МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

«ВНУТРІШНІЙ МЕСЕНДЖЕР ТАНЦЮВАЛЬНОГО КОЛЕКТИВУ З АВТОМАТИЧНИМ ПІДБОРОМ УЧАСНИКІВ ДЛЯ ВИСТУПУ»

Здобувач освіти гр. ІН – 82 Михайло СВЕРДЛІКОВ

Науковий керівник,
кандидат фізико-математичних наук,
асистент кафедри комп'ютерних наук Олександр ВЛАСЕНКО

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Свердлікова Михайла Михайловича.

Тема: «ВНУТРІШНІЙ МЕСЕНДЖЕР ТАНЦЮВАЛЬНОГО КОЛЕКТИВУ З АВТОМАТИЧНИМ ПІДБОРОМ УЧАСНИКІВ ДЛЯ ВИСТУПУ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) літературний огляд за обраною тематикою роботи; 2) методика вирішення поставлених задач; 3) практична реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Олександр ВЛАСЕНКО
Завдання прийняв до виконання Михайло СВЕРДЛІКОВ

РЕФЕРАТ

Записка: 42 стор., 9 рис., 3 табл., 1 додаток, 12 джерел.

Об’єкт дослідження – веб-додатки.

Мета роботи – огляд сучасних методів розробки веб-додатків та розробка власного додатку, який буде виконувати функції чату з ідентифікацією користувача та ділення по кімнатам.

Результати – розроблено веб-додаток, за допомогою React, Node.js, Express, Socket.io. Додаток виконує авторизацію користувача, відображення даних про онлайн користувачів та їх імена та функцію обміну повідомленнями.

ВЕБ-ДОДАТОК, REACT, NODE.JS, EXPRESS, SOCKET.IO

ЗМІСТ

ВСТУП

Сучасний світ результат безкінечного розвитку інформаційних технологій. Починаючи з давніх часів, у кожну епоху був свій спосіб для обміну інформацією та збереження даних у тому чи іншому вигляді.

Результатом розвитку на даному етапі є велика кількість популярних соціальних мереж та месенджерів. Останні нині стрімко набирають популярність, витісняючи ряд соціальних мереж, переманюючи ряд користувачів зручним і простим інтерфейсом.

Актуальність роботи полягає у тому, що нині месенджери зайняли провідну нішу у світі для спілкування. Надзвичайно зручно мати у будь-який момент можливість написати будь-кому.

Тому, звісно, розробити функціонал, який буде таким же, як у провідних інструментів у світі одній людині складно, але створити власний аналог, який до того ж можна використовувати серед знайомих у своєму колі роботи або навчання, вартий того.

Основною метою роботи є розробка веб-додатку, який зможе у деякій мірі повторити функціонал сучасних месенджерів. Створити свій аналог, який зможуть використовувати люди різних професій та родів діяльності для організації інформацію у тому чи іншому колективі.

Додати до додатку найпопулярніші та найзручніші можливості, для цього необхідно провести порівняння та обґрунтувати необхідність того чи іншого функціоналу.

Розповсюдити розробку серед свого кола спілкування, зібрати відгуки та працювати над доробкою додатку та виправленням усіх проблем, які у будь-якому випадку будуть.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Порівняння найпопулярніших месенджерів

Нині у світі є декілька популярних месенджерів. Одні втрачають популярність, інші, навпаки, набирають. До яскравих прикладів можна віднести Whats App, Viber, Telegram.

Таблиця 1.1 – Порівняння функціоналу найпопулярніших месенджерів [1]

Функція/Додаток	Whats App	Viber	Telegram
Захист від скріншотів	-	+	+
Повідомлення, які знищуються через деякий час	-	-	+
Видалення для усіх учасників чату	+	+	+
Відправлення фото, відео, файлів	+	+	+
Голосові повідомлення	+	+	+
Відео чати	+	+	+
Відправлення файлів розміром 1Гб+	-	-	+

У таблиці 1.1 представлено порівняння найпопулярніших функцій, які притаманні месенджеру. З таблиці можна зробити висновок, що функціонал месенджерів максимально схожий. Різницю складає лише Telegram, який має увесь представлений функціонал.

Telegram нині набирає найбільшу популярність в Україні, там часом як Whats App та Viber, відходять на другий план. Так відбувається через те, що навіть у такій ніші як месенджери, розробники регулярно підтримують свій продукт та додають все новий та новий функціонал. Нині Telegram все більше і більше стає схожим на соціальну мережу, відкидаючи непотрібний функціонал, який лише заважає та роблячи акцент на деталях.

1.2. Аналіз плюсів та мінусів додатків, які порівнюються

Оскільки, як вже було сказано, Telegram, зайняв лідируючу позицію, то варто виділити проблемні місця конкурентів та зрозуміти, які мінуси та плюси мають додатки.

У месенджері Viber є декілька проблем, через які багато людей відмовляються від нього. По-перше, файли з Viber автоматично зберігаються у галереї та заповнюють пам’ять телефону. По-друге, невдалий UI. Невеликі кнопки, погана динаміка користування додатку негативно впливають на думку людей про додаток. По-третє, для того, щоб почати користуватись додатком з комфортом необхідно провести ряд налаштувань.

У месенджері Wats App проблеми схожі, але їх менше. Ui додатку дуже приємний, все працює плавно, а інтерфейс інтуїтивно зрозумілий. Проблемою є те, що він також зберігає фото та відео одразу в галерею.

З того часу, як додаток почала підтримувати Meta, він значно покращився. Додано ряд корисних функцій і для користування додатком не потрібно багато налаштувань.

З приводу Telegram важко виділити щось конкретне. Додаток розвивається неймовірно швидкими темпами та має на даний момент велику

спільноту користувачів. Telegram надає користувачам простір для креативу, дозволяючи створювати свої стікери та, навіть, локалізацію. Має велику кількість ботів, які, по-бажанню, можна підключати до своїх каналів. Також можна створювати власних ботів, які будуть мати великий простів функціоналу і підтримуватись платформою, що, безсумнівно, є великим плюсом.

З цього можна зробити висновок, що підтримка та креатив розробників Telegram поступово виводять месенджер на вершину у своїй галузі. Необхідно давати простір для користувачів, щоб додаток був популярний та регулярно додавати новий функціонал.

1.3 Постановка задачі

Для виконання поставленої мети, необхідно реалізувати наступні задачі:

1. проаналізувати найпопулярніші додатки у цій сфері;
2. виконати огляд інструментів для розробки;
3. обрати інструменти для розробки додатку;
4. розробити простий desktop-інтерфейс, який буду зрозумілий користувачу та містити необхідну інформацію;
5. спроектувати модель збереження даних локально, після ре-логіну у додаток, історія повідомлень повинна лишитись;
6. відображати коректну кількість учасників у кімнаті чату;
7. розробити спрощену модель авторизації через id кімнати на ім’я користувача.
8. провести тести додатку, необхідний функціонал: вхід/вихід із додатку, відправлення повідомлень, відображення учасників чату, їх кількості та імен, під кожним повідомленням повинно бути ім’я автора.

1.4 Методики розробки додатків

Веб-додатки у сучасному світі є одним із найпопулярнішим напрямом у сфері IT та дозволяють розробити як і простий функціонал, так і надзвичайно функціональний додаток. Яскравими прикладами є Facebook, YouTube, Twitch. Схоже між цими додатками є те, що вони функціонують як і у версії мобільного додатку так і у desktop версії. Розробка веб-додатків має багато варіантів інструментів для роботи та можливостей для написання проекту різноманітними способами та на різних стеках технологій.

Судячи із статистики, мобільний трафік нині трохи обходить трафік з комп'ютерів та ноутбуків, тому гарною ідеєю є інтегрування додатку на різні платформи.

Основною причиною вибору саме веб-додатку стало можливість тестування додатку за допомогою desktop-варіантів і реалізація повноцінного месенджеру на мобільній платформі враховуючи усі мінуси та рекомендації користувачів desktop-версії.

Також деякі технології дозволяють легко перейти з desktop-проекту до мобільного, використовуючи одне і те саме ядро, наприклад React та React Native.

