1.3. Порядок оформлення і пред'явлення результатів робіт

Роботи зі створення СКМД здаються Розробником поетапно відповідно до календарного плану Проєкту. Після закінчення всіх етапів Розробник здає готову роботу на перевірку в заданий термін.

1.4. Головний бенефіціар та потенційні користувачі системи

Головним бенефіціаром та потенційними користувачами СКМД є:

- Міністерство охорони здоров'я;
- професійні сертифіковані лікарі;
- громадяни України.

2. Мета та призначення створення системи

2.1. Призначення системи

Проєкт має на меті спростити процес отримання консультативної

лікарської допомоги через провадження дистанційної системи. Для цього СКМД має бути спроможна виконувати такі задачі:

- користувач повинен мати змогу в онлайн-режимі отримувати консультації лікаря;
- можливість спілкування в текстовому варіанті;
- наявність відеозв'язку між клієнтами.

2.2. Мета створення системи

СКМД створюється з метою:

- забезпечення населення зручним та зрозумілим Web-додатком для отримання лікарської допомоги дистанційно;
- забезпечення окремої ланки громадян, які живуть у віддалених сільських територіях з малою доступністю медичних послуг і браком медичних спеціалістів, можливістю отримувати професійні лікарські поради;
- усунення додаткової ймовірності перехопити можливе захворювання в будь-яких місцях скупчення людей (лікарні).

3. Вимоги до системи

3.1. Вимоги до системи в цілому

3.1.1. Вимоги до структури та функціонування системи, перелік підсистем

Система має складатися з двох основних підсистем:

- підсистема «real-time чат», яка призначена для текстового обміну інформацією між клієнтами;
- підсистема «real-time відеочат», яка призначена для спілкування з використанням мікрофону та камери.

3.1.1.1. Вимоги до способів і засобів інформаційного обміну між компонентами системи

В основі створення відеозв'язку повинна лежати технологія WebRTC, яка відповідає за пряму передачу потокових даних між браузерами в реальному часі.

А для з’ясування власної публічної адреси для подальшого з’єднання має використовуватися протокол STAN. Текстовий real-tine чат працюватиме на базі Firebase Realtime, яка дозволяє зберігати та синхронізувати дані (повідомлення) - об’єкти JSON - між користувачами в режимі реального часу.

3.1.1.2. Вимоги до режимів функціонування системи

Система має працювати в одному режимі, в якій підсистеми «real-time чат» та «real-time відеочат» повинні функціонувати незалежно одна від одної. Користувач повинен мати змогу приєднатися до будь-якого виду з чатів окремо, або разом – в залежності від необхідності.

3.1.1.3. Вимоги до діагностування системи

Діагностування системи має відбуватися вручну, шляхом перевірки з’єднання між користувачами. Кожна підсистема має працювати справно та виконувати свої функції. У разі помилки – завершення роботи системи, пошук несправностей та виправлення.

3.1.1.4. Вимоги до режимів управління системою

Підсистеми «real-time чат» повинна працювати (відкриється можливістю обміну повідомленнями) після виконання наступних кроків:

- натиснення на кнопку «Увійти за допомогою Google»;
- вибір необхідного облікового запису – ввід пошти та паролю;
- підтвердження вибору.

Підсистема «real-time відеочат» повинна працювати (відкриється відеозв’язок між користувачами) після виконання одного з наведених кроків:

- створення нової кімнати для відеозв’язку шляхом натиснення на кнопку «Створити нову кімнату»;
- приєднання до існуючої кімнати шляхом натиснення на кнопку «Приєднатися» біля адреси необхідної кімнати (якщо існують);
- вставити необхідну url-адресу потрібної кімнати (скопіювати з чату).

3.1.2. Показники призначення

3.1.2.1. Параметри, що характеризують ступінь відповідності системи призначенням

Система повинна забезпечувати можливість історичного зберігання текстових повідомлень протягом 1 місяця, оскільки база даних Firebase забезпечує безкоштовне використання її сервісів лише на даний (тестовий) період. У разі необхідності є можливість придбати більш широкий пакет з розширеним функціоналом.

Система повинна забезпечувати можливість одночасної роботи декількох користувачів, незалежно від їх кількості.

Час відгуку системи під час підключення до серверу не повинен перевищувати 10 секунд. У випадку помилки підключення повинно повторюватися кожні 10 секунд.

3.1.2.2. Вимоги до пристосовності системи до змін

Забезпечення пристосовності системи повинно виконуватися за рахунок:

- своєчасності адміністрування системи (перевірка на коректність роботи, а також додавання нових можливостей);
- модернізації існуючих процесів відповідно до нових вимог;
- оновлення підписки на сервісі Firebase.

3.1.2.3. Вимоги до збереження працездатності системи в різних ймовірних умовах

Залежно від різних ймовірних умов система повинна:

- залишатися ввійденою в обліковий запис Google користувача;
- відновити підключення до кімнати, до якої був підключений клієнт.

3.1.3. Вимоги до надійності

3.1.3.1. Склад показників надійності до системи в цілому

Рівень надійності повинен досягатися узгодженим застосуванням організаційних, організаційно-технічних заходів і програмно-апаратних засобів.

Надійність повинна забезпечуватися за рахунок:

- своєчасного виконання процесів адміністрування системи СКМД;
- попереднього навчання користувачів.
- дотримання правил експлуатації і технічного обслуговування програмноапаратних засобів;

3.1.3.2. Вимоги до надійності технічних засобів і програмного забезпечення

До надійності електропостачання ставляться такі вимоги:

- з метою підвищення відмовостійкості системи в цілому бажана комплектація серверів джерелом безперебійного живлення з можливістю автономної роботи системи;
- бажане забезпечення безперебійним живленням активного мережного обладнання.

Надійність апаратних і програмних засобів повинна забезпечуватися за рахунок наступних організаційних заходів:

- попереднього навчання користувачів;
- своєчасного виконання процесів адміністрування системи СКМД;
- дотримання правил експлуатації і технічного обслуговування програмно-апаратних засобів.

Надійність програмного забезпечення підсистем повинна забезпечуватися за рахунок:

- надійності загальносистемного ПЗ та ПЗ, що розробляється;
- проведенням комплексу заходів пошуку і виключення помилок.

3.1.3.3. Вимоги до методів оцінки і контролю показників надійності на різних стадіях створення системи

Перевірка виконання вимог по надійності повинна проводитися на етапах випробувань і експлуатації за методикою Розробника.

