МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Порівняльний аналіз алгоритмів захисту веб-ресурсів з використанням Node.js»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Проценко О.Б.**

Студента групи КБ – 71 — **Микитченко М.І.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи КБ-71 спеціальності "Кібербезпека" денної форми навчання Микитченко Максима Ігоровича.

Тема: "**Порівняльний аналіз алгоритмів захисту веб-ресурсів з використанням Node.js** "

Затверджена наказом по СумДУ

№_____ от _____2021 р.

Зміст пояснювальної записки: 1) Інформаційний огляд; 2)вибір методів реалізації; 3)практична реалізація.

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____Проценко О. Б.

Завдання приняв до виконання _____ Микитченко М.І.

РЕФЕРАТ

Записка: 98 стор., 19 рис., 3 табл., 1 додаток, 18 джерел.

Об’єкт дослідження — алгоритми захисту веб-додатків.

Мета роботи — дослідження алгоритмів захисту веб-додатків, їх порівняння та створення додатку з їх використанням на основі Node.js.

Методи дослідження — емпіричний науковий метод.

Результати — був проведений порівняльний аналіз алгоритмів захисту веб додатків. Також був розроблений додаток для обміну повідомленнями в реальному часі з шифруванням повідомлень та хешуванням паролів. Для розробки веб-додатку використовувались: Node.js для серверної частини та ReactJS для клієнтської частини.

ШИФРУВАННЯ, АЛГОРИТМИ ЗАХИСТУ, КРИПТОГРАФІЯ, ВЕБ-ДОДАТКИ, ХЕШУВАННЯ, AES, DES, RSA, ELGAMAL, SHA-256, SHA-1, NODE.JS, REACTJS, MONGODB, WEBSOCKET API.

ЗМІСТ

ВСТУП

Сьогодні складно переоцінити значення веб-додатків, адже саме завдяки їм люди отримали чудові умови для навчання, роботи, розваг, тощо. Саме на сайтах можна знайти неосяжну кількість корисної та цікавої інформації, весело провести час, поспілкуватися з друзями в соціальних мережах, знайти хороший фільм, серіал або музичні композиції. Веб-сайти зручні тим, що реалізують принцип крос-платформи. Зараз ти вдома відкриваєш веб-додаток зі свого комп'ютера, а вже через годину можеш відкрити цей же додаток в метро на смартфоні. Актуальність веб-сайтів зрозуміла.

Веб-додатки можуть містити важливі та секретні данні. Отже зловмисники матимуть плани здобути їх. Розробка нових сайтів часто фокусується на запитах замовника і майже завжди вона націлена на забезпечення необхідної функціональності і зрозумілого якісного інтерфейсу для користувача. Також часто увага надається і серверній частині додатків. Питаннями безпеки інформації в додатках в той же час нехтують.

Методів захисту веб-додатків дуже багато. В цій роботі я розгляну методи захисту веб-додатків взагалі та більш детально буде розглянуті шифрування даних та хешування конфіденційної інформації.

Кожного дня ми обмінюємося даними з іншими людьми в реальному часі. Іноді ми відсилаємо важливі дані, а злодії можуть перехопити ці дані, та прочитати їх. Подолати цю проблему можна зашифровуючи усі повідомлення. Якщо зловмисники перехоплять зашифровані повідомлення, то вони нічного не зрозуміють.

Також в цій роботі я розроблю чат з використанням Node.js, в якому я буду шифрувати усі повідомлення. Разом з цим я буду хешувати паролі всіх користувачів при реєстрації задля ускладнення підбору паролів методом грубої сили.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Види загроз і захисту веб-додатку

Веб-додаток це клієнт-серверний додаток, в якому клієнт взаємодіє з сервером через браузер. Перевагою клієнт-серверного підходу є розділення навантажень між клієнтом та сервером. Обмін інформацією між цими частинами відбувається по мережі. У веб додатків є багато вразливих місць.

Види загроз веб-додатків:

- Атака з використанням SQL-ін’єкцій. Злочинець використовує поле веб-форми чи параметри URL-адреси для того, щоб отримати доступ до бази даних або модифікувати її. Коли використовується звичайний Transact SQL, легко можна вставити в запит шахрайський код і отримати доступ до бази даних. Можна легко захиститися від SQL-ін’єкцій, використовуючи запити з параметрами.
- Атаки з використанням міжсайтових сценаріїв. Злочинець впроваджує JavaScript код, який потім запускається в браузері користувача й виконує злочинні функції, може красти дані або змінювати сторінку. Гарним варіантом захисту проти міжсайтових сценаріїв є CSP. Це заголовок, який повертає сервер браузеру і повідомляє, як і який JavaScript обмежити в виконанні. Можна обмежити вбудований JavaScript.
- Отримання конфіденційних даних з помилок і виключень. Користувачам потрібно давати мінімальну інформацію про помилки, щоб вони не могли побачити в помилці ключ API чи пароль від бази даних.
- Обхід валідації браузера. Валідація повинна виконуватись завжди і на стороні клієнта і на стороні браузера. Так як валідацію зі сторони клієнта можна легко обійти, то на сервері повинна бути більш повна

та глибока валідація. Нехтування цим може призвести до потрапляння злочинного коду на сервер.

- Підбір паролів методом грубої сили. Злочинці можуть підібрати паролі до акаунтів користувачів, збитки і проблеми можуть бути колосальними. Захиститися від цього можна хешуванням паролів при реєстрації користувачів. Навіть якщо злочинці зможуть отримати доступ до бази даних, то там їх будуть очікувати паролі після процесу хешування, з якими вони не зможуть нічого зробити. Про хешування детальніше буде описано далі в цій роботі.
- Впровадження сценаріїв через завантаження файлів на сайті. Можливість користувача завантаження файлів на сайті може представляти загрозі безпеки веб-додатку. У завантаженому файлі може бути небезпечний сценарій, який може запуститися на сервері і створити багато проблем. Найлегшим способом є перевірка файлів по типу. Дозволяти відкривати лише зображення.
- Перехоплення даних. Злочинці можуть перехопити дані, поки вони будуть йти с клієнта на сервер або навпаки. Найкращим варіантом захисту буде шифрування даних. Якщо злочинці перехоплять зашифровані дані, то вони не будуть корисними для них. Про шифрування детальніше буде описано далі.

1.2 Шифрування

Шифрування - це спосіб приховування даних перестановкою елементів, щоб лише авторизовані користувачі мали можливість прочитати дані. Технічно, це процес зміни зрозумілого для людини відкритого тексту в незрозумілий текст, також відомий як зашифрований текст. Шифрування бере звичайні дані і перетворює їх так, щоб вони виглядали, як випадковий набір символів. Шифрування потребує використання криптографічного ключа: набору

математичних значень, узгоджених як відправником, так і одержувачем зашифрованого повідомлення.

При тому, що дані після процесу шифрування виглядають як випадковий набір символом, саме шифрування відбувається за певним алгоритмом, даючи можливість користувачеві, який володіє вірним ключем, провести зворотний процес шифрування - дешифрування і перетворити дані назад в зрозумілий людині текст. Щоб шифрування було безпечним використовуються складні ключі, при яких сторонні користувачі малоймовірно зможуть розшифрувати або зламати шифровані дані.

Головною метою шифрування є захист інформації при зберіганні чи передачі даних в незахищеній середі.

Надійність алгоритму шифрування залежить від його криптографічної стійкості. Криптографічна стійкість це властивість алгоритму протистояти криптографічному аналізу. Будь-який алгоритм шифрування може бути зламаний перебором усіх можливих ключів методом грубої сили. Виключаючи абсолютно крипостійкі алгоритми шифрування.

Метод грубої сили це процес, при якому зловмисник не знаючи ключа дешифрування, підбирає цей ключ велику кількість раз. Швидкість виконання цього підбору залежить від ресурсів комп’ютера. Більшість сучасних шифрів стійкі до методу грубої сили, але в майбутньому ці шифри можуть стати вразливими з ростом потужності комп’ютерів[1].

Абсолютно стійкий шифр – шифр, який не може бути розкритий як практично, так і теоретично при будь-якій потужності та кількості ресурсів користувача.

Для абсолютно стійких шифрів є декілька вимог:

- Для кожного повідомлення генерується свій ключ
- Довжина ключа повинна бути більшою або рівною довжини повідомлення
- Послідовність символів повністю випадкова

- Ймовірність появи кожного символу однакова
- Прикладом абсолютно стійких шифрів є шифр Вернама.
- Також існують достатньо стійкі шифри. Це шифри, які можуть бути потенційно зламані.

Оцінка криптографічної стійкості проходить в декілька етапів.

Етапи оцінки криптографічної стійкості:

- Початкова оцінка
- Поточна оцінка

Способи початкової оцінки базуються на обчислювальній складності, яка може бути виражена в часі, грошах або чомусь іншому, так як спочатку оцінюють метод грубої сили, тому що він можливий для усіх криптографічних алгоритмів, окрім абсолютно стійких та може бути єдиним існуючим.

Криптографічний ключ – це набір символів, який використовується в певному алгоритмі шифрування даних. Лише користувач з правильним ключем зможе розшифрувати ці дані.

Шифрування поділяється на симетричне та асиметричне. Асиметричне шифрування також називають шифруванням з відкритим ключем.