1.5 Технології для розробки веб-додатків у 2022 році

Технологій для розробки веб-додатків у сучасній IT-сфері неймовірно багато. Звичайно, основою кожного додатку є HTML, CSS, JavaScript. Але, написання додатку на початковому стеку забере багато часту та ресурсів на розробку. Тому нині, популярністю користуються фреймворки та бібліотеки JavaScript.

Найпопулярнішою є React. Бібліотека працює на основі Virtual DOM-дерева, що слідкує за змінами компонентів, накопичує деяку кількість змін та вносить ці зміни лише там, де вони потрібні. У випадку з простим JavaScript відбувається re-render усього DOM. Ця особливість дозволяє забезпечити високу швидкість роботи. Бібліотеку розробила та нині підтримує компанія

Meta, що гарантує актуальність бібліотеки на ринку IT та довгострокову підтримку від розробника.

Другим за популярністю іде Vue.js. Це фреймворк, який у свої основі має дещо схоже на Virtual DOM React. Має велику аудиторію та широку підтримку. До плюсів варто віднести відносну легкість у вивченні та написанні додатків на Vue.js. Саме через це, компанії роблять ставку саме на нього.

Найбільш складним представником є Angular. Його розробила та підтримує компанія Google. Як у випадку з React, це гарантує довгострокову підтримку. Фреймворк написаний за допомогою TypeScript, тому має ідеальну інтеграцію з Angular. Але це в той самий час досить сильно підіймає поріг входу та складність написання додатку. Angular підходить для написання великих проектів, які мають надзвичайно широкий функціонал та багато обчислень.

Також, окрім фреймворків JS, є ряд інших технологій, за допомогою який можна створювати веб-додатки. Яскравим прикладом є Python. Також, нині з’явилось багато компіляторів коду, які переводять деяку мову програмування у JS, який розуміє браузер. Тобто програмісту лише залишається зробити вибір серед такого широкого спектру технологій і почати розробку.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Стек технологій для розробки додатку

У якості головного інструменту для написання клієнтської частини був обраний React. Інструмент, який ідеально підходить для створення простих додатків та не буде навантажувати кінцевий результат розробки. Враховуючи специфіку роботи React, то для коректної роботи, необхідно додати декілька бібліотек, які будуть використовуватись разом.

Враховуючи сучасні тенденції, коли кожна дія користувача викликає взаємодію клієнта з сервером, то для відправлення запитів на сервер необхідний axios. Він написаний на база fetch, який присутній у стандартному JS, але зменшує кількість рутинної роботи та дозволяє спростити синтаксис взаємодії із сервером [2].

Далі, необхідно належним чином взаємодіяти з сокетами на клієнтській частині додатку. Для цього було обрано Socket.io-client. Це більш зрозуміла версія для роботи з сокетами, побудована вона на базі WebSocket, але має більш зрозумілий інтерфейс та краще наповнену документацію [3].

Для написання серверної частини додатку було обрано Node.js. Він дозволяє створити сервер за допомогою JavaScript та взаємодіяти і з клієнтською частиною, і з серверною на одній мові програмування. Node.js також дозволяє запускати JS поза межами браузера, що і зробило його одним із найпопулярніших інструментів [4].

Але працювати нині на чистому Node.js незручно і не практично. Тому з ним у парі використовується Express. Це легкий фреймворк для створення серверної частини. Він працює з Node.js та суттєво полегшує створення архітектури серверу завдяки закладеним властивостям у сам фреймворк [5].

Далі необхідно працювати з сокетами на серверній частині. Для цього було використано socket.io. Він дозволяє легко відстежувати всі взаємодії з сокетами, які надходять з клієнтської частини та відповідним способом

обробляти дані, які отримуються. Це інструмент, який дозволяє взаємодіяти frontend та backend частині у максимально короткі стоки, що потрібно нам у форматі розробки месенджеру, де надзвичайно важлива швидкість надходження інформації [6].

Також при розробці сервера була використана утиліта nodemon. Її суть полягає у тому, що при кожній зміні у коді, вона автоматично перезапускає сервер у режимі розробки та суттєво зберігає час при роботі [7].

2.2 Клієнтська частина додатку

При розробці клієнтської частини основним завданням є створити проект, над яким у майбутньому буде легко працювати. Для цього необхідно створити якісну файлову структуру, з відокремленням стилів, бізнес-логіки, компонентів.

Стилі повинні розміщуватись у окремій папці у проекті та підключатись за допомогою import/export, які надає JS. Для написання стилів додатку було вирішено скористатись стандартним CSS, не підключаючи сторонніх бібліотек та фреймворків. Також важливо, не писати стилі в одному файлі. Структура React дозволяє організувати стилі багатьма способами, але один з найкращих підходів є відокремлення стилів, які стосуються конкретного компонента в окремий файл. Важливою умовою було написання відповідних класів, щоб не допустити їх повторення та некоректної роботи CSS.

Відокремлення бізнес-логіки проекту від компонентів потрібен для того, щоб підтримувати принцип єдиної відповідальності. Основна задача компонентів це render UI. Тому необхідно мінімізувати взаємодію файлів компонентів з логікою по обробці даних додатку.

Також, при плануванні роботи додатку, було вирішено не використовувати react-router-dom. Він потрібен для реалізації SPA-формату. У нинішньому стані, додаток має дві основні сторінки, це логін та сам

месенджер. Тому немає сенсу додавати додаткову бібліотеку на даному етапі. Але при збільшенні масштабів додатку, це буде обов'язковою умовою.

Для клієнт-серверної взаємодії необхідно створити ряд функцій, які будуть викликатись при деяких діях користувача. При процедурі логіну у додаток, відправлення повідомлень та відображенні даних про кількість учасників та самих учасниках буде працювати клієнт-серверна взаємодія. Але, важливо зауважити, що вона буде реалізована за допомогою сокетів.

2.3 Серверна частина додатку

Розробка серверної частини має за мету створення віддаленого сервісу, завдяки якому користувач зможе отримати дані з бази даних. Суть полягає у тому, що на сервері створюються N кількість роутів, які необхідні для роботи з тим чи іншим елементом додатку. Коли на один із них приходить запит, то для нього прописана деяка логіка, яка і повертає необхідні дані на клієнтську частину, при умові виконання усіх вимог.

Основним завданням є реалізувати якісну взаємодію з сокетами.

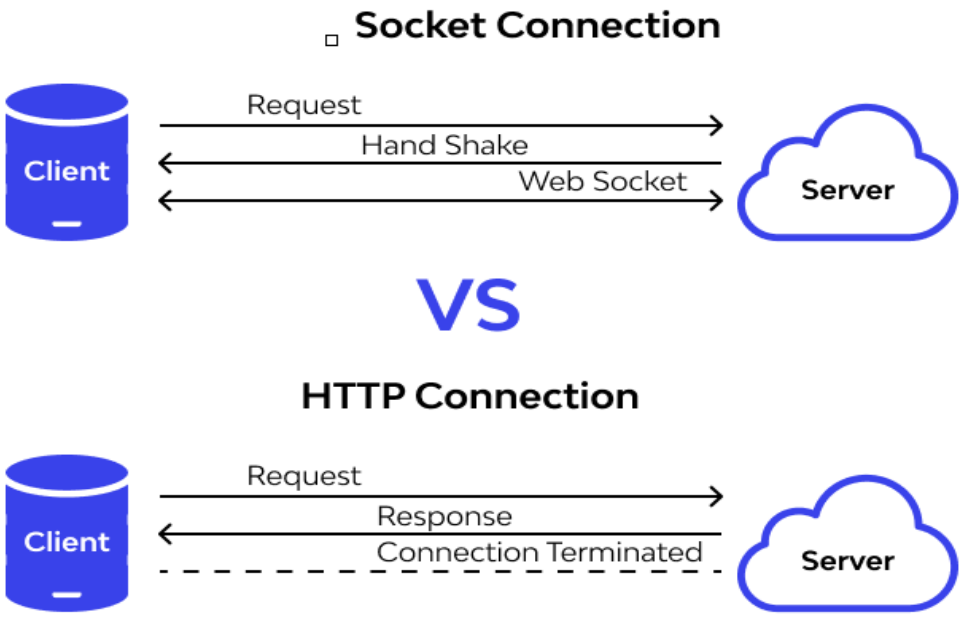

Рисунок 2.1 – Різниця між роботою серверу з сокетами та без них [8]

На рисунку 2.1 показана наглядна різниця між цими двома підходами. При наявності сокетів, створюється постійне сполучення між частинами додатку, що дозволяє максимально швидко отримати оновлені дані від сервера.