3.1.4. Вимоги до ергономіки та технічної естетики

Система повинна забезпечувати зручний для кінцевого користувача інтерфейс, який відповідає таким вимогам:

- повинно бути забезпечено наявність україномовного інтерфейсу;
- повинен використовуватися шрифт: Times New Roman;
- розмір шрифту повинен бути: 16px;
- колірна палітра повинна бути: синьо-біла.

3.1.5. Вимоги до експлуатації, технічного обслуговування, ремонту

і зберігання компонентів системи

Види і періодичність обслуговування системи відбуватимуться за умови фактичного виявлення помилок. Будуть проходити виправні роботи та оновлення системи. Також за умови використання тестового режиму бази даних Firebase система вимагатиме оновлення підписки на даному сервісі кожні 30 днів з повною очисткою бази.

3.1.6. Вимоги до захисту інформації від несанкціонованого доступу

Захист інформації від несанкціонованого доступу в базі даних Firebase гарантує сам сервіс. Доступ до бази повинен мати лише Розробник. Також передбачається, що в системі повідомлення між користувачами знаходитимуться у відкритому доступі лише для тих клієнтів, які авторизувалися в СКМД за допомогою облікового запису Google.

3.1.7. Вимоги до захисту від впливу зовнішніх факторів

Електромагнітне випромінювання радіодіапазону, що виникає при роботі електропобутових приладів, електричних машин і установок, прийомопередаючих пристроїв, що експлуатуються на місці розміщення системи, не повинні призводити до порушень працездатності системи.

Система повинна мати можливість функціонування в діапазоні допустимих температур навколишнього середовища, встановлених виробником апаратних засобів.

3.2. Вимоги до видів забезпечення

3.2.1. Вимоги до математичного забезпечення

Вимоги не пред’являються.

3.2.2. Вимоги до інформаційного забезпечення

Зберігання даних (повідомлень з текстового real-time чату) в СКМД повинно забезпечуватися платформою Firebase щонайменше на 30 календарних днів з можливістю створення нової бази на такий самий період.

Інформаційних обмін в межах підсистеми «real-time чат» передбачає використання даних (ім’я та фотографія) з облікового запису Google користувача. Дана інформація має відображатися на сторінці після відправлення

коритувачем повідомлення в чаті.

Інформація в базі даних системи повинна зберігатися при виникненні аварійних ситуацій, пов'язаних зі збоями електроживлення, оскільки дані знаходитимуться в хмарній базі.

4. Вимоги до програмного забезпечення

Застосування системи передбачає використання браузеру на комп’ютерному чи мобільному пристрої, під’єднаному до мережі Інтернет.

5. Вимоги до технічного забезпечення

Для повноцінного використання СКМД необхідно забезпечитися одним з двох варіантів:

- комп’ютерний пристрій з виходом до Інтернету, комп’ютерна миша, клавіатура, веб-камера, навушники з мікрофоном;
- смартфон з виходом до Інтернету.

6. Вимоги до методичного забезпечення

Методичне забезпечення СКМД складається з комплексу методичних вказівок, рекомендацій і положень стосовно впровадження системи, повний список яких знаходиться в Списку використаних джерел Проєкту.

ПРОГРАМА ТА МЕТОДИКА ТЕСТУВАННЯ

Повністю розроблена система готова до проведення тестування на коректність роботи. Відбуватися це буде шляхом прямої перевірки працездатності кожної підсистеми та її компонентів.

Запустимо проект за допомогою команди npm run start в командному рядку. В браузері відкривається додаток – відразу бачимо навігаційну панель з кнопкою та банер з текстом (Рис. 1):

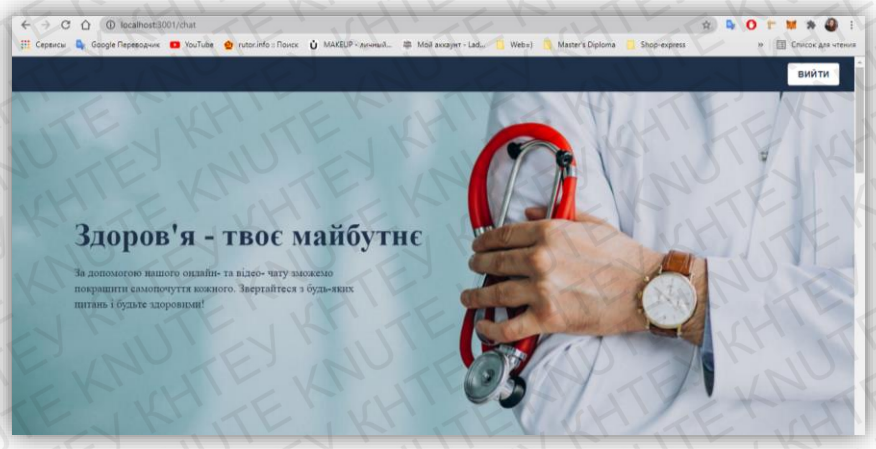


Рис. 1. Початковий екран додатку

Нижче знаходиться місце для онлайн-чату, який не відображається до тих пір, поки ми не увійдемо в обліковий запис Google (Рис. 2). Натискаємо на кнопку «Увійти за допомогою Google».

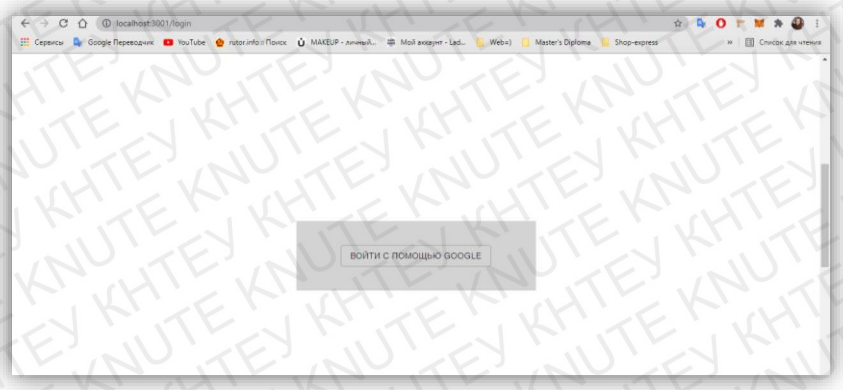


Рис. 2. Кнопка входу в акаунт Google

З’являється вікно для входу в обліковий запис – обираємо наш профіль (Рис. 3), або натискаємо на «Використати інший акаунт» вводимо логін і пароль у відповідні поля.