Процес шифрування забезпечує три головні категорії безпеки даних:

- Конфіденційність – категорія безпеки інформації, при якій доступ до даних мають лише власник та передбачуваний отримувач
- Цілісність – категорія безпеки інформації, яка гарантує, що дані не були змінені під час їх передачі.
- Доступність – категорія безпеки інформації, яка гарантує, що користувачі, які мають права доступу до даних, в будь-який момент часу зможуть переглянути ці дані.

1.2.1 Симетричне шифрування

Симетричне шифрування – тип шифрування, при використанні якого для шифрування та дешифрування інформації використовується лише один ключ,

його називають секретним. Користувачі при процесі симетричного шифрування повинні обмінюватися цим ключем для подальшого дешифрування інформації.

При використанні алгоритмів симетричного шифрування, інформація перетворюється в незрозумілий текст тому, в кого немає ключа для дешифрування. Отримавши повідомлення, користувач з ключем може дешифрувати його. Секретний ключ може бути буд-яким набором символів, які були згенеровані безпечним генератором випадкових чисел.

Є два різні типи симетричного шифрування:

- Блочний. Довжина бітів шифрується в певних блоках даних з використанням секретного ключа.
- Потоковий. Дані шифруються під час потокової передачі та не зберігаються в пам’яті.

Прикладами симетричного шифрування:

- AES
- DES
- 3DES
- IDEA

Симетричне шифрування є більш старим, ніж асиметричне. Також симетричне шифрування швидше й ефективніше. Через це симетричне шифрування використовується для шифрування дуже великих об’ємів інформації[2].

Також перевагою симетричного шифрування є конфіденційність даних, отримана без зайвих складнощів, при використанні декількох ключів.

Складність симетричного ключа залежить від довжини цього ключа.

Симетричне шифрування також має недоліки, найбільшою проблемою є аспекти управління ключами. До них входять:

- Ключове виснаження це проблема при якій кожне використання ключа може призвести до втрати важливої інформації, яка може допомогти встановити ключ.

- Відсутність ключових атрибутів. Відносно асиметричного шифрування ключ симетричного шифрування не має атрибутів, таких як дати закінчення строку дії чи параметри доступу.

Розглянемо симетричний шифр AES.

AES(Advanced Encryption Standard) – симетричний блоковий ітеративний алгоритм шифрування. AES це американський стандарт шифрування. Також AES дуже розповсюджений, є одним з найпопулярніших алгоритмів шифрування, використовується в багатьох сферах.

Цей алгоритм працює на архітектурі SQUARE, це означає, що алгоритм працює одночасно в двох напрямках, зі строками та стовпцями.

Для алгоритму AES характерно:

- Блоки, що шифруються в виді двомірного байтового масиву.
- За один раунд шифрується весь блок даних.
- Виконання шифрування як окремих байтів в масивах, так і його стовпців зі строками.

До загальних характеристик алгоритму шифрування Advanced Encryption Standard входять:

- Є можливість три різних по довжині ключа(128, 192 або 256 біт).
- AES може шифрувати та розшифровувати 128-бітні блоки інформації.
- Всі раунди операцій при шифруванні або розшифруванні однакові, окрім останнього.
- Кількість раундів залежить від довжини ключа. При ключі 128 біт буде десять раундів, 192 біти – дванадцять раундів, 256 біт – чотирнадцять раундів.

При переборі грубою силою потрібно 2^{127} операцій для ключа 128 біт, 2^{191} операцій для ключа 192 біти, 2^{255} для ключа в 256 біт. Навіть для найменшого ключа ця атака в наші дні не має значення на практиці, тобто AES можна вважати стійким[3].

Алгоритм шифрування AES є надійним проти таких видів атак:

- Лінійний криптографічний аналіз.
- Диференційний криптографічний аналіз.
- Криптографічний аналіз на основі зв’язаних ключів.

Розглянемо алгоритм роботи AES.

Важливі позначення:

- Cipher Key – секретний ключ, який потрібен при процедурі Key Expansion, щоб здобути раундові ключі.
- Key Expansion – процес генерації раундових ключів з Cipher Key.
- State – проміжний результат шифрування.
- S-box – таблиця замін, яка використовується в Key Expansion. S-box можна побачити на рисунку 1.1.
- Nb – кількість стовбців, які входять в State. В AES завжди 4.
- Nk – кількість 32-бітних слів в шифроключі. В AES 4, 6 або 8.
- Nr – кількість раундів. В AES 10, 12 або 14.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рисунок 1.1 – Таблиця S-box для Key Expansion

SubBytes() – процедура, яка обробляє кожний байт State, виконується нелінійна заміна байтів за допомогою S-box. Ця операція виконується за допомогою формули 1.1. Додатковими умовами до формули 1 є ($0 \leq i \leq 8$), b_i це і-ий біт b, c_i це і-ий біт константи c.

$$b_i' = b_i \; XOR \; b_{(i+4)} \; mod8 \; XOR \; b_{(i+5)} \; mod8 \; XOR \; b_{(i+6)} \; mod8 \; XOR \; b_{(i+7)} \; mod8 \; XOR \; c_i \quad (1.1)$$

ShiftRows() – процедура, яка працює зі строками State. Строки State залежно від номера строки по горизонталі циклічно зсуваються на r байт. Для нульової строки r = 0, для першої r = 1, і так далі.

MixColumns() – процедура, при якій кожний стовпчик State множиться на матрицю, яка показана на рисунку 1.2.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Рисунок 1.2 - Матриця, яка використовується в MixColumns

AddRoundKey() – процедура, при якій раундовий ключ об’єднується з State. Для цього відбувається XOR кожного байта State з кожним байтом RoundKey.

Шифрування відбувається в такій послідовності:

1. Відбувається генерація раундових ключів - KeyExpansion.
2. Відбувається AddRoundKey – складаються по модулю 2 проміжні масиви з RoundKey.

3. Всі раунди окрім останнього відбуваються в такій послідовності:
    a. SubBytes()
    b. ShiftRows()
    c. MixColumns()
    d. AddRoundKey()
4. Останній раунд відбувається в такій послідовності:
    a. SubBytes()
    b. ShiftRows()
    c. AddRoundKey()

Розшифрування AES відбувається в зворотному порядку.

Розглянемо симетричний шифр DES.

DES(Data Encryption Standard) – симетричний блоковий алгоритм шифрування.

В 1975 році NIST створила DES.

Загальна характеристика Data Encryption Standard:

- DES шифрує та розшифровує 64-бітні блоки інформації.
- Довжина ключа – 56 біт.
- Процес шифрування складається з двох раундів перестановок і шістнадцяти раундів Фейстеля.

Найбільшою проблемою DES є його довжина ключа, лише 56 біт. Для перебору грубою силою достатньо всього 2^{56} операцій. В наш час це дуже мало, це означає, що можна досить швидко зламати шифр.

В алгоритмі DES також є можливими слабкі ключі або напівслабкі ключі.

DES є вразливим до:

- Лінійний криптографічний аналіз(2^{43} операцій) за умови наявності великої кількості пар “відкритий текст - шифротекст”
- Диференційний криптографічний аналіз(2^{47} операцій) за умови наявності великої кількості пар “відкритий текст - шифротекст”

Вважається, що найпростішою атакою на шифр DES є перебір грубою силою, так як отримання великої кількості пар “відкритий текст - шифротекст” досить трудомісткий процес.

Розглянемо алгоритм роботи DES.

Генерація ключів відбуваються в такій послідовності:

1. Приймається 64-бітний ключ. Беруться 56 біт, інші 8 біт відкидаються.
2. Далі ключ ділиться на 2 частини по 28 біт.
3. Робиться перестановка.
4. Ключ розбивається на дві частини, C_0, D_0.
5. Знаходиться C_i, D_i за допомогою циклічного зсуву C_{i-1} та D_{i-1} на величину зсуву відповідно таблиці 1.1.
6. K_i обирається з C_iD_i. З 56 біт отримуємо 48 біт.

Таблиця 1.1 Величина зсуву

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Число здвигу	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Алгоритм шифрування відбувається в такій послідовності:

1. Розбиття вхідних даних на блоки по 64 біти. Якщо розмір вхідних даних не кратний 64, то дописати у масив, отриманий з вхідних даних необхідну кількість нулів.
2. Зробити початкову перестановку для кожного блоку.
3. Для кожного блоку запустити алгоритм Фейстеля.
4. Зробити кінцеву перестановку.

Алгоритм розшифрування майже ідентичний до алгоритму шифрування, але замість алгоритму шифрування Фейстеля, потрібно виконати алгоритм розшифрування Фейстеля.

Розглянемо порівняльний аналіз шифрів AES та DES.

Порівняння характеристик цих алгоритмів буде відображено в таблиці 1.2.

Таблиця 1.2 Порівняльний аналіз алгоритмів AES та DES

Назва	Довжина ключа, біт	Довжина блоку інформації, біт	Кількість операцій для злому грубою силою
AES	128, 192, 256	128, 192, 256(залежить від довжини ключа)	2^{127}, 2^{191}, 2^{255}(залежить від довжини ключа)
DES	56	64	2^{56}

Довжина ключа AES значно більша, ніж в алгоритмі DES. Це впливає на кількість операцій для злому грубою силою. З цього випливає, що AES є надійнішим, ніж DES.