У цьому випадку, було прийнято рішення використати локальний аналог бази даних. Буде використано локальні можливості JavaScript такі як Map для створення місця для збереження даних та local storage для збереження даних у браузері.

2.4 Керування даними у додатку

Існує декілька підходів для керування бізнес логікою додатку. Найпопулярнішим підходом є використання state-manager, наприклад Redux.

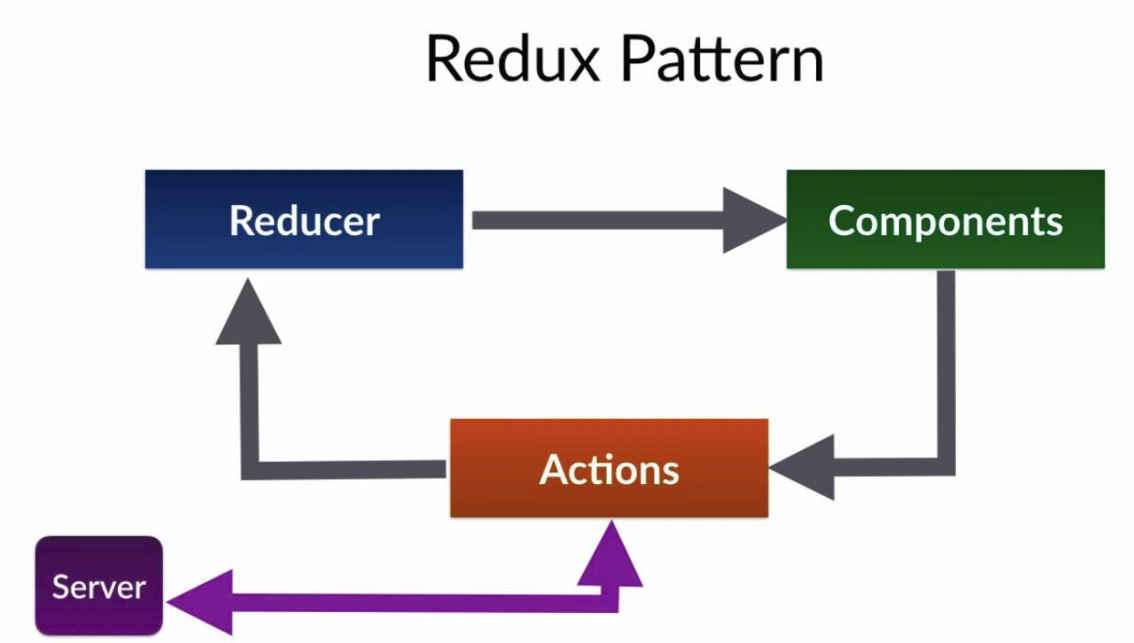

Рисунок 2.2 – Взаємодія Redux з додатком

На рисунку 2.2 наглядно показано, як впливає Redux на додаток. З’являється можливість відокремити логіку отримання та обробки даних з сервера від компонентів та спростити читання коду.

У випадку з месенджером буде використано спрощений варіант, який надає сам React – хук useReducer. Він надає трохи менші можливості для роботи зі станом додатку, але в масштабах мессенджеру їх достатньо. Хук надає можливість в окремому файлі створити reducer з типами та виконувати ту чи іншу логіку в компоненті.

Зберігати стан додатку дозволяє хук useState, який також надає React. Хук повертає сам стан та функцію, яка дозволяє його змінювати. Таким чином буде реалізована робота з формами та відправлення даних, які ввів користувач, на сервер.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

При розробці кожного додатку існує уявлення приблизного шляху користувача всередині додатку. На рисунку 3.1 приведені всі варіанти, які може обрати користувач.

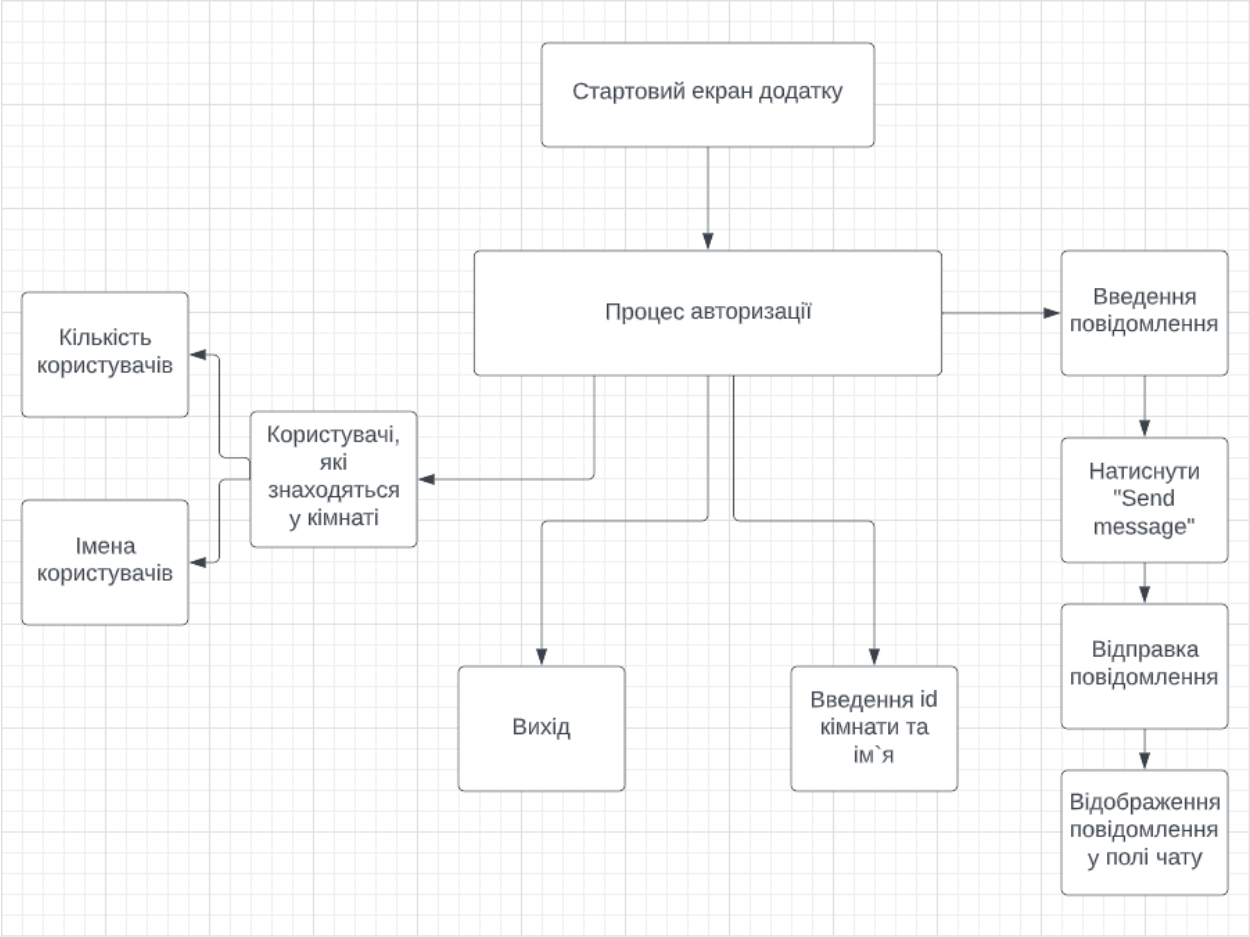

Рисунок 3.1 – Навігація по додатку

Таблиця 3.1 – Модель User для роботи

Users	Name:	Id:	Messages:	RoomId
	Data type: string	Data type: number	Data Type: Array	Data Type: number

Модель User представляє собою конфігурацію для збереження даних про користувача додатку. У таблиці 3.1 приведена реалізація для кожного користувача. Обов’язковими умовами є ім’я, його id та масив його повідомлень, які він залишив у рамках окремої кімнати(roomId).