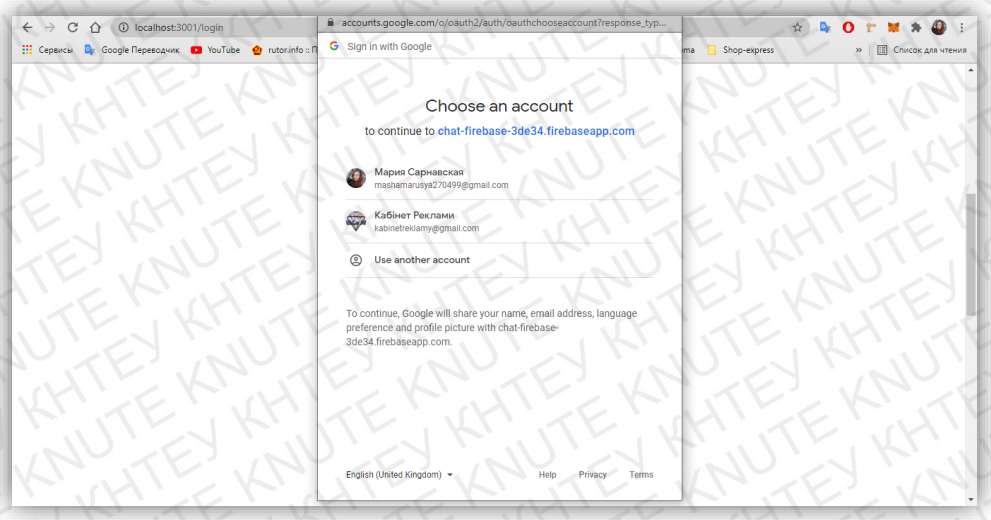


Рис. 3. Вибір користувача акаунту Google

Отримали сторінку з онлйн-чатом. Вводимо тестове повідомлення – бачимо, що забраження користувача, його ім’я та саме повідомлення коректно підтягнулися та відобразилися в блоці (Рис. 4):

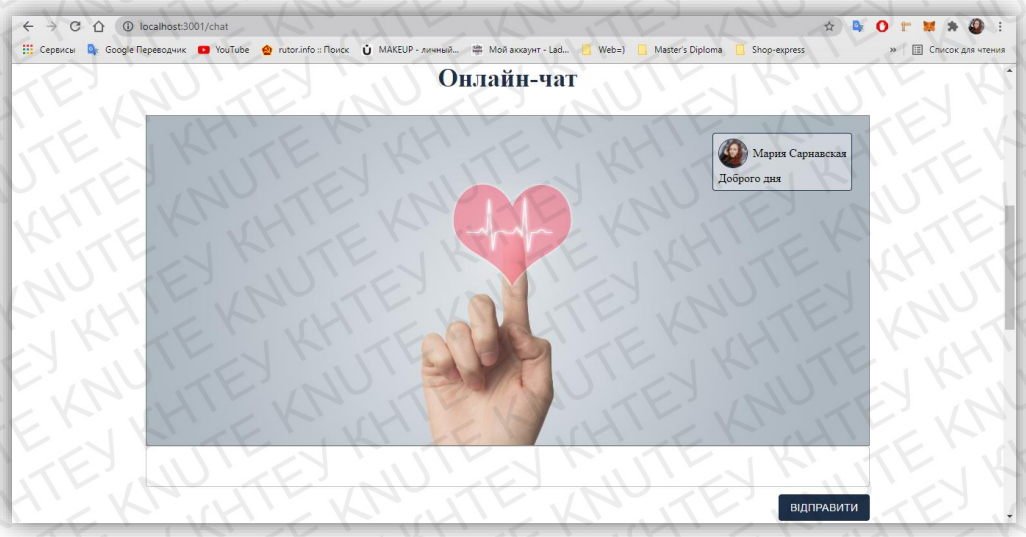
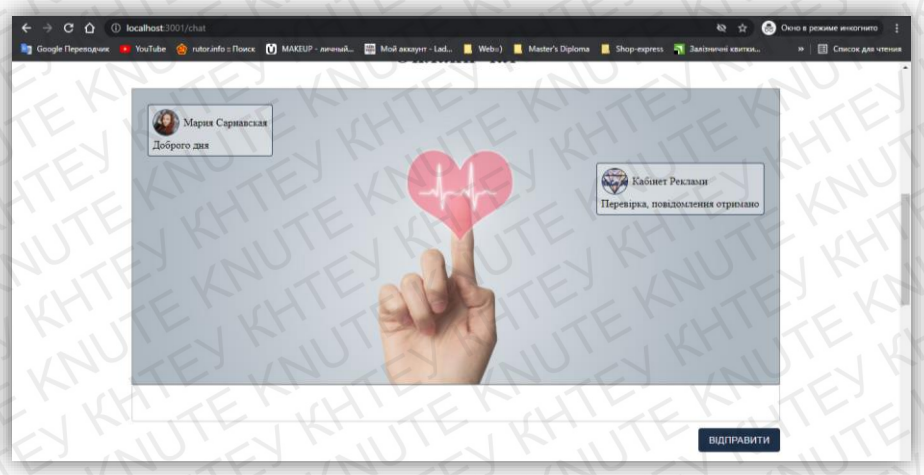


Рис. 4. Real-time чат додатку

У іншому вікні браузера під’єднуємося до чату за допомогою іншого акаунту Google. Можемо помітити, що в секції вже є наше перше повідомлення, яке знаходиться ліворуч. Надсилаємо відповідь. Перший користувач також отримав зворотнє повідомлення. Отже, все працює правильно (Рис. 5-6):



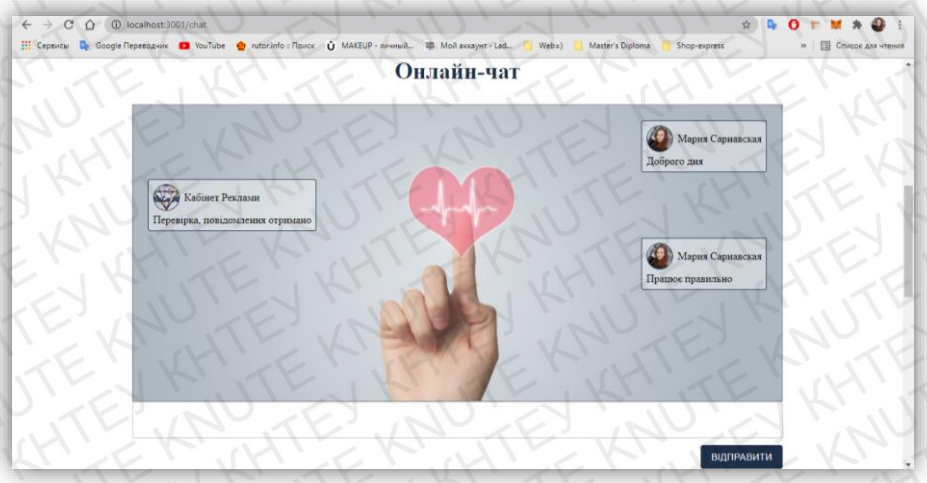


Рис. 5-6. Тестовий обмін повідомленнями в чаті

Перейдемо до перевірки відеочату, який знаходиться нижче. В полі «Доступні кімнати» немає ніяких адрес, оскільки ще не створено жодної кімнати. Натиснемо на кнопку «Створити нову кімнату» (Рис. 7):