DES є вразливим до лінійного криптографічного аналізу, диференційного криптографічного аналізу, в той час як AES не є вразливим до цих атак.

Алгоритм шифрування DES є ненадійним в наш час. AES навпаки, навіть з найкоротшим ключем у 128 бітів вважається надійним[4].

1.2.2 Асиметричне шифрування

Асиметричне шифрування – тип шифрування, який використовує два зв’язані між собою ключі, один публічний та один приватний. Публічним ключем можна лише шифрувати повідомлення, а приватним можна і шифрувати, і розшифровувати. Асиметричне шифрування також називають шифруванням з відкритим ключом. Розшифрувати повідомлення при асиметричному шифруванні може лише користувач з приватним ключом, через це приватний ключ потрібно зберігати в повній секретності.

Пару ключів в асиметричному шифруванні генеруються за допомогою спеціального алгоритму. Ці ключи математично зв’язані між собою в різних алгоритмах по різному.

Приклади асиметричного шифрування:

- RSA
- ElGamal
- DSA
- Diffie-Hellman

Переваги асиметричного шифрування:

- Можливість захисту даних та обміну ключами в публічних незахищених каналах
- Складність підбору ключів, так як асиметричне шифрування використовує більш довгі ключі, ніж симетричне. Зараз використовують ключі довжиною 1024 біт та більше.
- Є більш надійним, ніж симетричне
- Процес розподілення ключів, так як за безпеку відкритого ключа можна не перейматися

Недоліки асиметричного шифрування:

- Велика витрата ресурсів. Так як ключів два та їх довжина досить велика. Асиметричне шифрування використовує більш інтенсивно ЦП та має проблеми з продуктивністю.
- Процес шифрування та дешифрування повільніший, ніж в симетричному шифруванні. Так як алгоритми в асиметричному шифруванні складніші.

Не рекомендується шифрувати великі обсяги даних асиметрично, так як це збільшить навантаження на сервер.

Асиметрично шифрування часто використовується для встановлення безпечних каналів зв’язку, в яких надалі буде використовуватися симетричне шифрування.

Розглянемо асиметричний шифр RSA.

RSA(Rivest, Shamir та Adleman) – асиметричний криптографічний алгоритм. RSA має в основі задачу факторизації великих цілих чисел. RSA використовується як в шифруванні, так і в цифровому підписі.

Через великої складності злому RSA використовується в великій кількості сфер: покупки в інтернеті, обмін даними, банківські паролі, кабельне телебачення і так далі. Алгоритм RSA є найбільш використовуваний криптографічний алгоритм з відкритим ключом у світі та являється стандартом де факто.

Стійкість шифру RSA вимірюється довжиною ключа. 512-бітний ключ вже не вважається надійним, так як методом грубої сили в наш час можуть добувати закриті ключи всього за декілька годин. 768-бітні та 1024-бітні ключі також вважаються не дуже надійними, більшість зараз використовують ключи довжиною 4096 біт[5].

В алгоритмі шифрування RSA слабким місцем може бути не дуже гарно обрана пара простих чисел p і q або відкрита експонента.

Головним недоліком алгоритму RSA є використання великої кількості ресурсів на пристрої, де виконується шифрування або розшифрування.

Розглянемо алгоритм роботи RSA[6].

Генерація ключів відбувається в такій послідовності:

1. Спочатку обираються два простих числа p та q.
2. Потім розраховується модуль ключа n, як добуток p та q.
3. Розраховується функція Ейлера за формулою 1.2, за якої ми отримуємо значення F(n).
4. Обирається відкрита експонента e, яка є частиною відкритого ключа. Відкрита експонента повинна бути взаємно простою з F(n).
5. Визначається закрита експонента d, як мультиплікативно зворотну до відкритої експоненти e. Тобто потрібно знайти таке d, щоб виконувалася умова формули 1.3.

6. Пара {e, n} – відкритий ключ.
7. Пара {d, n} – закритий ключ.

$F(n) = F(p) * F(q) = (p-1)(q-1)$(1.2)

$d * e = 1(mod\big(F(n)\big))$(1.3)

Шифрування відбувається в такій послідовності:

1. Береться повідомлення m та відкритий ключ {e, n}.
2. Визначити шифротекст C за формулою 1.4.

$C = me(mod\ n)$(1.4)

Розшифрування відбувається в такій послідовності:

1. Приймається шифротекст C.
2. Береться закритий ключ {d, n}.
3. Отримується вхідний текст за формулою 5.

$m = Cd(mod\ n)$(1.5)

Розглянемо асиметричний шифр ElGamal.

ElGamal це криптосистема з відкритим ключом, ключ шифрування можна публікувати, а ключ дешифрування потрібно зберігати в тайні. Оскільки шифрування да дешифрування це зворотні процеси, вони повинні бути зв’язані математично.

Надійність криптосистем з відкритим ключом залежить від того, наскільки складно здобути приватний ключ із значення публічного ключа.

В основі шифру ElGamal лежить важка задача обчислення дискретних логарифмів в кінцевому полі. Це дозволяє без пониження стійкості швидко генерувати ключи. Також цей шифр в свій алгоритм включає алгоритм обміну ключами Діффі-Хелмана.

Переваги шифру ElGamal:

- Шифр є ймовірним, це означає, що один й той самий текст в зашифрованому вигляді може виглядати зовсім по різному. Це робить його безпечнішим.

- Шифр використовується в багатьох програмах для шифрування та цифрового підпису. Наприклад в GNU Privacy Guard та PGP.

Недоліки шифру ElGamal:

- Необхідність в випадковості, через це менша швидкість операцій, особливо при електронному підписі.
- Через те, що шифр ймовірний, розмір зашифрованого тексту вдвічі більший, ніж вхідного. Процес розширення вдвічі відбувається в процесі шифрування, що також зменшує його швидкодію.

Розглянемо алгоритм роботи ElGamal.

Генерація ключів відбувається в такій послідовності:

1. Обирається випадкове просте число p.
2. Обирається ціле число g, який є первісним коренем числа p.
3. Обирається випадкове ціле число x, яке має умову (1 < x < p – 1)
4. Розраховується число y за формулою 1.6.
5. Відкритий ключ – {y, q, p}, закритий ключ – {x}.

$y = g^x \ mod \ p$(1.6)

Шифрування відбувається в такій послідовності:

1. Отримується повідомлення M, яке повинно бути меншим числа p.
2. Обирається випадкове число k взаємно просто з (p – 1), яке має умову (1 < k < p – 1). k це сесійний ключ.
3. Число a обчислюється за формулою 1.7, число b обчислюється за формулою 1.8.
4. Пара чисел (a, b) і є шифротекстом.

$a = g^k \ mod \ p$(1.7)

$b = \ y^k M \ mod \ p$(1.8)

Розшифрування відбувається в такій послідовності:

1. Береться приватний ключ x.
2. Отримаємо відкритий текст за формулою 1.9.

$M = b(a^x)^{-1} \ mod \ p = b * a^{(p-1-x)} \ mod \ p$(1.9)

Розглянемо порівняльний аналіз шифрів RSA та ElGamal.

Алгоритм RSA має кращу швидкість шифрування. Так як ElGamal є ймовірним і довжина зашифрованого тексту вдвічі більша, ніж звичайного. У шифрі RSA цей показник значно менший, приблизно в 1.4 рази зашифрований текст більший, ніж вхідний.

Алгоритм ElGamal розшифровує текст швидше при використанні великих ключів. Так як при збільшенні ключів час розшифрування ElGamal зростає лінійно, а у шифрі RSA час розшифрування зростає експоненціально.

Криптографічна стійкість в цих алгоритмів є однакова. RSA та ElGamal мають $2.7 * 10^{28}$ MIPS для ключів розміром 1300 біт.

Також перевагою RSA є чітко описаний стандарт, який є безкоштовним. Для ElGamal такого стандарту взагалі немає.

1.3 Хешування

Хешування – перетворення одного значення в зовсім інше значення внаслідок використання деякого математичного алгоритму. Результатом хешування є хеш. Незалежно від того яка буде довжина вхідного значення, після операції хешування ми отримаємо значення сталого розміру. Розмір вихідного значення для кожного алгоритму буде свій.

Зробити зворотну операцію та перетворити хеш назад в текст майже неможливо. Для кожного тексту генерується свій унікальний хеш. Внаслідок незначної зміни текстового значення, хеш в результаті буде виглядати зовсім по іншому[8].

Але для кожного хеш алгоритму можливі колізії. Колізія хеш-функції це однакові вихідні хеші для різних вхідних даних. Для нормальних хеш-функцій шанс виникнення колізії неймовірно малий.

Усі хеш-функції є детермінованими. Це означає, що хешування одного й того ж тексту завжди буде видавати одне й те саме значення, доки текст не зміниться.

Хешування потрібно для цілісності даних. Хешування використовується для різних цілей. Таких як зберігання паролів, зберігання даних в базах даних або комп’ютерних системах. Загалом є багато алгоритмів хешування, кожний алгоритм має свої переваги та недоліки. Деякі алгоритми хешування оптимізовані для швидкої обробки даних або для максимальної безпеки даних, інші оптимізовані для спеціальних типів даних[7].