Таблиця 3.2 – Модель Messages для роботи

Messages	roomId	Text:
	Data type: number	Data type: string

Модель Messages, яка приведена на рисунку 3.2, демонструє конфігурацію даних для повідомлень у додатку. Повідомлення будуть зберігатися у масиві, мати ідентифікатор кімнати, у якій вони мають відображатися та, відповідно, текст повідомлень.

3.2 Програмна реалізація

На рисунку 3.1 показано інтерфейс форми авторизації у додатку. Для того, щоб увійти у додаток та почати спілкуватися необхідно ввести id кімнати. Якщо такого id не буде знайдено, буде створена нова кімната та додатний один користувач. Також необхідно ввести ім’я. Це ім’я буде знаходитись під повідомленнями, для того, щоб було зрозуміло, хто відправник.

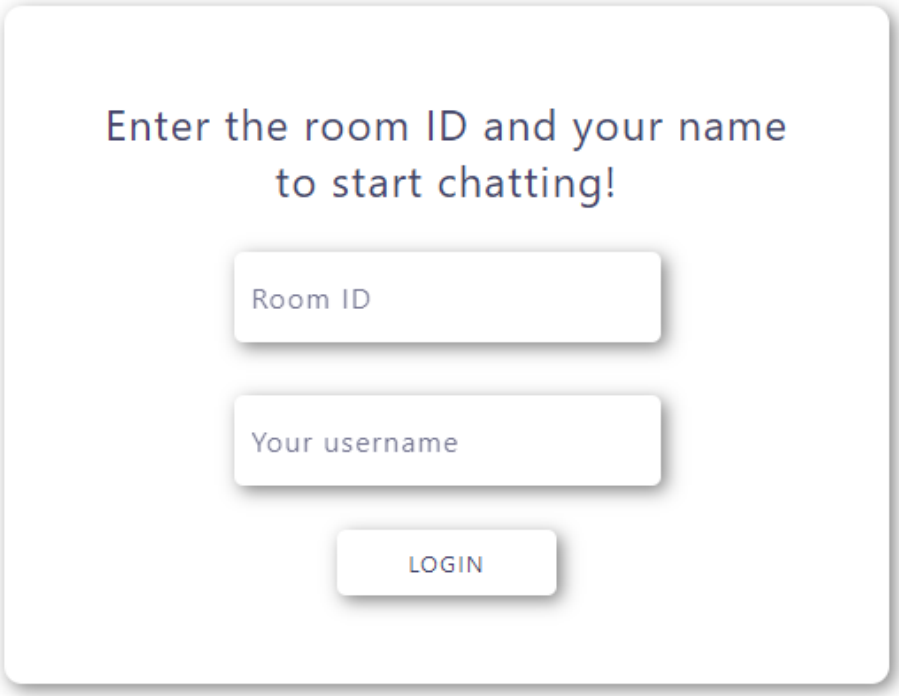

Рисунок 3.2 – Форма авторизації у додатку

Процес створення форми авторизації користувача у додатку полягає у наступному. Є div з класом “join-block”, який слугує контейнером для усіх елементів у ньому. Саме до цього елементу застосовується ряд flex-правил, які відповідають за позиціонування елементів у додатку.

```
<div className='join-block'>
                  <div className='title'>
                        Enter the room ID and your name to start chatting!
                  </div>
                  <input
                        type='text'
                        placeholder='Room ID'
                        value={roomId}
                        onChange={e => setRoomId(e.target.value)}
                  />
                  <input
                        type='text'
                        placeholder='Your username'
                        value={userName}
                        onChange={e => setUserName(e.target.value)}
                  />
                  <button     disabled={isLoading}     onClick={onEnter}
className='login'>
                        {isLoading ? 'Loading...' : 'Login'}
                  </button>
            </div>
      );
```

У контейнері знаходиться два input-поля для введення даних користувач. Перший для id, другий для імені. Ці два input-поля є керованими за рахунок того, що ми зберігаємо value у локальному стані компонента, який нам надає React хуком useState. Також, щоб отримувати актуальний стан кожного поля, у onChange ми передаємо функцію, яка динамічно змінює локальний стан додатку при введенні кожного символу та передає туди останнє значення.

У функції onEnter створена логіка відправлення даних на сервер для подальшого створення кімнати або входу у вже існуючу кімнату.

```
const [roomId, setRoomId] = useState('');
      const [userName, setUserName] = useState('');
      const [isLoading, setLoading] = useState(false);

      const onEnter = async () => {
            if (!roomId || !userName) {
                  return alert('Incorrect data');
            }
            const obj = {
                  roomId,
                  userName
            };
            setLoading(true);
            await axios.post('/rooms', obj);
            onLogin(obj);
      };
```

Спочатку йде невеличка перевірка на наявність даних у полях введення, далі формується об’єкт з даними, який і буде відправлено на сервер за допомогою axios.

Для відображення стану завантаження використовуються можливости useReducer.

Після того, як на сервер приходить post запит по шляху “/rooms” отримуємо id кімнати та ім’я користувача з тіла запиту – req. З допомогою деструктуризації об’єкту створюємо змінні для них для подальшого використання. Далі перевіряємо наявність кімнати і створюємо нову, якщо такої не знайшлось.

```
app.post('/rooms', (req, res) => {
  const { roomId, userName } = req.body;
```

```
  if (!rooms.has(roomId)) {
    rooms.set(
      roomId,
      new Map([
        ['users', new Map()],
        ['messages', []],
      ]),
    );
  }
  res.send();
});
```

Долучення до існуючого чату реалізовано на основі принципу роботи сокетів. При підключенні користувача у додаток, автоматично починають працювати сокети і викликають функцію зворотного виклику у цьому коді. За допомогою socket.join/get додаємо нового користувача, а далі оновлюємо список всіх користувачів у кімнаті для відображення коректної інформації на клієнтській частині додатку.

```
io.on('connection', (socket) => {
  socket.on('ROOM:JOIN', ({ roomId, userName }) => {
    socket.join(roomId);
    rooms.get(roomId).get('users').set(socket.id, userName);
    const users = [...rooms.get(roomId).get('users').values()];
    socket.to(roomId).broadcast.emit('ROOM:SET_USERS', users);
  });
```

Після авторизації користувач потрапляє на основну сторінку, яка показана на рисунку 3.3 і рисунку 3.4. Саме тут і відбувається листування між користувачами. У лівому верхньому куті вказано id кімнати, у якій знаходяться користувачі, кількість онлайн учасників одночасно та їх імена.