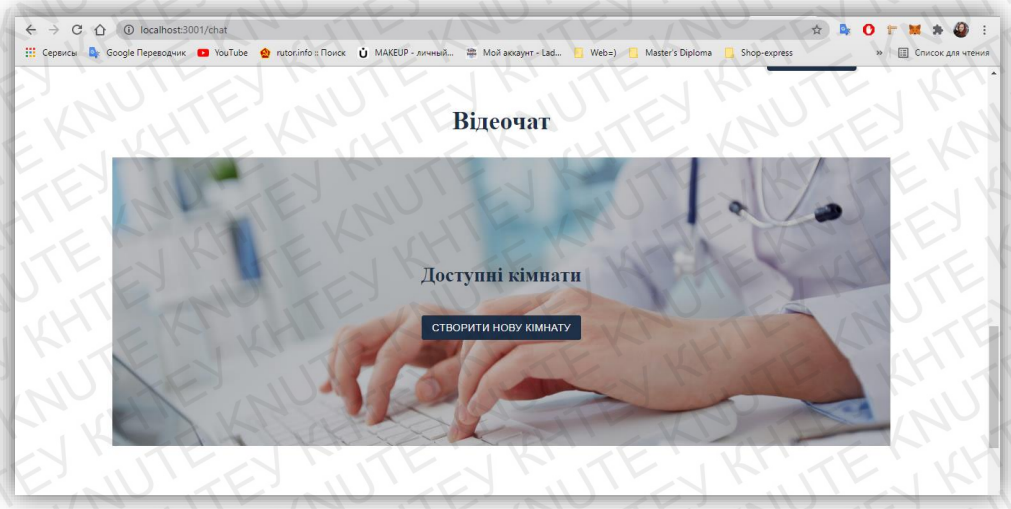


Рис. 7. Створення нової кімнати відеочату

З’являється блок з відеотрансляцією клієнта. Поки що відображається одне відео, оскільки в кімнаті лише один користувач (Рис. 8):

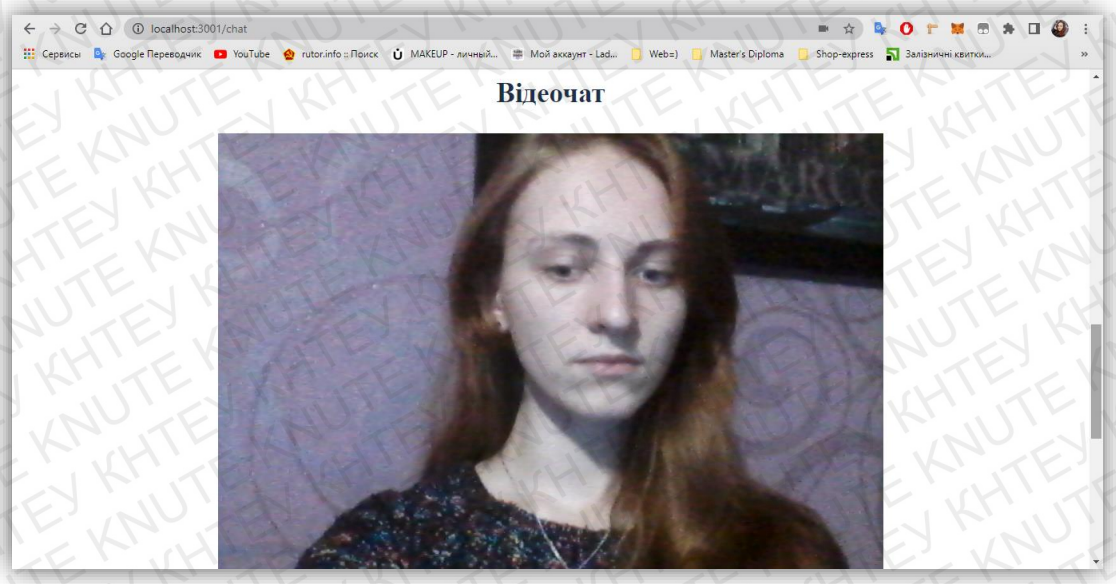


Рис. 8. Відеопотік першого користувача

Відразу після створення кімнати інший користувач вже бачить, що в доступі є одна кімната, до якої можна приєднатися. Натискаємо «Приєднатися до кімнати» (Рис. 9):

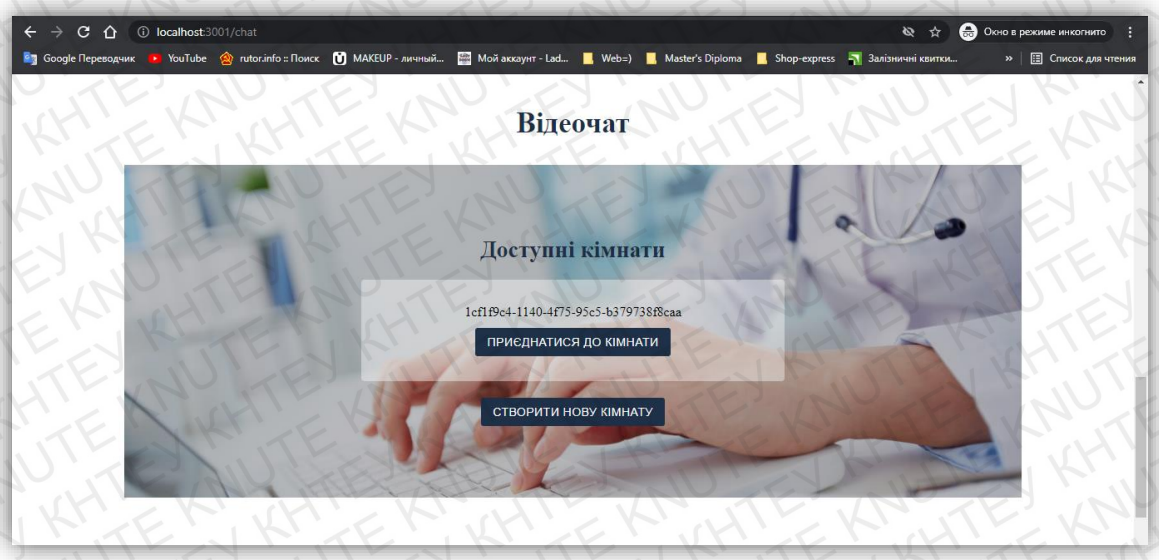


Рис. 9. Під’єднання до існуючої кімнати

Як і в першому випадку, отримуємо запит на дозвіл використовувати мікрофон та камеру. Натискаємо «Дозволити» (Рис. 10):