Приклади хешування:

- SHA-0
- SHA-1
- SHA-256
- MD5

Також хешування використовується в SSL сертифікатах. SSL сертифікат потрібен для того, щоб злодії не мали змоги перехопити особисті дані, які користувач використовує на сайті. Коли SSL сертифікат підписується, підпис проходить алгоритм хешування і цей хеш входить до деталей сертифікату. Коли клієнт заходить на сайт, він виконує алгоритм хешування для того самого підпису і порівнює хеші. Якщо хеші однакові, це показує, що підпис не підроблений.

Розглянемо SHA-1.

SHA-1 це алгоритм криптографічного хешування, який реалізує хеш-функцію. Ця функція виконує ідею стискання. SHA розшифровується як Secure Hash Algorithm[9].

Результатом хешування SHA-1 завжди буде хеш довжиною сорок символів.

В 1993 році NSA та NIST створили алгоритм шифрування SHA-0, але знайшли в ньому помилку та вже через два роки створили новий алгоритм, виправивши цю помилку. Новий алгоритм отримав назву SHA-1.

SHA-1 працює за таким алгоритмом:

1. Вхідний текст розбивається на символи та перетворюється кожний символ у ASCII коди.
2. Перетворюється ASCII коди з попереднього кроку у двійкові та додається попереду кожного стільки нулів, щоб загалом стало вісім символів.
3. Об’єднується усі двійкові символи та додається у кінець 1.
4. Додається у кінець нулі, доки довжина двійкового числа не буде 512 mod 448.
5. Береться результат другого кроку, об’єднуються ці двійкові числа та результатом є довжина нового двійкового числа.
6. Довжина з попереднього кроку перетворюється в двійкове число та додається попереду стільки нулів, щоб довжина цього числа вийшла 64.
7. Результат попереднього кроку приєднується у кінець результату четвертого кроку. Загальна довжина цього двійкового числа повинна бути 512.
8. Попереднє двійкове число розбивається на двійкові числа по 32 біти.
9. За допомогою побітової операції “циклічний здвиг наліво” шістнадцять 32-бітних слів перетворюються в вісімдесят 32-бітних слів.
10. Створюється п’ять 32-бітних змінних. За допомогою побітових операцій проходиться кожне з вісімдесяти 32-бітних слів з дев’ятого пункту та перепризначуються ці п’ять змінних.
11. Додаються початкові значення п’яти змінних з попереднього кроку до значень після побітових операцій.
12. Перетворюються ці 32-бітні змінні з результату попереднього пункту в шістнадцятирічну систему.
13. Після цього вони об’єднуються та отримується хеш довжиною в сорок символів.

Результатом цього алгоритму для тексту “sha” є d8f4590320e1343a915b6394170650a8f35d6926.

Розглянемо SHA-256.

SHA-256 це різновид SHA-2. Це односпрямована хеш-функція, яка потрібна для створення дайджестів для повідомлень будь-якої довжини[10].

Результатом хешування SHA-256 завжди буде хеш довжиною шістдесят чотири символи.

В 2002 році NSA створили алгоритм хешування SHA-2 на заміну алгоритму SHA-1. А вже в 2012 році NSA додали до сімейства SHA-2 алгоритм SHA-256.

SHA-256 працює за схожим алгоритмом до SHA-1, але більш складним.

SHA-256 за таким алгоритмом:

1. Вхідний текст розбивається на символи та перетворюється кожний символ у ASCII коди. Перетворюється у двійкові числа та додається попереду кожного стільки нулів, щоб загалом стало вісім символів.
2. Об’єднується усі двійкові символи та додається у кінець 1. Додається у кінець нулі, доки довжина двійкового числа не буде 512 mod 448.
3. Береться результат першого кроку, об’єднуються ці двійкові числа та результатом э довжина нового двійкового числа.
4. Довжина з попереднього кроку перетворюється в двійкове число та додається попереду стільки нулів, щоб довжина цього числа вийшла 64.
5. Результат попереднього кроку приєднується у кінець результату четвертого кроку. Загальна довжина цього двійкового числа повинна бути 512.
6. Попереднє двійкове число розбивається на двійкові числа по 32 біти.
7. За допомогою побітової операції “циклічний здвиг наліво” шістнадцять 32-бітних слів перетворюються в вісімдесят 32-бітних слів.

8. Створюється вісім 32-бітних змінних. За допомогою побітових операцій проходиться кожне з вісімдесяти 32-бітних слів з дев’ятого пункту та перепризначуються ці вісім змінних.
9. Додаються початкові значення п’яти змінних з попереднього кроку до значень після побітових операцій.
10. Перетворюються ці 32-бітні змінні з результату попереднього пункту в шістнадцятирічну систему.
11. Після цього вони об’єднуються та отримується хеш довжиною в шістдесят чотири символи.

Результатом цього алгоритму для тексту “sha” є d600474b1b8e50d3633c91c0cf1efc454b79c9624a43fd7de441ee71745726ab.

Розглянемо порівняльний аналіз SHA-1 та SHA-256.

Ці алгоритми досить схожі між собою, так як SHA-256 є модифікацією SHA-1[11]. Порівняння цих алгоритмів буде відображено в Таблиці 1.3.

Таблиця 1.3 Порівняльний аналіз хешувань

Назва	Розмір вихідного хешу(біт)	Знайдені колізії	Потрібно операцій для перебору грубою силою
SHA-1	160	2^{52} операцій	2^{160}
SHA-256	256	-	2^{256}

Розмір вихідного хешу SHA-256 більша, ніж у SHA-1.

Для SHA-256 колізій ще не було знайдено, на відміну від SHA-1. Ймовірність знайти колізію у алгоритмі хешування SHA-1 дорівнює 2^{52} операцій.

Криптографічний аналіз цих алгоритмів хешування відносно перебору грубої сили показує, що SHA-256 значно надійніший.

З цього можна зробити висновок, що SHA-256 є більш надійним в наші дні[12].

1.4 Постановка задачі

Метою випускної роботи є розроблення веб-додатку для обміну повідомленнями в реальному часі з шифруванням повідомлень та хешуванням паролів.

Головною особливістю додатку є шифрування усіх повідомлень, що робить його листування в ньому досить безпечним.

Веб-додаток повинен мати такі сторінки:

- Сторінка авторизації, яка потрібна для входу в систему.
- Сторінка реєстрації для створення аккаунтів користувачів. Під час процесу реєстрації повинно бути враховано хешування паролів.
- Головна сторінка для створення нових кімнат для листувань. Також на цій сторінці повинна бути можливість приєднатися до вже існуючих кімнат.
- Сторінка чату. На цій сторінці повинен бути головний функціонал всього додатку, а саме обмін повідомленнями у реальному часі.

2 ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

Для демонстрації алгоритмів захисту веб-додатків розробимо додаток для обміну повідомленнями шифруванням даних та хешуванням паролів. Веб-додаток буде мати закриті кімнати, до яких зможуть зайти користувачі, в яких є запрошення.

Серверною (Backend) частиною додатку буде програмна платформа Node.js, а саме фреймворк Express.js на мові JavaScript.

Клієнтською (Frontend) частиною додатку буде виступати ReactJS. Це UI-бібліотека від Facebook з відкритим кодом. Для оптимізації роботи з даними буде використовуватися Redux.

Для реалізації бази даних я обрав нереляційну СУБД MongoDB, яка є найпопулярнішою нереляційною СУБД.

Для обміну даних між клієнтською та серверною частиною буде використовуватися і REST API, і WebSocket API.

Клієнтська та серверна частина розгорнуті на платформі Heroku.

Дані шифруватися будуть за допомогою шифрів AES та RSA, паролі будуть хешуватися алгоритмом SHA-256.

2.1 Обрані технології

Розглянемо усі технології, які будуть використовуватися в процесі розробки додатку більш детально.

2.1.1 Node.js

Node.js це програмна платформа, яка працює на движку V8. Движок V8 – движок JavaScript з відкритим початковим кодом, який був розроблений Google. Node.js перетворює із вузькоспеціалізованої мови в мову загального використання. Ця платформа дозволяє мові програмування JavaScript

взаємодіяти з пристроями введення-виведення через свій API, підключати інші бібліотеки.

Платформа Node.js, яка написана на C++, є однією з найпопулярніших платформ для розробки ефективних та масштабних REST API в наші дні. Вона також підходить для побудови мобільних додатків, десктопних програм і навіть для Internet of things[13].

Переваги Node.js:

- Головною перевагою Node.js є система введення-виведення, яка не блокує. Ця система керується подіями і виконується асинхронно, будуючи чергу подій за пріоритетністю. Коли до серверу підключається одночасно багато людей, цій системі набагато легше подолати навантаження, так як не потрібно створювати окремий потік для кожного підключення. Розподіл ресурсів допомагає витримати більшу кількість з'єднань.
- Можливість використовувати одну й ту саму мову програмування, як на клієнті, так і на сервері. Так як на ринку розробки клієнтської частини веб-додатків JavaScript є повним монополістом, то вивчити надбудову над мовою легше, ніж повність нову технологію.
- Швидкість написання серверної частини. Створення робочого прототипу, який буде витримувати навантаження, не відніме багато часу. Перше налаштування додатку відбувається дуже швидко.
- API Node.js. До API входить дуже багато корисних пакетів, які можна використовувати одразу без завантаження. Прикладами таких пакетів є OS(працює з операційною системою приладу, де виконується код), Crypto(має багато криптографічних функцій, які можна швидко використати), FS(виконує різні операції з файлами) і так далі[14].