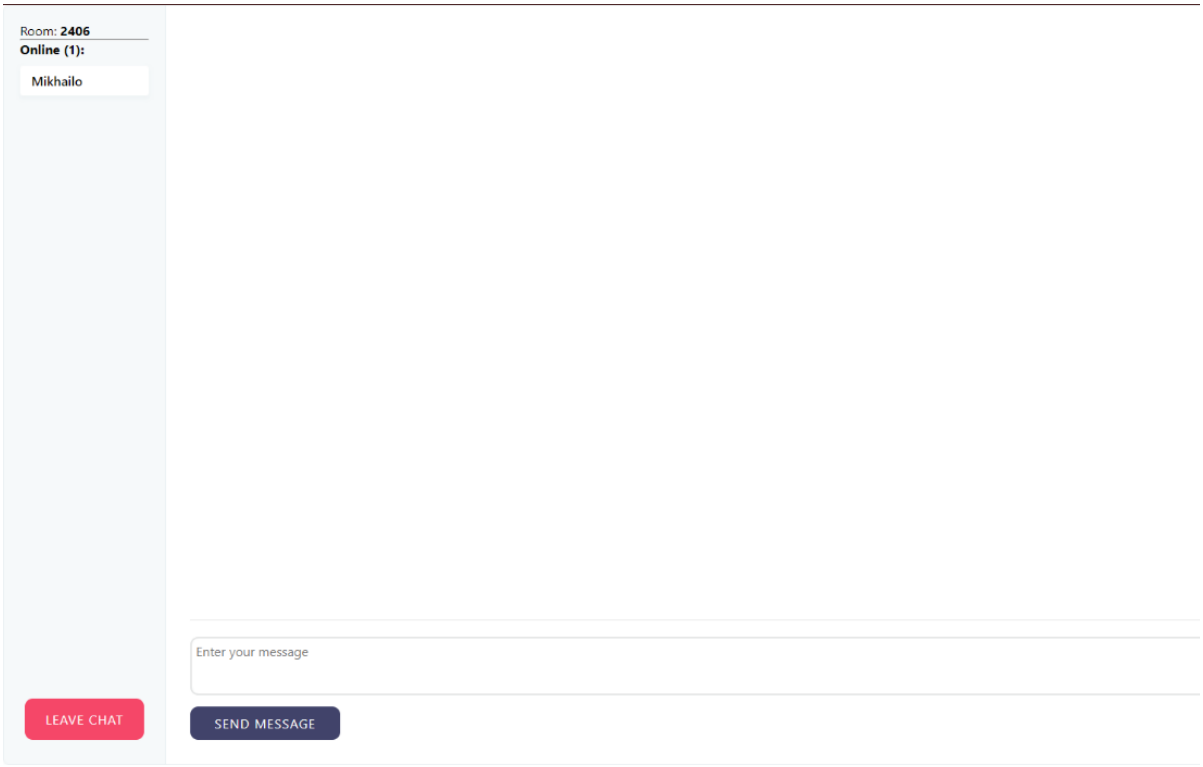

Рисунок 3.3 – Основна сторінка додатку

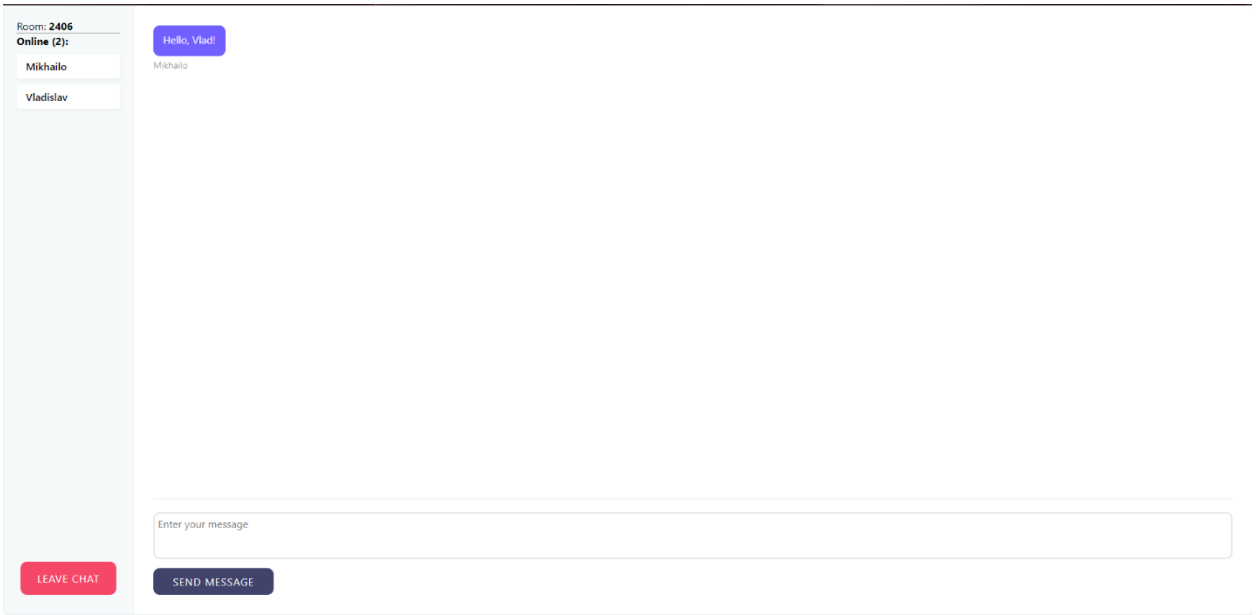

Рисунок 3.4 – Основна сторінка додатку, якщо у ньому декілька учасників

Розмітка для реалізації панелі користувачів. Для вказання id кімнати забираємо це значення с серверу. Також з серверу на клієнтську частину приходить масив користувачів. Для реалізації показу кількості учасників у чаті, викликаємо у цього масиву метод length. Далі за допомогою методу

масиву map додаємо усіх учасників на екран, виводячи їх імена. Вихід із додатку реалізований за допомогою useReducer.

```
<div className='chat-users'>
      <div>
      Room: <b>{roomId}</b>
      <hr />
      <b>Online ({users.length}):</b>
      <ul>
            {users.map((name, index) => (
            <li key={name + index}>{name}</li>
      ))}
      </ul>
      </div>
      <div className='chat-users__button'>
      <button className='exit' onClick={() => exit()}>
            Leave chat
      </button>
      </div>
</div>
```

Якщо другий користувач також увійшов у створену кімнату, наприклад з id: 2406, то він побачить історію повідомлень та інших учасників чату. Кількість учасників динамічно змінилась.

Форма для відправлення повідомлення реалізована наступним чином. Є textarea стан якого відстжується за допомогою локального стану та button на який вішається обробник події onClick та по кліку викликається необхідна функція.

```
<form>
      <textarea
            value={messageValue}
            onChange={e => setMessageValue(e.target.value)}
            className='form-control'
            placeholder='Enter your message'
            rows='3'
            ></textarea>
      <button
            onClick={onSendMessage}
            type='button'
            className='send'
            >
            Send message
      </button>
</form>
```

Відправлення повідомлень відбувається за допомогою сокетів. Ми відправляємо запит по шляху “ROOM:NEW_MESSAGE” та передаємо відповідні дані, після цього стан додатку оновлюється. Додавання

повідомлення на клієнті відбувається за допомогою action, який викликає необхідний reducer у контексті useReducer.

```
const onSendMessage = () => {
            socket.emit('ROOM:NEW_MESSAGE', {
                  userName,
                  roomId,
                  text: messageValue
            });
            onAddMessage({ userName, text: messageValue });
            setMessageValue('');
      };
```

На рисунку 3.5 продемонстровано адаптивний інтерфейс додатку під розміри ноутбука. Додаток має мінімалістичний дизайн, який легко відображати на різних пристроях.