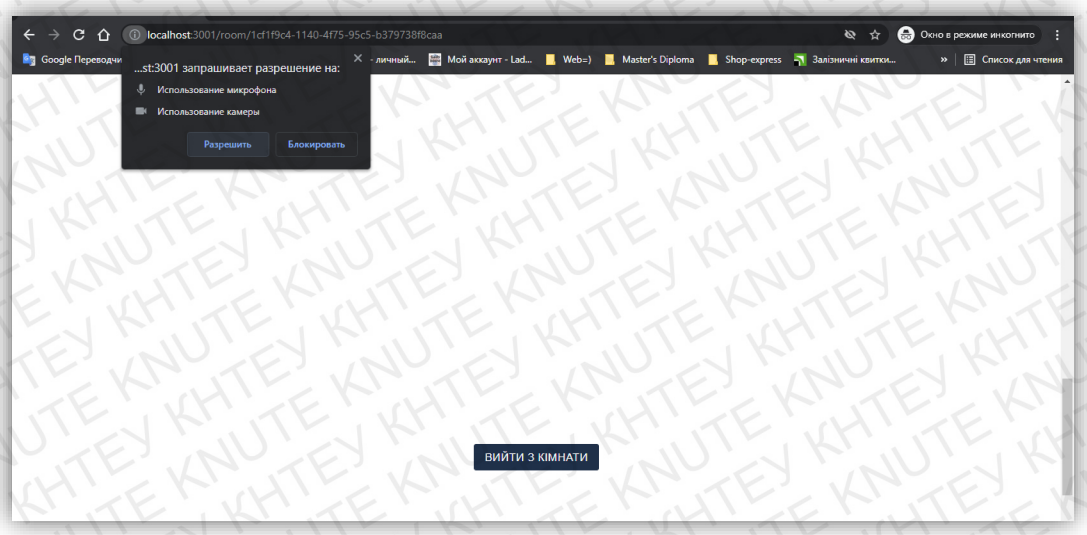


Рис. 10. Запит на дозвіл використання мікрофону та камери

Тепер на екрані відображається два відеопотоки, звук також працює правильно (Рис. 11). Для припинення трансляції натискаємо на кнопку «Вийти з кімнати».

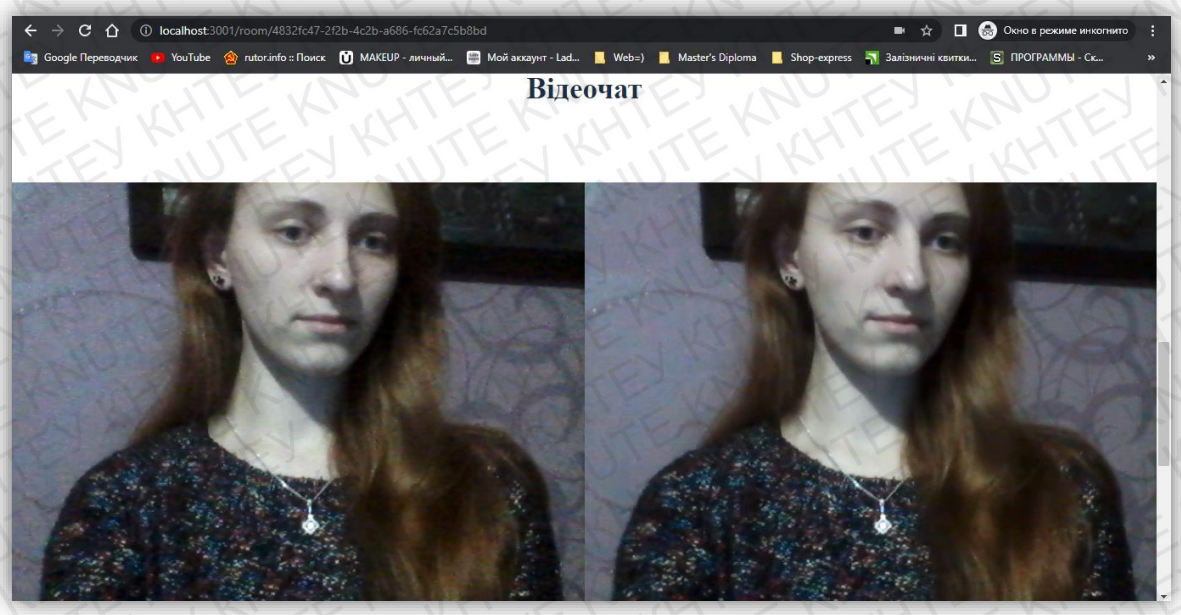


Рис. 11. Відеопотоки двох клієнтів

З самого низу знаходиться інформаційний блок з текстом про елемедицину, а також футер із назвою додатку (Рис. 12):