- Зручність використання сторонніх пакетів. У Node.js входить пакетний менеджер npm, за допомогою якого досить легко шукати та встановлювати різні пакети, розроблені іншими користувачами.
- Node.js стрімко поліпшується. Над Node.js працюють багато програмістів по всьому світу. Процес координує фонд Node.

Недоліки Node.js:

- Головною проблемою Node.js є низька продуктивність при роботі з великими і тяжкими обчислювальними задачами. Так як Node.js має лише один потік, то отримавши задачу, для вирішення якої потрібно багато ресурсів процесору, Node.js використовує всі можливі ресурси процесору, а інші задачі повинні чекати, доки ця задача виконається. Виникає так звана черга.
- Іноді погана читаність коду. Node.js побудований на функціях зворотного виклику, які викликаються після завершення певної задачі в черзі. Збереження ряду задач в правильній черзі може сильно ускладнити розуміння коду.

Розглянемо фреймворк Express.js, який працює на платформі Node.js.

Express використовується досить довго для розробки додатків і завдяки своїй стабільності посів позицію одного із найпопулярніших фреймворків Node.js[15].

Для Express.js характерний невеликий обсяг базового функціоналу. Всі інші необхідні можливості і функції потрібно бути отримувати за допомогою зовнішніх модулів. В чистому вигляді Express це сервер і у нього може не бути жодного модуля. Завдяки цьому розробник отримує швидкий і легкий інструмент, який він може розвивати і розширювати. Також важливим зауваженням є, що вибір модулів не має обмежень: ні с функціональними, ні с кількісними. Express дає змогу вирішувати будь-які задачі, не обмежуючи користувача в виборі. Також це означає, що кожне створене рішення буде унікальним, але на нього потрібно буде більше часу.

Переваги Express.js:

- Гнучкість
- Гарна масштабованість
- Детальна документація
- Широкий вибір модулів
- Простота

Недоліки Express.js:

- Великий об’єм ручної роботи
- Використання застарілого підходу callback functions

Я вважаю, що Express.js на платформі Node.js є ідеальним вибором для мого додатку.

2.1.2 MongoDB

MongoDB це популярна система управління базами даних з відкритим вхідним кодом, яка орієнтована на документи. Для зберігання даних в цій системі використовується JSON-подібний формат. JSON – текстовий формат для обміну даних, який базується на мові програмування JavaScript. Головними характеристиками СУБД MongoDB є масштабованість, безпека та доступність[16].

Головні особливості MongoDB:

1. Ця СУБД не вимагає опису схем таблиць, як це робиться в реляційних базах даних. Дані зберігаються в виді документів та колекцій.
2. В MongoDB немає складних для розуміння з’єднань типу RIGHTJOIN, LEFTJOIN та інших. Зазвичай з’єднання йде шляхом об’єднання документів при збереженні даних.
3. Це кроссплатформенна NoSQL база даних з відкритим кодом, яка працює з документами.

4. У різних колекцій не обов’язково повинна бути схожа структура. У них може бути зовсім різний набір полів. Як типом, так і кількістю.
5. В цій СУБД дані зберігаються у вигляді BSON. BSON це бінарні JSON-подібні документи.

Розглянемо структуру сховища MongoDB.

Порівняємо основні терміни MongoDB і порівняємо їх з термінами мови структурованих запитів(SQL):

- Колекція – група документів MongoDB. Це називають таблицею в мові SQL.
- База даних це фізичний контейнер для колекції.
- Документ – набір пар ключ-значення, запис в колекції. Це рядок на мові SQL.
- Поле – ключ в документі. Схоже на стовпець в SQL.

Переваги MongoDB:

- База, що працює з документами. Це означає, що в цій СУБД дані зберігаються в виді документів замість форматів реляційного типу. Ця перевага адаптує MongoDB до бізнес-вимог и робить її дуже гнучкою. Також є важливим можливість роботи з різними типами даних у випадку з використанням великої кількості інформації, яка береться з різних джерел.
- В цій СУБД доступна індексація. Це означає, що в користувача є можливість створити індекс для будь-якого поля в документі, щоб сильно пришвидшити пошук в цьому документі.
- MongoDB підтримує спеціальні запити. Є підтримка пошуку по полям, пошук по регулярним виразам, пошук по діапазонам. Можна зробити запити для повернення визначених полів с документу.
- Використовується балансування навантаження. MongoDB використовує розподіляє дані між кількома екземплярами БД. СУБД може працювати на кількох серверах задля підтримки

працездатності в випадку апаратних збоїв або балансувати навантаження.

- Цю СУБД можливо розгорнуті на хмарному сховищі. Можна отримати готову до роботи, з оптимальною конфігурацією, базу даних всього за декілька хвилин.
- Доступність. MongoDB підтримує усі досить популярні мови програмування і її можна використовувати безкоштовно.
- Створення копій. MongoDB зберігає одразу декілька копій. Якщо щось трапиться з головною, то вона заміниться на запасну.

Недоліки MongoDB:

- Виконувати транзакції з використанням цієї СУБД дуже важко.
- MongoDB не підпадає під норми ACID(атомарність, узгодженість, ізольованість, стійкість). Реляційні бази даних підпадають під ці норми.

Так як в моєму додатку не буде ніяких транзакцій, я вважаю, що MongoDB гарний варіант.

2.1.3 ReactJS

React – це бібліотека, яка написана на мові програмування JavaScript, для написання користувацьких інтерфейсів чи компонентів користувацьких інтерфейсів. Ця бібліотека має відкритий код. React був створений і підтримується компанією Facebook.

Цю бібліотеку можна використовувати, як при розробці веб-сайтів так і при розробці мобільних додатків[17].

React реалізує концепт Single Page Application. SPA це односторінкові додатки, які для оновлення контенту, який динамічно змінюється, не потребують оновлення сторінки. Це покращую покращує швидкість праці веб-додатку, а також покращує враження користувача під час використання цього

додатку. Також можна додати, що SPA та WebSocket API найкраща комбінація для реалізації веб-додатку с обміном повідомленнями між користувачами.

Головною ідеєю ReactJS є розділення користувацьких інтерфейсів на окремі компоненти з метою подальшого перевикористання. Також однією з головних особливостей цієї бібліотеки є надбудова мови програмування JavaScript – JSX. Весь React побудований на JSX. За допомогою цієї надбудови у користувача є можливість писати розмітку прямо в коді JavaScript, що сильно пришвидшує розробку. React, використовуючи JSX будує власне віртуальне DOM-дерево для відображення контенту на сторінці.

Відносно конкурентів AngularJS та VueJS, DOM-дерево в ReactJS працює значно швидше, шо значно зменшує час, який потрібний на відображення контенту на веб-сторінці.

Переваги ReactJS:

- Ця бібліотека досить легка в навчанні. React можна значно швидше вивчити, ніж його конкурента AngularJS.
- React може легко працювати при великих навантаженнях з використанням ES6-ES10 можливостях. ES це EcmaScript, нові версії JavaScript, в яких є багато нових можливостей.
- Неймовірна підтримка від Facebook та сторонніх користувачів. Так як код у цієї бібліотеки є відкритим, то оновлення виходять кожного дня.
- Дуже висока швидкодія.
- Легка міграція між версіями. Facebook надає codemods для автоматизації цих процесів.

Недоліки ReactJS:

- Недостатня кількість офіційної документації. Так як розробка React зараз йде дуже швидко, документація іноді може виглядати дуже хаотично.

- Проблема менеджменту стану даних в великих проектах. Цю проблему легко вирішує стороння дуже велика бібліотека для JavaScript – Redux. В своєму додатку я і використовував саме цю бібліотеку.

Так як ми визначилися, що для чату нам потрібен SPA, то ReactJS нам дуже добре підійде.

2.1.4 WebSocket API

WebSocket – технологія, яка дає можливість створювати двостороннє підключення між сервером і клієнтом для обміну повідомленнями в реальному часі.

WebSocket це протокол прикладного рівня, отже на транспортному рівні він базується на TCP. TCP в даному випадку забезпечує зв’язок у двох напрямках.

WebSocket API прийшло на заміну так званого методу “polling”. Це метод, при якому клієнтська частина додатку періодично запитувала дані з серверу. Досить очевидним мінусом підходу “polling” є велике навантаження на клієнтську та серверну частину додатку. Також проблемою цього підходу є лише ілюзія отримання даних в реальному часі. Тож так званий “полінг” є поганим варіантом[18].

Для встановлення з’єднання з використанням WebSocket API, клієнту потрібно прислати запит на зміну протоколу. Якщо з’єднання пройшло успішно, сервер поверне статус HTTP 101 Switching Protocol. Після цього протокол змінюється на WebSocket та відкривається канал для обміну повідомленнями. Зміна протоколу відображена на рисунку 2.1.

```
HTTP/1.1 101 Switching Protocols
Connection: upgrade
Upgrade: websocket
Sec-WebSocket-Accept: dka7KhbVhjjw5aZ
+musp06qV2dA=
Sec-websocket-Extensions: permessage-deflate
```

Рисунок 2.1 - Приклад зміни протоколу на WebSocket

2.1.5 AES та RSA, SHA-256

Алгоритми шифрування AES, RSA та алгоритм хешування SHA-256 я детально розібрав в теоретичній частині. В цьому розділі я поясню, чому я обрав саме ці алгоритми.