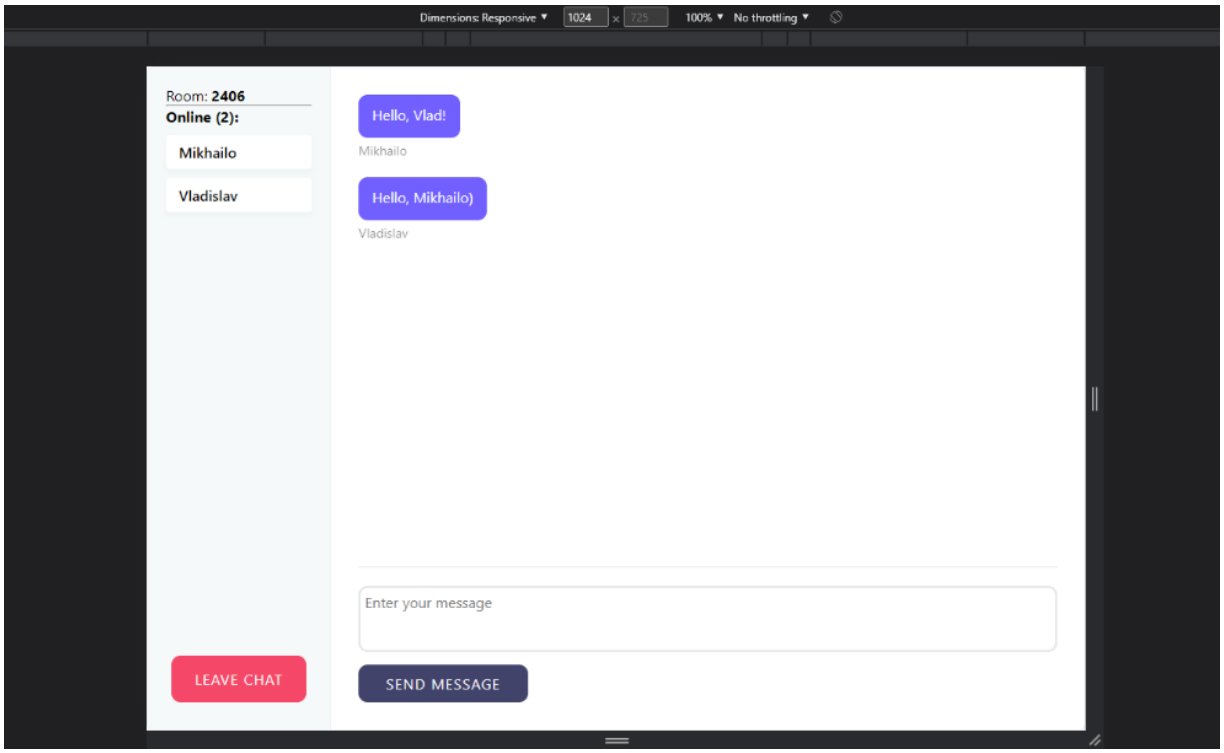

Рисунок 3.5 – Додаток на найменшому екрані ноутбука, який надає для тестування Google Chrome (1024px)

На рисунку 3.6 показано планшетну ширину екрану. Додаток коректно працює при такій ширині надаючи комфорт при користуванні користувачу.

На рисунку 3.7 показано відображення додатку на 320px – ширина iPhone 5. Блок з користувачами зникає, а кнопка для відправлення повідомлень

центрована. Зроблено адаптивне відображення UI за допомогою media-запитів у CSS.

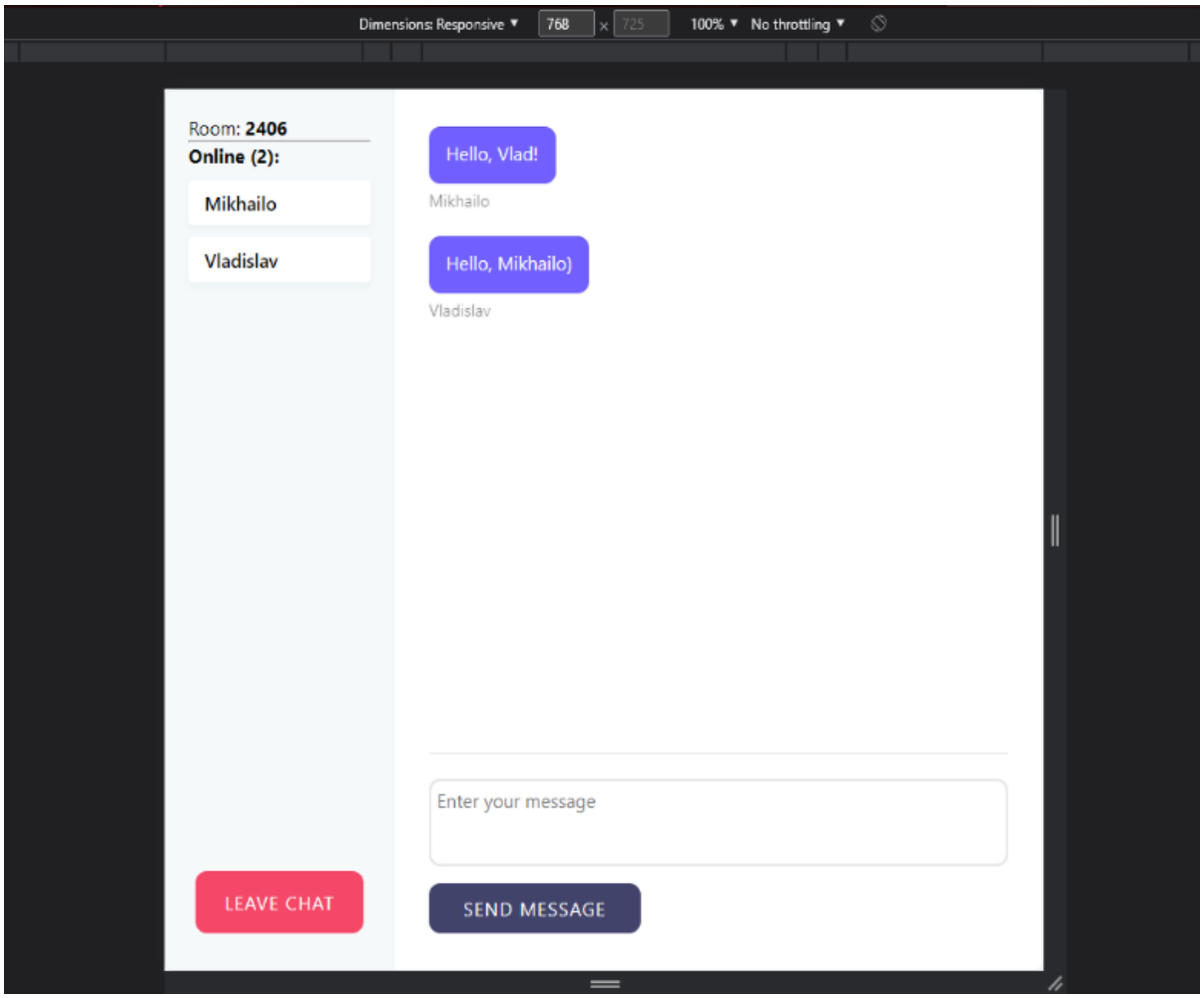

Рисунок 3.6 – Додаток на екрані планшета, який надає для тестування Google Chrome(768px)