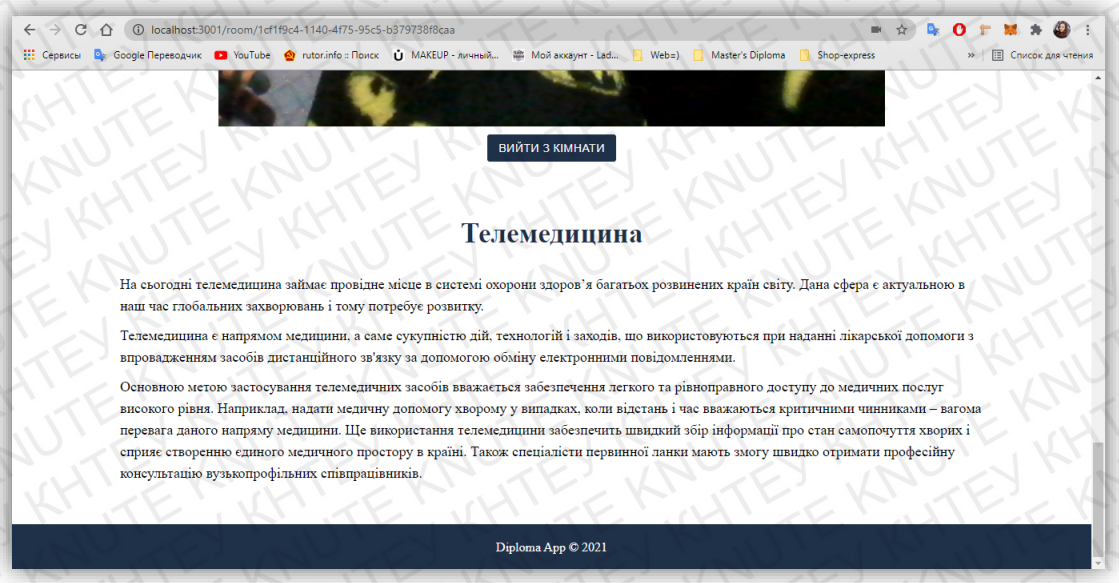


Рис. 12. Інформаційний блок додатку

Отже, розроблений додаток вдало пройшов перевірку. Усі підсистеми та її компоненти працюють правильно. Система виконує поставлене завдання - користувач має змогу в онлайн-режимі отримувати консультації лікаря.

КЕРІВНИЦТВО КОРИСТУВАЧА

Даний розділ призначений для полегшення роботи з системою користувачами. Для використання окремих компонентів системи варто покроково виконати пункти, описані у відповідних списках.

Підключення до real-time чату

1) Натискаємо на кнопку «Увійти за допомогою Google»;
2) Обираємо обліковий запис Google, вводимо пароль;
3) Підтрверджуємо вибір акаунту.

Відправка повідомлень в чаті

1) В секції «Онлайн-чат» під зображенням знаходимо поле для вводу повідомлень;
2) Пишемо необхідний текст для відправки;
3) Натискаємо на кнопку «Відправити».

Створення нової кімнати real-time відеочату

1) В секції «Відеочат» натискаємо на кнопку «Створити нову кімнату»;
2) Для надання дозволу на використання мікрофону та камери натискаємо на кнопку «Дозволити» у впливаючому вікні;
3) Для повідомлення користувачам адреси кімнати необхідно скопіювати та надіслати ID кімнати в чат (знаходиться в URL-адресі після «room/»).

Підключення існуючої кімнати real-time відеочату

1) В секції «Відеочат» знаходимо необхідний ID кімнати, до якої потрібно під’єднатися;
2) Натискаємо на кнопку «Приєднатися до кімнати».

ДОДАТКИ

Додаток А

Код файлу App.js

```
import { BrowserRouter, Switch, Route, Redirect } from 'react-router-dom';
import Room from './pages/Room';
import Main from './pages/Main';
import './App.css';
import NotFound404 from './pages/NotFound404';
import Footer from './components/Footer';

function App() {
 return (
  <BrowserRouter>
   <Switch>
    <Route exact path='/room/:id' component={Room} />
    <Route exact path='/' component={Main} />
    {/* <Route component={NotFound404}/> */}

    <Redirect to={"/"} />
   </Switch>
   <Footer />
  </BrowserRouter>
 );
}

export default App;
```

Код файлу server.js

```
const path = require('path');
const express = require('express');
const app = express();
const server = require('http').createServer(app);
const io = require('socket.io')(server);
const { version, validate } = require('uuid');

const ACTIONS = require('./src/socket/actions');
const PORT = process.env.PORT || 3001;

function getClientRooms() {
 const { rooms } = io.sockets.adapter;

 return Array.from(rooms.keys()).filter(roomID => validate(roomID) && version(roomID) === 4);
}

function shareRoomsInfo() {
 io.emit(ACTIONS.SHARE_ROOMS, {
  rooms: getClientRooms()
 })
}

io.on('connection', socket => {
 shareRoomsInfo();

 socket.on(ACTIONS.JOIN, config => {
  const { room: roomID } = config;
  const { rooms: joinedRooms } = socket;

  if (Array.from(joinedRooms).includes(roomID)) {
   return console.warn(`Already joined to ${roomID}`);
  }

  const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);

  clients.forEach(clientID => {
   io.to(clientID).emit(ACTIONS.ADD_PEER, {
    peerID: socket.id,
    createOffer: false
   });
```

```
socket.emit(ACTIONS.ADD_PEER, {
      peerID: clientID,
      createOffer: true,
    });
  });

  socket.join(roomID);
  shareRoomsInfo();
});

function leaveRoom() {
  const { rooms } = socket;

  Array.from(rooms)
    // LEAVE ONLY CLIENT CREATED ROOM
    .filter(roomID => validate(roomID) && version(roomID) === 4)
    .forEach(roomID => {

      const clients = Array.from(io.sockets.adapter.rooms.get(roomID) || []);

      clients
        .forEach(clientID => {
          io.to(clientID).emit(ACTIONS.REMOVE_PEER, {
            peerID: socket.id,
          });

          socket.emit(ACTIONS.REMOVE_PEER, {
            peerID: clientID,
          });
        });

      socket.leave(roomID);
    });
  shareRoomsInfo();
}

socket.on(ACTIONS.LEAVE, leaveRoom);
socket.on('disconnecting', leaveRoom);

socket.on(ACTIONS.RELAY_SDP, ({ peerID, sessionDescription }) => {
  io.to(peerID).emit(ACTIONS.SESSION_DESCRIPTION, {
    peerID: socket.id,
    sessionDescription,
  });
});
```

```
  socket.on(ACTIONS.RELAY_ICE, ({ peerID, iceCandidate }) => {
   io.to(peerID).emit(ACTIONS.ICE_CANDIDATE, {
     peerID: socket.id,
     iceCandidate,
   });
  });

});

const publicPath = path.join(__dirname, 'build');

app.use(express.static(publicPath));

app.get('*', (req, res) => {
 res.sendFile(path.join(publicPath, 'index.html'));
});

server.listen(PORT, () => {
 console.log('Server Started!')
})
```