Щодо хешування паролів в наш час зрозуміло, що у SHA-2 немає конкурентів. В теоретичній частині я порівнював SHA-2 і SHA-1, по всім пунктам SHA-2 виграє у SHA-1. SHA-256 це один із підвидів SHA-2, якого будить досить, щоб захешувати паролі в моєму додатку.

Ми маємо чат, в якому нам потрібно шифрувати усі повідомлення, а потім їх розшифровувати. При масштабуванні додатку, кількість повідомлень може дуже сильно збільшитися. Виходячи з цього вибір падає на симетричне шифрування, якому потрібно значно менше ресурсів, ніж шифруванню з відкритим ключем. Але якщо шифрувати повідомлення симетрично, то ключ потрібно передавати разом з повідомленнями, це може призвести до проблем. Якщо злочинці перехоплять повідомлення разом із ключем, то вони легко зможуть розшифрувати повідомлення. Отже я вирішив асиметрично шифрувати ключ, який ми отримаємо при симетричному шифруванні. І передавати зашифрований ключ з зашифрованими повідомленнями.

Отже в моєму алгоритмі буде потрібно використовувати, як симетричне, так і асиметричне шифрування.

Щодо симетричного шифрування вибір тут очевидний і він падає на AES. Так як це загальний стандарт шифрування і в нього немає конкурентів.

Щодо асиметричного шифрування я обрав шифр RSA. В RSA є конкурент ElGamal. Але я обрав саме RSA через те, що він шифрує дані значно швидше, а також він має загальний стандарт, який може стати розробнику у нагоді.

Детальніше алгоритм шифрування повідомлень я опишу в одному з наступних розділів

2.2 Підведення підсумків щодо стеку технологій

Відносно технологій, що я обрав, план їх використання у моєму додатку виглядатиме наступним чином:

- Клієнтська частина додатку написана на ReactJS + Redux і буде займатися відображенням контенту, отриманням повідомлень та надсиланням повідомлень на сервер. Клієнтська частина буде розгорнута на буде розгорнута на платформі Heroku.
- WebSocket API буде виступати з’єднувальною ланкою між клієнтською частиною та серверною частиною для обміну повідомленнями.
- Серверна частина додатку написана на платформі Node.js з використанням фреймворку Express.js і буде створювати користувачів, створювати кімнати, обробляти та записувати повідомлення. Серверна частина також буде розгорнута на платформі Heroku.
- База даних, побудована на основі СУБД MongoDB, буде зберігати усі дані додатку.
- Алгоритм SHA-256 буде хешувати паролі користувачів при реєстрації. Алгоритми RSA та AES будуть відповідати за шифрування та розшифрування повідомлень у кімнатах.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Опис алгоритму шифрування повідомлень

В алгоритмі шифрування повідомлень я буду використовувати шифри AES та RSA.

В моєму веб-додатку для обміну повідомленнями в реальному часі може бути велика кількість повідомлень, отже шифрувати всі повідомлення асиметрично дуже невигідно відносно витрати ресурсів. Представлений алгоритм шифрування даних використовує симетричне шифрування за допомогою AES . Так як ми маємо передачу даних з клієнта до серверу і навпаки, то щоб зашифрувати та розшифрувати дані, зашифровані симетрично, нам потрібен ключ. Отже нам потрібно цей ключ передати з серверу до клієнта. Це небезпечно, так як в разі перехвату трафіку, буде викрадений ключ та повідомлення, які можна буде легко розшифрувати.

Щоб зашифрувати або розшифрувати повідомлення на клієнті, нам потрібен ключ AES. Цей ключ отримується за таким алгоритмом:

1. На клієнті створюється пара ключів за алгоритмом RSA(асиметричне шифрування).
2. Створений публічний ключ відправляється до серверу.
3. На сервері створюється ключі AES(симетричне шифрування).
4. Створений ключ AES шифрується отриманим публічним ключем RSA та відправляється на клієнт.
5. Отриманий зашифрований ключ AES на клієнті ми розшифровуємо приватним ключем RSA.

Для більшої наглядності алгоритм показаний на рисунку 3.1.

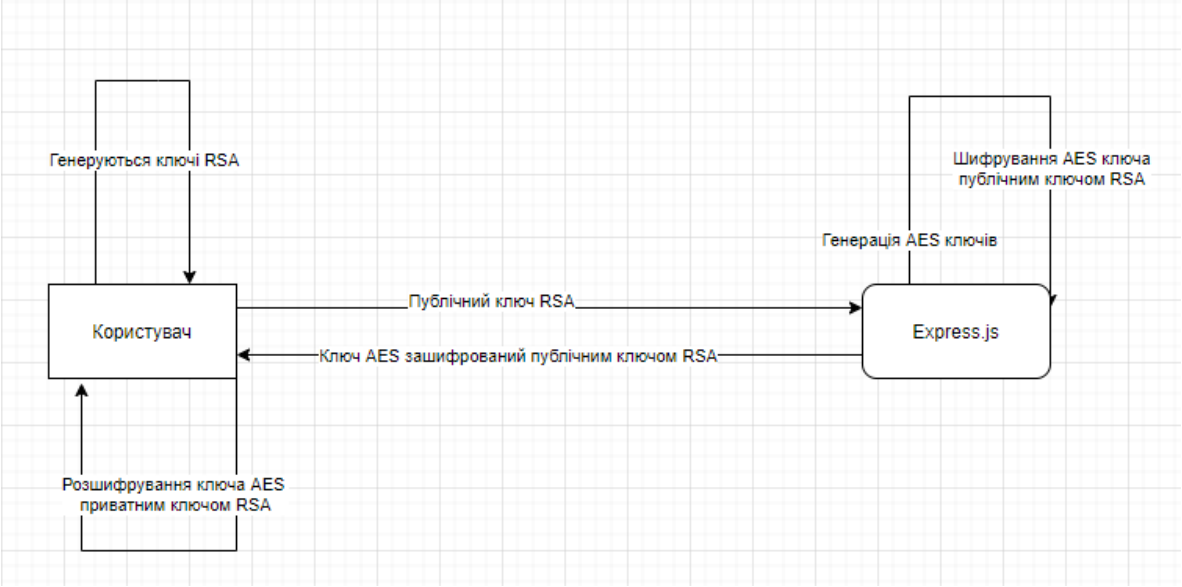

Рисунок 3.1 - Алгоритм генерації ключа для шифрування

Маючи ключ AES можна легко зашифровувати та розшифровувати повідомлення в будь-який момент часу.

3.2 Моделювання додатку

Розглянемо ключові моменти моделювання та планування окремо.

3.2.1 Моделювання бази даних

Моделюючи структуру бази даних для веб-додатку, головним функціоналом якого є обмін повідомленнями в реальному часі, в першу чергу потрібно проаналізувати усі сутності, які будуть у базі даних.

Так як мій додаток є чатом, там можуть обмінюватися повідомленнями користувачі. Це означає, що першою сутністю буде сам користувач. Користувача потрібно мати власну ідентифікація для того щоб авторизуватись у додатку. Для цього користувачу потрібно мати email та пароль. Також для створення і відображення автора повідомлення, користувачу потрібно мати username.

Далі розглянемо сутність кімнати чату, в яку можуть зайти лише певні користувачі, які були додані до білого списку кімнати при її створенні. Отже сутність кімнати повинна мати список користувачів, які можуть увійти до кімнати. Також користувач повинен потрапляти до кімнати певним чином. Для

цього в сутності кімнати буде унікальний ідентифікатор id. Ввівши цей ідентифікатор, користувач зможе увійти до кімнати.

Наступною сутністю буде сутність повідомлення. Саме повідомленнями будуть обмінюватися користувачі у кімнатах. Перш за все сутності повідомлення потрібен контент, а саме текст, який буде відображатися користувачам. Далі потрібно додати відправника, щоб відобразити його біля повідомлення. Також потрібно не забути про дату відправлення повідомлення, яка теж відобразиться біля повідомлення. Дату ще буде потрібно для поділення повідомлень по дням, коли ці повідомлення були написані.

Останньою сутністю буде абстрактна допоміжна сутність головного меню, щоб краще продемонструвати зв’язки.

Структура бази даних буде виглядати, як на рисунку 3.2.