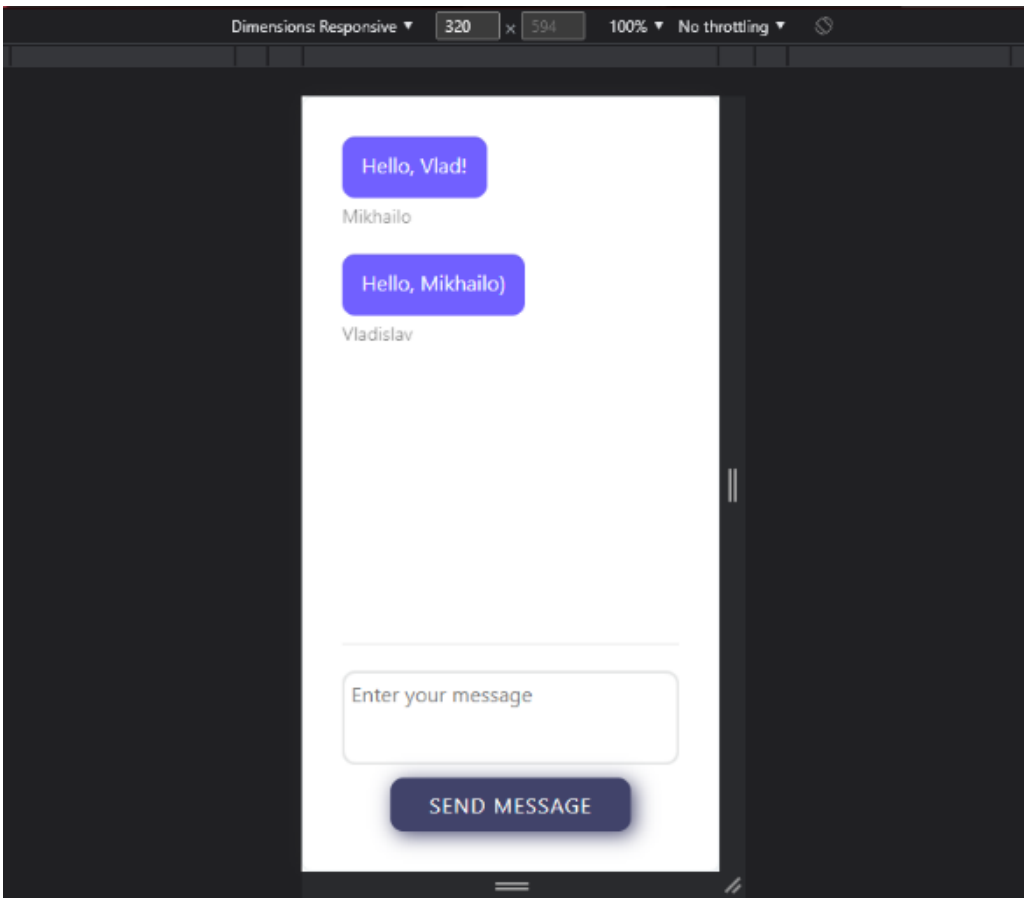

Рисунок 3.7 – Додаток на екрані найменшого телефона, який надає для тестування Google Chrome (320px)

Тестування створеного веб-додатку показало задовільні результати. Таким чином результатом роботи є функціонуючий додаток з функціями месенджера. Реалізовано ділення користувачів по кімнатам з можливістю обмінюватись повідомленнями та відображенням он-лайну та списку користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи було виконано:

1) порівняння сучасних месенджерів, виділення основних плюсів та мінусів;

2) проведено огляд сучасних технологій та обрано актуальний стек для розробки додатку;

3) було сформовано вимоги до додатку, чітко окреслений функціонал, який необхідно розробити;

4) розроблено концепцію розробки клієнтської та серверної частини додатку;

5) описано основні вимоги, які необхідно виконати стосовно структури проекту;

6) описано модель взаємодії з даними у додатку.

Результатом роботи є повністю функціонуючий додаток, який виконує функції месенджера. Реалізовано ділення користувачів по кімнатам, додавання повідомлень, відображення он-лайну та списку користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Стаття «Месенджери. Instant messangers» [Електронний ресурс] – Режим доступу до ресурсу: https://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:%D0%9C%D0%B5%D1%81%D1%81%D0%B5%D0%BD%D0%B4%D0%B6%D0%B5%D1%80%D1%8B_(Instant_Messenger,_IM)
2. Посібник: знаймоство з Axios [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/axios/axios
3. Посібник: знаймоство з Socket.io-client [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/socket.io-client
4. Посібник: знаймоство з NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/dist/latest-v17.x/docs/api/
5. Посібник: знаймоство з Express [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/
6. Посібник: знаймоство з Socket.io [Електронний ресурс] – Режим доступу до ресурсу: https://socket.io/docs/v4/how-it-works
7. Посібник: знаймоство з nodemon [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/remy/nodemon#nodemon
8. Стаття: «What is WebSocket and How It Works» [Електронний ресурс] – Режим доступу до ресурсу: https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is
9. Стаття: «React Redux Architecture Part-1» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/android-news/react-native-redux-architecture-part-1-8178fc9065c2
10. Посібник: знаймоство з useReducer [Електронний ресурс] – Режим доступу до ресурсу: https://ru.reactjs.org/docs/hooks-reference.html#usereducer
11. Посібник: знаймоство з useState [Електронний ресурс] – Режим доступу до ресурсу: https://ru.reactjs.org/docs/hooks-reference.html#usestate

12. Посібник: Адаптивна верстка сайтів: огляд підходів та CSS фреймворків– Режим доступу до ресурсу: https://jazzteam.org/ru/technical-articles/overview-of-approaches-and-css-frameworks-for-adaptive-web-page-layout/

ДОДАТОК

Файл index.html

```
<!DOCTYPE html>
<html lang="en">
	<head>
		<meta charset="utf-8" />
		<meta name="viewport" content="width=device-width, initial-
scale=1" />
		<title>S&P Chat</title>
	</head>
	<body>
		<div id="root"></div>
	</body>
</html>
```

Файл Chat.jsx

```
import React, { useState, useEffect, useRef } from 'react';
import socket from '../socket';

import '../styles/Chat.css';

function Chat({ users, messages, userName, roomId, onAddMessage, exit }) {
	const [messageValue, setMessageValue] = useState('');
	const messagesRef = useRef(null);

	const onSendMessage = () => {
		socket.emit('ROOM:NEW_MESSAGE', {
			userName,
			roomId,
			text: messageValue
		});
		onAddMessage({ userName, text: messageValue });
		setMessageValue('');
	};

	useEffect(() => {
		messagesRef.current.scrollTo(0, 99999);
	}, [messages]);

	return (
```

```
            <div className='chat'>
                  <div className='chat-users'>
                        <div>
                              Room: <b>{roomId}</b>
                              <hr />
                              <b>Online ({users.length}):</b>
                              <ul>
                                    {users.map((name, index) => (
                                          <li key={name + index}>{name}</li>
                                    ))}
                              </ul>
                        </div>
                        <div className='chat-users__button'>
                              <button className='exit' onClick={() => exit()}>
                                    Leave chat
                              </button>
                        </div>
                  </div>
                  <div className='chat-messages'>
                        <div ref={messagesRef} className='messages'>
                              {messages.map(message => (
                                    <div className='message'>
                                          <p>{message.text}</p>
                                          <div>

      <span>{message.userName}</span>
                                          </div>
                                    </div>
                              ))}
                        </div>
                        <form>
                              <textarea
                                    value={messageValue}
                                    onChange={e =>
setMessageValue(e.target.value)}
                                    className='form-control'
                                    placeholder='Enter your message'
                                    rows='3'
                              ></textarea>
                              <button
                                    onClick={onSendMessage}
                                    type='button'
```

```
                                    className='send'
                                >
                                    Send message
                                </button>
                            </form>
                        </div>
                    </div>
            );
}

export default Chat;
```

Файл JoinBlock.jsx

```
import React, { useState } from 'react';
import axios from 'axios';

import '../styles/JoinBlock.css';

function JoinBlock({ onLogin }) {
    const [roomId, setRoomId] = useState('');
    const [userName, setUserName] = useState('');
    const [isLoading, setLoading] = useState(false);

    const onEnter = async () => {
        if (!roomId || !userName) {
            return alert('Incorrect data');
        }
        const obj = {
            roomId,
            userName
        };
        setLoading(true);
        await axios.post('/rooms', obj);
        onLogin(obj);
    };

    return (
        <div className='join-block'>
            <div className='title'>
                Enter the room ID and your name to start chatting!
            </div>
            <input
                type='text'
```

```
                        placeholder='Room ID'
                        value={roomId}
                        onChange={e => setRoomId(e.target.value)}
                  />
                  <input
                        type='text'
                        placeholder='Your username'
                        value={userName}
                        onChange={e => setUserName(e.target.value)}
                  />
                  <button disabled={isLoading} onClick={onEnter}
className='login'>
                        {isLoading ? 'Loading...' : 'Login'}
                  </button>
            </div>
      );
}

export default JoinBlock;
```

Файл reducer.js

```
export default (state, action) => {
      switch (action.type) {
            case 'JOINED':
                  return {
                        ...state,
                        joined: true,
                        userName: action.payload.userName,
                        roomId: action.payload.roomId
                  };

            case 'SET_DATA':
                  return {
                        ...state,
                        users: action.payload.users,
                        messages: action.payload.messages
                  };

            case 'SET_USERS':
                  return {
                        ...state,
                        users: action.payload
                  };
```

```
            case 'NEW_MESSAGE':
                  return {
                        ...state,
                        messages: [...state.messages, action.payload]
                  };
            case 'EXIT':
                  return {
                        ...state,
                        joined: false
                  };

            default:
                  return state;
      }
};
```

Файл App.css

```
* {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,
Oxygen,
            Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
      outline: none;
      color: black;
      margin: 0;
      padding: 0;
      box-sizing: border-box;
}

.wrapper {
      min-height: 100vh;
      display: flex;
      flex-direction: column;
      justify-content: center;
}
```