Код файлу Main.js

```
import { useState, useEffect, useRef } from 'react';
import socket from '../../socket';
import ACTIONS from '../../socket/actions';
import { useHistory } from 'react-router';
import { v4 } from 'uuid';

export default function Main() {
 const history = useHistory();
 const [rooms, updateRooms] = useState([]);
 const rootNode = useRef();

 useEffect(() => {
  socket.on(ACTIONS.SHARE_ROOMS, ({ rooms = [] } = {}) => {
   if (rootNode.current) {
    updateRooms(rooms);
   }
  });
 }, []);

 return (
  <div className="main-video-container">
   <div className="title" >
    <h2>Відеочат</h2>
   </div>
   <div className="main-video" ref={rootNode}>
    <h3>Доступні кімнати</h3>
    <ul>
     {rooms.map(roomID => (
      <li key={roomID}>
       {roomID}
       <button onClick={() => {
        history.push(`/room/${roomID}`);
       }}>Приєднатися до кімнати</button>
      </li>
     ))}
    </ul>
    <button onClick={() => {
     history.push(`/room/${v4()}`);
    }}>Створити нову кімнату</button>
   </div>
  </div>
 );
}
```

Код файлу Room.js

```
import { useParams } from 'react-router';
import useWebRTC, { LOCAL_VIDEO } from '../../hooks/useWebRTC';
import { useHistory } from 'react-router';

function layout(clientsNumber = 1) {
  const pairs = Array.from({ length: clientsNumber })
    .reduce((acc, next, index, arr) => {
      if (index % 2 === 0) {
        acc.push(arr.slice(index, index + 2));
      }

      return acc;
    }, []);

  const rowsNumber = pairs.length;
  const height = `${100 / rowsNumber}%`;

  return pairs.map((row, index, arr) => {

    if (index === arr.length - 1 && row.length === 1) {
      return [{
        width: '100%',
        height,
      }];
    }

    return row.map(() => ({
      width: '50%',
      height,
    }));
  }).flat();
}

export default function Room() {
  const { id: roomID } = useParams();
  const { clients, provideMediaRef } = useWebRTC(roomID);
  const videoLayout = layout(clients.length);

  const history = useHistory();
```

Продовження дод. Г

```
return (
  <div className="room-page">
    <div className="title" >
     <h2>Відеочат</h2>
    </div>

    <div style={{
     display: 'flex',
     alignItems: 'center',
     justifyContent: 'center',
     flexWrap: 'wrap',
     height: '100vh',
    }}>

     {clients.map((clientID, index) => {
      return (
       <div key={clientID} style={videoLayout[index]} id={clientID}>
        <video
         width='100%'
         height='100%'
         ref={instance => {
          provideMediaRef(clientID, instance);
         }}
         autoPlay
         playsInline
         muted={clientID === LOCAL_VIDEO}
        />
       </div>
      );
     })}
    </div>

    <div className="room-exit-button">
     <button onClick={() => { history.push('/'); }}>Вийти з кімнати</button>
    </div>
  </div>
 );
}
```

Код файлу useWebRTC.js

```
import {useEffect, useRef, useCallback} from 'react';
import freeice from 'freeice';
import useStateWithCallback from './useStateWithCallback';
import socket from '../socket';
import ACTIONS from '../socket/actions';
export const LOCAL_VIDEO = 'LOCAL_VIDEO';

export default function useWebRTC(roomID) {
 const [clients, updateClients] = useStateWithCallback([]);

 const addNewClient = useCallback((newClient, cb) => {
  updateClients(list => {
   if (!list.includes(newClient)) {
    return [...list, newClient]
   }
   return list;
  }, cb);
 }, [clients, updateClients]);

 const peerConnections = useRef({});
 const localMediaStream = useRef(null);
 const peerMediaElements = useRef({
  [LOCAL_VIDEO]: null,
 });

 useEffect(() => {
  async function handleNewPeer({peerID, createOffer}) {
   if (peerID in peerConnections.current) {
    return console.warn(`Already connected to peer ${peerID}`);
   }

   peerConnections.current[peerID] = new RTCPeerConnection({
    iceServers: freeice(),
   });

   peerConnections.current[peerID].onicecandidate = event => {
    if (event.candidate) {
     socket.emit(ACTIONS.RELAY_ICE, {
      peerID,
      iceCandidate: event.candidate,
     });
    }
   }
```

Продовження дод. Д

```
  let tracksNumber = 0;
  peerConnections.current[peerID].ontrack = ({streams: [remoteStream]}) => {
   tracksNumber++

   if (tracksNumber === 2) { // video & audio tracks received
    tracksNumber = 0;
    addNewClient(peerID, () => {
     if (peerMediaElements.current[peerID]) {
      peerMediaElements.current[peerID].srcObject = remoteStream;
     } else {

      let settled = false;
      const interval = setInterval(() => {
       if (peerMediaElements.current[peerID]) {
        peerMediaElements.current[peerID].srcObject = remoteStream;
        settled = true;
       }

       if (settled) {
        clearInterval(interval);
       }
      }, 1000);
     }
    });
   }
  }

  localMediaStream.current.getTracks().forEach(track => {
   peerConnections.current[peerID].addTrack(track, localMediaStream.current);
  });

  if (createOffer) {
   const offer = await peerConnections.current[peerID].createOffer();

   await peerConnections.current[peerID].setLocalDescription(offer);

   socket.emit(ACTIONS.RELAY_SDP, {
    peerID,
    sessionDescription: offer,
   });
  }
 }

 socket.on(ACTIONS.ADD_PEER, handleNewPeer);
```

```
    return () => {
      socket.off(ACTIONS.ADD_PEER);
    }
  }, []);

  useEffect(() => {
    async function setRemoteMedia({peerID, sessionDescription: remoteDescription}) {
      await peerConnections.current[peerID]?.setRemoteDescription(
        new RTCSessionDescription(remoteDescription)
      );

      if (remoteDescription.type === 'offer') {
        const answer = await peerConnections.current[peerID].createAnswer();

        await peerConnections.current[peerID].setLocalDescription(answer);

        socket.emit(ACTIONS.RELAY_SDP, {
          peerID,
          sessionDescription: answer,
        });
      }
    }

    socket.on(ACTIONS.SESSION_DESCRIPTION, setRemoteMedia)

    return () => {
      socket.off(ACTIONS.SESSION_DESCRIPTION);
    }
  }, []);

  useEffect(() => {
    socket.on(ACTIONS.ICE_CANDIDATE, ({peerID, iceCandidate}) => {
      peerConnections.current[peerID]?.addIceCandidate(
        new RTCIceCandidate(iceCandidate)
      );
    });

    return () => {
      socket.off(ACTIONS.ICE_CANDIDATE);
    }
  }, []);

  useEffect(() => {
    const handleRemovePeer = ({peerID}) => {
      if (peerConnections.current[peerID]) {
```

```
    peerConnections.current[peerID].close();
   }

   delete peerConnections.current[peerID];
   delete peerMediaElements.current[peerID];

   updateClients(list => list.filter(c => c !== peerID));
  };

  socket.on(ACTIONS.REMOVE_PEER, handleRemovePeer);

  return () => {
   socket.off(ACTIONS.REMOVE_PEER);
  }
 }, []);

 useEffect(() => {
  async function startCapture() {
   localMediaStream.current = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: {
     width: 1280,
     height: 720,
    }
   });

   addNewClient(LOCAL_VIDEO, () => {
    const localVideoElement = peerMediaElements.current[LOCAL_VIDEO];

    if (localVideoElement) {
     localVideoElement.volume = 0;
     localVideoElement.srcObject = localMediaStream.current;
    }
   });
  }

  startCapture()
   .then(() => socket.emit(ACTIONS.JOIN, {room: roomID}))
   .catch(e => console.error('Error getting userMedia:', e));

  return () => {
   localMediaStream.current.getTracks().forEach(track => track.stop());
   socket.emit(ACTIONS.LEAVE);
  };
 }, [roomID]);
```