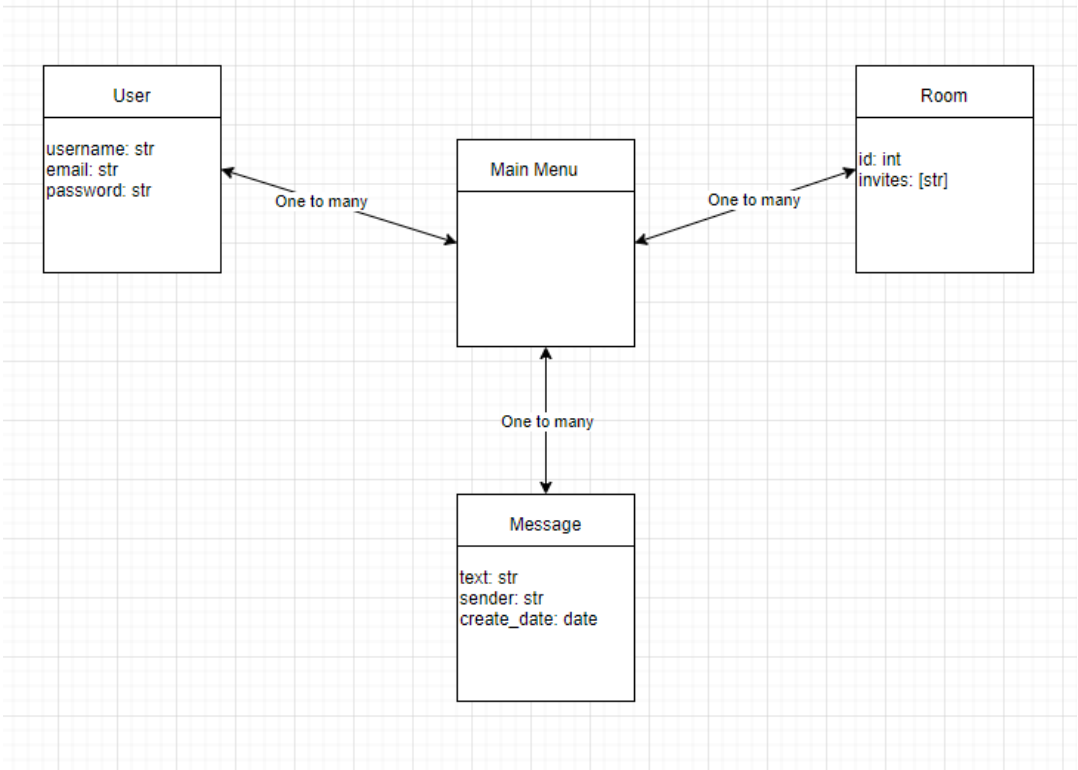

Рисунок 3.2 - Структура бази даних

3.2.2 Моделювання подій

У цьому розділі потрібно змоделювати та описати усі події, які будуть виконуватися під час роботи веб-додатку.

1. Реєстрація. Найпершою подією, яка буде відбуватися на веб-сайті буде реєстрація користувачів. Це потрібно для подальшою авторизації користувачів та для відображення автора біля відправленого повідомлення. Моделювання зображене на рисунку 3.3. Оскільки при події реєстрації не потрібне двостороннє з'єднання, то буде досить використати REST API ендпоінт.

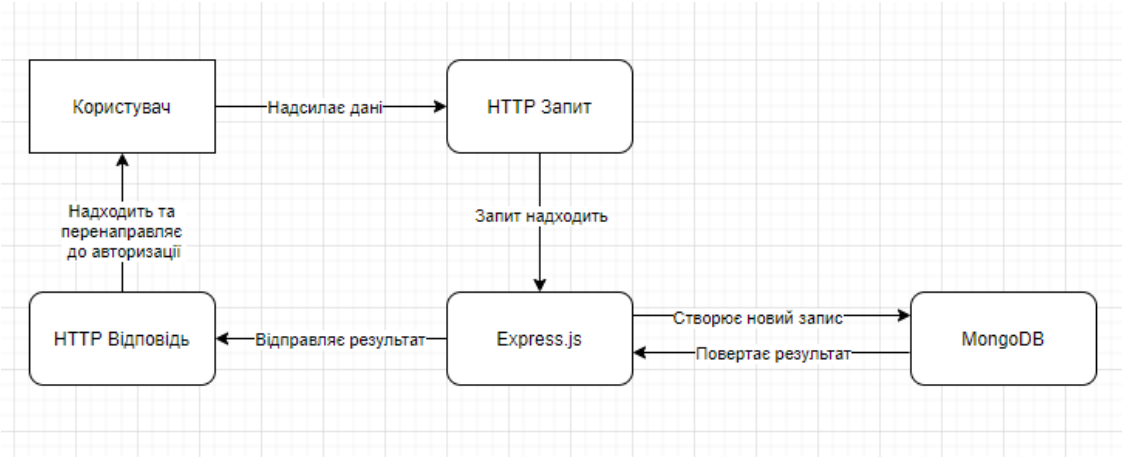

Рисунок 3.3 - Моделювання події реєстрації

2. Авторизація або іншими словами вхід до системи. Ця процедура означає потрапляння до системи користувача з його ідентифікацією. При даному сценарію також буде достатньо одного REST API ендпоінту, в який потрібно буде передати дані у форматі JSON.
3. Створення кімнати. Звичайна процедура, яку може виконати будь-який користувач у головному меню, якщо пройде процедуру авторизації. Для створення кімнати також достатньо лише одного REST API ендпоінту, в який потрібно буде передати email користувачів, яким буде дозволено увійти в цю кімнату. Після створення кімнати, користувача одразу перенаправить до цієї кімнати.
4. Перевірка користувача на можливість перебувати у певній кімнаті. Після заходження користувача в певну кімнату, потрібно перевірити, чи

є в нього права на знаходження в даній кімнаті. Для цього відправляється POST запит, в який передається номер кімнати і email людини, яка зайшла в кімнату.

5. Підключення до кімнати. Для створення двостороннього підключення потрібно створити підключення на клієнті та зареєструвати це підключення на сервері. Для підключення до кімнати користувач робить запит зміни протоколу на WebSocket до серверу і очікує відповідь. Після успішного підключення користувач змінює свій протокол на WebSocket та підключається до кімнати. Процес підключення зображений на рисунку 3.4.

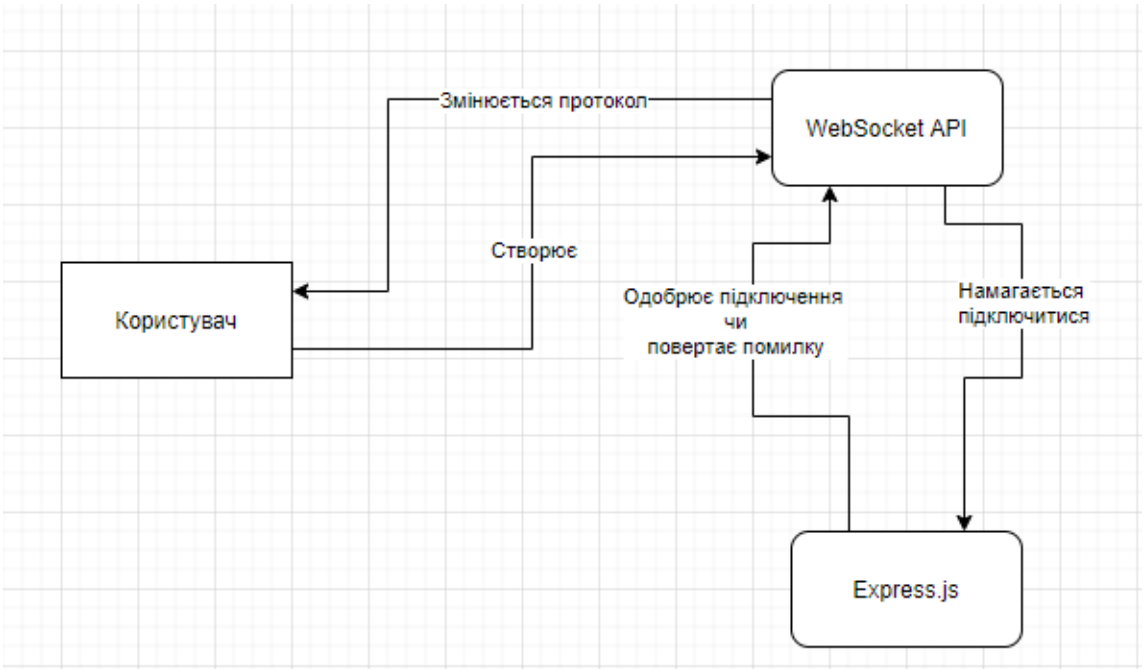

Рисунок 3.4 – Моделювання події підключення до кімнати

6. Відключення від кімнати. Ця процедура відбувається, коли користувач виходить з кімнати. Подія відключення від кімнати дуже схожа на подію підключення до кімнати, яка описана перед цим. Різниця лише в тому, що протокол змінюється назад. Також на серверній частині відбувається відключення даного користувача з кімнати.
7. Отримання AES ключа, який буде використовуватися для розшифрування повідомлень, доки користувач не вийде з кімнати. Для цього потрібен POST запит, куди буде відправлений публічний ключ

RSA, у відповідь ми отримаємо зашифрований ключ AES. Алгоритм шифрування був описаний в одному з попередніх розділів.

8. Отримання історії повідомлення. Коли користувач заходить до кімнати, йому потрібно надати можливість побачити повідомлення, які були написані в цій кімнаті до цього моменту. Отже потрібно отримувати історію повідомлення після того, як зайшов до кімнати. Процес отримання історії повідомлень зображений на рисунку 3.5.