Файл Chat.css

```
.chat {
      display: flex;
      height: 500px;
      border: 1px solid rgba(159, 183, 197, 0.2);
      border-radius: 8px;
      overflow: hidden;
```

```
      box-shadow: 5px 5px 10px rgba(159, 183, 197, 0.2);
      min-height: 100vh;
}

.chat-users {
      border-right: 1px solid rgba(159, 183, 197, 0.1);
      padding: 20px;
      width: 200px;
      background-color: #f6f9fa;
      display: flex;
      flex-direction: column;
      justify-content: space-between;
}
.chat-users__button {
      display: flex;
      justify-content: center;
      margin-bottom: 10px;
}
.chat-messages {
      display: flex;
      flex-direction: column;
      justify-content: space-between;
      flex: 1;
      padding: 30px;
}

.messages {
      flex: 1;
      height: calc(100% - 155px);
      overflow: auto;
}

.message {
      margin-bottom: 20px;
}

.message p {
      display: inline-flex;
      border-radius: 10px;
      border-top: 1px solid rgba(0, 0, 0, 0.1);
      background-color: #7160ff;
      padding: 10px 15px 15px;
```

```
      color: #fff;
      margin-bottom: 2px;
}

.message span {
      opacity: 0.5;
      font-size: 14px;
}

.chat-users ul {
      list-style: none;
      padding: 0;
      margin: 0;
}
.exit {
      border: none;
      background: #f54768;
      padding: 15px 25px;
      border-radius: 10px;
      text-transform: uppercase;
      color: #fff;
      letter-spacing: 1px;
      font-size: 16px;
}

.exit:hover {
      cursor: pointer;
      box-shadow: 3px 3px 15px #f54768;
      transition: all 0.3s;
}

.chat-users ul li {
      margin-top: 10px;
      border-radius: 3px;
      background-color: #fff;
      padding: 8px 14px;
      box-shadow: 0 3px 5px rgba(159, 183, 197, 0.1);
      font-weight: 500;
}

.chat-messages form {
      margin-top: 20px;
```

```
      padding-top: 20px;
      border-top: 1px solid rgba(0, 0, 0, 0.1);
}

.chat-messages form textarea {
      width: 100%;
      border: 2px solid rgba(7, 17, 24, 0.1);
      border-radius: 10px;
      font-size: 16px;
      padding: 5px 0 0 5px;
      color: #41436a;
      margin-bottom: 10px;
      resize: none;
}

.chat-messages form textarea:focus {
      box-shadow: 3px 3px 15px rgba(16, 33, 43, 0.1);
}

.send {
      border: none;
      background: #41436a;
      padding: 10px 30px;
      border-radius: 10px;
      text-transform: uppercase;
      color: #fff;
      letter-spacing: 1px;
      font-size: 16px;
}

.send:hover {
      cursor: pointer;
      box-shadow: 3px 3px 15px #41436a;
      transition: all 0.3s;
}

@media (max-width: 420px) {
      .chat-users {
            display: none;
      }
      form {
            display: flex;
```

```
            flex-direction: column;
            align-items: center;
      }
} ;
```

Файл JoinBlock.css

```
.join-block {
      margin: 0 auto;
      width: 300px;
      display: flex;
      flex-direction: column;
      align-items: center;
      padding: 50px 250px;
      box-shadow: 3px 3px 10px #808080;
      border-radius: 10px;
}

.title {
      letter-spacing: 1px;
      font-weight: 400;
      margin-bottom: 10px;
      font-size: 24px;
      color: #41436a;
      width: 400px;
      text-align: center;
}

.join-block input {
      padding: 15px 50px 15px 10px;
      border-radius: 5px;
      border: none;
      outline: none;
      box-shadow: 3px 3px 10px #808080;
      margin: 15px;
      font-size: 16px;
      letter-spacing: 1px;
      color: #41436a;
}

.join-block input::placeholder {
      color: #41436a;
      opacity: 0.75;
}
```

```
.join-block input:focus {
	box-shadow: 3px 3px 10px #41436a;
}

.join-block button {
	border: none;
	padding: 10px 40px;
	text-transform: uppercase;
	letter-spacing: 1px;
	box-shadow: 3px 3px 10px #808080;
	margin-top: 10px;
	border-radius: 5px;
	background: none;
	color: #41436a;
}

.login:hover {
	background: #41436a;
	color: #fff;
	transition: 0.3s;
	cursor: pointer;
}
```

Файл App.js

```
import React from 'react';
import axios from 'axios';

import socket from './socket';

import reducer from './reducers/reducer';
import JoinBlock from './components/JoinBlock';
import Chat from './components/Chat';

import './styles/App.css';

export function App() {
	const [state, dispatch] = React.useReducer(reducer, {
		joined: false,
		roomId: null,
		userName: null,
		users: [],
		messages: []
	});
```

```
const onLogin = async obj => {
      dispatch({
            type: 'JOINED',
            payload: obj
      });
      socket.emit('ROOM:JOIN', obj);
      const { data } = await axios.get(`/rooms/${obj.roomId}`);
      dispatch({
            type: 'SET_DATA',
            payload: data
      });
};

const setUsers = users => {
      dispatch({
            type: 'SET_USERS',
            payload: users
      });
};

const addMessage = message => {
      dispatch({
            type: 'NEW_MESSAGE',
            payload: message
      });
};
const exit = () => {
      dispatch({
            type: 'EXIT'
      });
};

React.useEffect(() => {
      socket.on('ROOM:SET_USERS', setUsers);
      socket.on('ROOM:NEW_MESSAGE', addMessage);
}, []);

window.socket = socket;

return (
      <div className='wrapper'>
```

```
                {!state.joined ? (
                      <JoinBlock onLogin={onLogin} />
                ) : (
                      <Chat {...state} onAddMessage={addMessage} exit={exit}
/>
                )}
            </div>
      );
}
```

Файл index.js

```
import React from 'react';
import ReactDOM from 'react-dom';

import { App } from './App';

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

Файл socket.js

```
import io from 'socket.io-client';

const socket = io();

export default socket;
```

Файл server.js

```
const express = require('express');

const app = express();
const server = require('http').Server(app);
const io = require('socket.io')(server);

app.use(express.json());

const rooms = new Map();

app.get('/rooms/:id', (req, res) => {
  const { id: roomId } = req.params;
  const obj = rooms.has(roomId)
    ? {
        users: [...rooms.get(roomId).get('users').values()],
        messages: [...rooms.get(roomId).get('messages').values()],
```

```
      }
    : { users: [], messages: [] };
  res.json(obj);
});

app.post('/rooms', (req, res) => {
  const { roomId, userName } = req.body;
  if (!rooms.has(roomId)) {
    rooms.set(
      roomId,
      new Map([
        ['users', new Map()],
        ['messages', []],
      ]),
    );
  }
  res.send();
});

io.on('connection', (socket) => {
  socket.on('ROOM:JOIN', ({ roomId, userName }) => {
    socket.join(roomId);
    rooms.get(roomId).get('users').set(socket.id, userName);
    const users = [...rooms.get(roomId).get('users').values()];
    socket.to(roomId).broadcast.emit('ROOM:SET_USERS', users);
  });

  socket.on('ROOM:NEW_MESSAGE', ({ roomId, userName, text }) => {
    const obj = {
      userName,
      text,
    };
    rooms.get(roomId).get('messages').push(obj);
    socket.to(roomId).broadcast.emit('ROOM:NEW_MESSAGE', obj);
  });

  socket.on('disconnect', () => {
    rooms.forEach((value, roomId) => {
      if (value.get('users').delete(socket.id)) {
        const users = [...value.get('users').values()];
        socket.to(roomId).broadcast.emit('ROOM:SET_USERS', users);
      }
```

```
    });
  });

  console.log('user connected', socket.id);
});

server.listen(9999, (err) => {
  if (err) {
    throw Error(err);
  }
  console.log('Server has been started!');
});
```