Продовження дод. Д

```
 const provideMediaRef = useCallback((id, node) => {
  peerMediaElements.current[id] = node;
 }, []);

 return {
  clients,
  provideMediaRef
 };
}
```

Код файлу Navbar.js

```
import React, { useContext } from 'react';
import AppBar from "@material-ui/core/AppBar";
import Toolbar from "@material-ui/core/Toolbar";
import { Button, Grid } from "@material-ui/core";
import { NavLink } from "react-router-dom";
import { LOGIN_ROUTE } from "../utils/consts";
import { Context } from "../index";
import { useAuthState } from "react-firebase-hooks/auth";

const Navbar = () => {
  const { auth } = useContext(Context)
  const [user] = useAuthState(auth)

  return (
    <AppBar color={"secondary"} position="static">
      <Toolbar variant={"dense"}>
        <Grid container justify={"flex-end"}>
          {user ?
            <Button onClick={() => auth.signOut()} variant={"outlined"}>Вийти</Button>
            :
            <NavLink to={LOGIN_ROUTE}>
              <Button variant={"outlined"}>Логін</Button>
            </NavLink>
          }
        </Grid>
      </Toolbar>
    </AppBar>
  );
};

export default Navbar;
```

Код файлу index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { createContext } from 'react';
import App2 from './App2';
import firebase from "firebase";
import 'firebase/firestore'
import 'firebase/auth'

// Initialize Firebase
firebase.initializeApp({
  apiKey: "AIzaSyDGeh35gxgnHDqnJwIWV6sma9KtWDxKyPA",
  authDomain: "chat-firebase-3de34.firebaseapp.com",
  projectId: "chat-firebase-3de34",
  storageBucket: "chat-firebase-3de34.appspot.com",
  messagingSenderId: "409262147537",
  appId: "1:409262147537:web:8f616addf907ee2491c2f0",
  measurementId: "G-J4Q3K6NC29"
}
);

export const Context = createContext(null)

const auth = firebase.auth()
const firestore = firebase.firestore()

ReactDOM.render(
  <Context.Provider value={{
    firebase,
    auth,
    firestore
  }}>
    <App2 />
  </Context.Provider>,
  document.getElementById('chat')
);
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Код файлу Login.js

```
import React, {useContext} from 'react';
import {Button, Container, Grid} from "@material-ui/core";
import Box from "@material-ui/core/Box";
import {Context} from "../index";
import firebase from "firebase";


const Login = () => {
  const {auth} = useContext(Context)

  const login = async () => {
    const provider = new firebase.auth.GoogleAuthProvider()
    const {user} = await auth.signInWithPopup(provider)
    console.log(user)
  }

  return (
    <Container>
      <Grid container
            style={{height: window.innerHeight - 50}}
            alignItems={"center"}
            justify={"center"}
      >
        <Grid style={{width:400, background: 'lightgray'}}
              container
              alignItems={"center"}
              direction={"column"}
        >
          <Box p={5}>
            <Button onClick={login} variant={"outlined"}>Войти с помощью Google</Button>
          </Box>
        </Grid>
      </Grid>
    </Container>
  );
};

export default Login;
```

Код файлу Chat.js

```
import React, { useContext, useState } from 'react';
import { Context } from "../index";
import { useAuthState } from "react-firebase-hooks/auth";
import { Avatar, Button, Container, Grid } from "@material-ui/core";
import TextField from "@material-ui/core/TextField";
import { useCollectionData } from "react-firebase-hooks/firestore";
import Loader from "./Loader";
import firebase from "firebase";

const Chat = () => {
  const { auth, firestore } = useContext(Context)
  const [user] = useAuthState(auth)
  const [value, setValue] = useState('')
  const [messages, loading] = useCollectionData(
    firestore.collection('messages').orderBy('createdAt')
  )

  const sendMessage = async () => {
    firestore.collection('messages').add({
      uid: user.uid,
      displayName: user.displayName,
      photoURL: user.photoURL,
      text: value,
      createdAt: firebase.firestore.FieldValue.serverTimestamp()
    })
    setValue('')
  }

  if (loading) {
    return <Loader />
  }

  return (
    <Container>
      <Grid container
        direction={"column"}
        justify={"center"}
        style={{ marginTop: 20 }}>
        <div className="title" >
          <h2>Онлайн-чат</h2>
        </div>
        <div className="chat-block" style={{ width: '80%', height: '450px', border: '1px solid gray', overflowY: 'auto',
margin: "0 auto" }}>
```

Продовження дод. К

```
            {messages.map(message =>
              <div className="chat-one-message" style={{
                margin: 10,
                marginLeft: user.uid === message.uid ? 'auto' : '10px',
                width: 'fit-content',
                padding: 7,
              }}>
                <Grid container>
                  <Avatar src={message.photoURL} />
                  <div>{message.displayName}</div>
                </Grid>
                <div>{message.text}</div>
              </div>
            )}
          </div>
          <Grid
            container
            direction={"column"}
            alignItems={"flex-end"}
            style={{ width: '80%', margin: "0 auto" }}
          >
            <TextField
              fullWidth
              rowsMax={2}
              variant={"outlined"}
              value={value}
              onChange={e => setValue(e.target.value)}
            />
            <Button className="send-button" onClick={sendMessage} variant={"outlined"}>Відправити</Button>
          </Grid>
        </Grid>
      </Container>
    );
};

export default Chat;
```