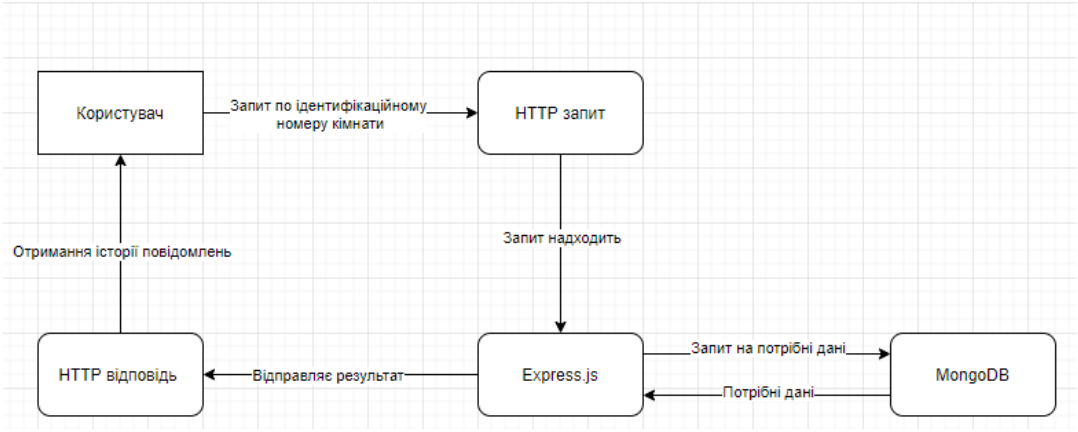

Рисунок 3.5 - Моделювання події отримання історії повідомлень

9. Надсилання повідомлення. Результат цієї події повинен надсилатися не лише автору повідомлення, а й усім користувачам, що підключені до кімнати, в якій це повідомлення було створене. Процес надсилання повідомлення зображено на рисунку 3.6.

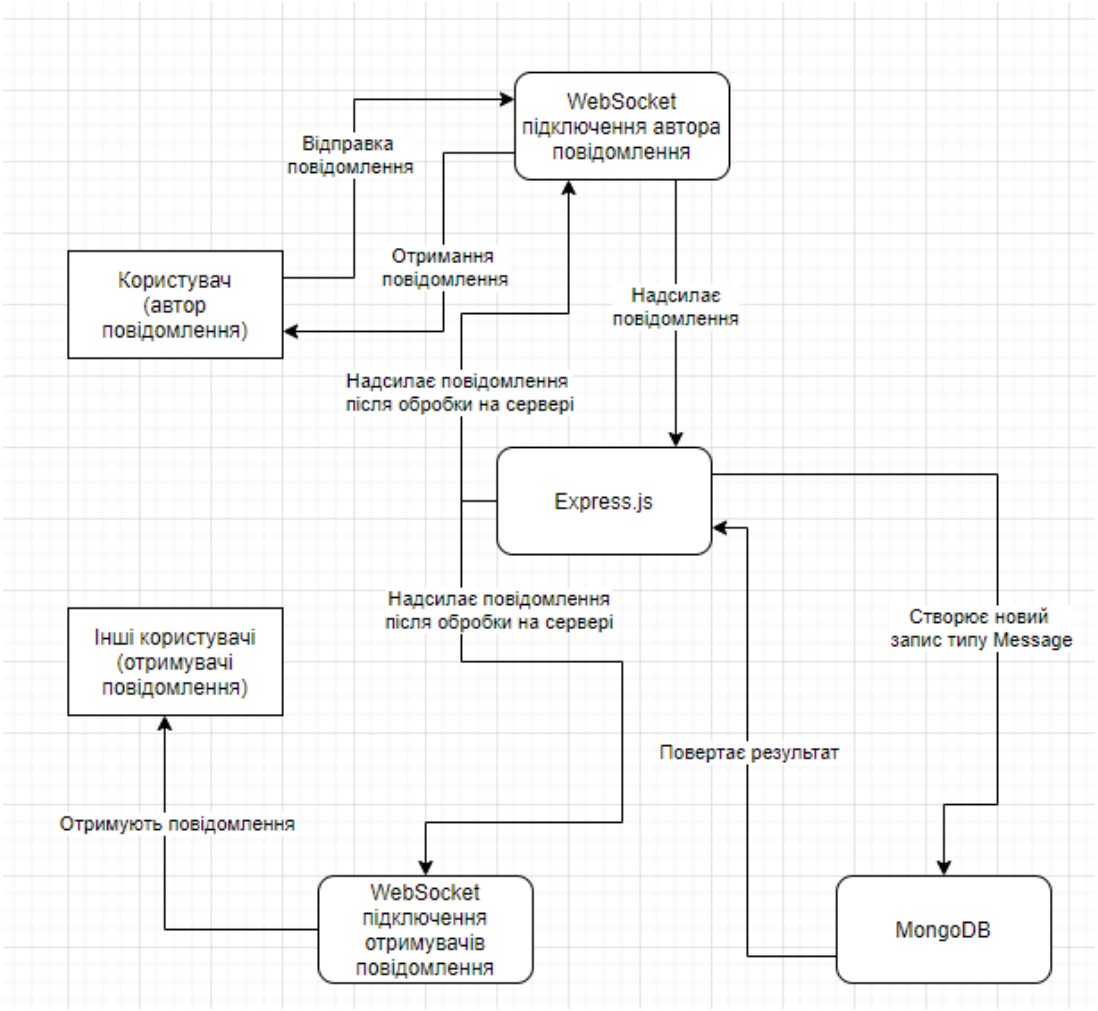

Рисунок 3.6 - Моделювання події відправки повідомлення

3.2.3 Моделювання інтерфейсу додатку

Якісний інтерфейс в веб-додатках є однією з запорук успіху, так як це проміжна ланка між користувачем і додатком. Потрібно спланувати, як буде виглядати користувацький інтерфейс.

Сплануємо усі сторінки та елементи, які будуть потрібні для користувача під час використання додатку.

1. Першою сторінкою додатку стане авторизація. На цій сторінці у користувача буде можливість ввести свій email та пароль, щоб відправити дані на обробку та отримати результат. Користувач не зможе відправляти повідомлення, доки не пройде цей етап. Приблизний дизайн сторінки авторизації можна побачити на рисунку 3.7.

Рисунок 3.7 - Дизайн сторінки авторизації

2. Для того щоб була можливість авторизуватися, спочатку потрібно зареєструвати свій акаунт, отже форма реєстрації буде виглядати приблизно так само, як і форма авторизації. Але буде додано ще одне поле username. Також слід додати, що авторизовані користувачі не можуть потрапити до сторінки реєстрації та авторизації, доки не виконають процедуру виходу із системи.
3. Перша сторінка, яку бачить користувач після авторизації це головна сторінка. На цій сторінці у користувача є можливість створити кімнату або зайти у вже існуючу ввівши id кімнати. Сама сторінка буде виглядати, як на рисунку 3.8. При натисканні на кнопку створити кімнату буде відкриватися модальне вікно, як на рисунку 3.9.

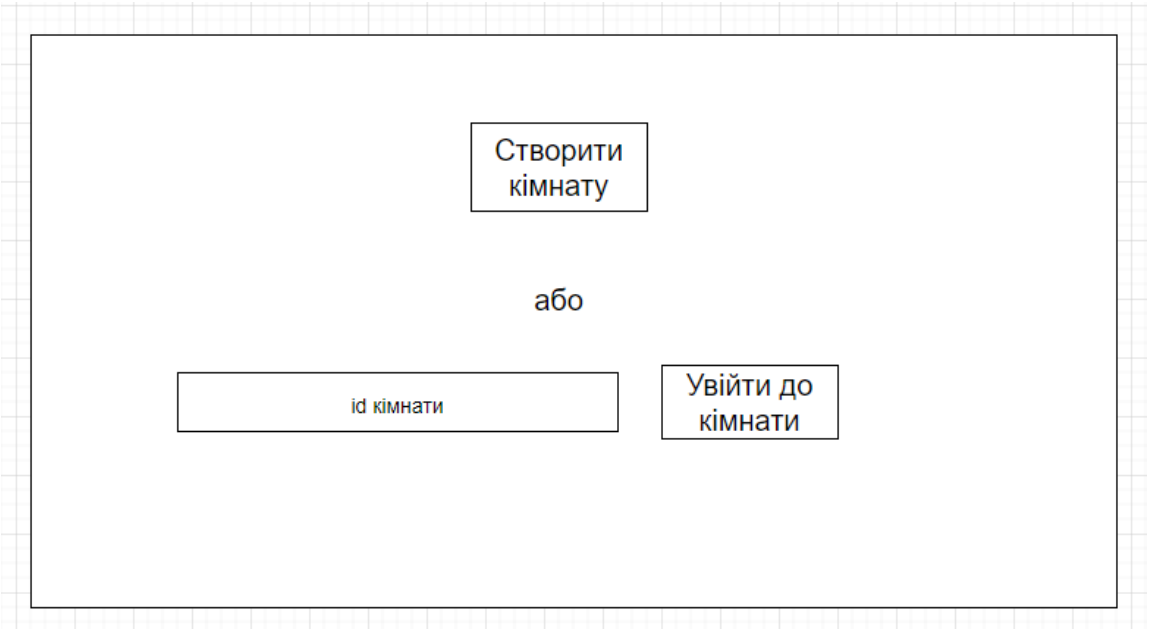

Рисунок 3.8 - Дизайн головної сторінки

Рисунок 3.9 - Дизайн модального вікна для створення кімнати

4. І найголовнішою сторінкою стане саме чат. На цій сторінці користувач зможе переглядати історію усіх повідомлень, які були написані у цій кімнаті, надсилати та отримувати повідомлення у реальному часі. Дизайн сторінки можна побачити на рисунку 3.10.

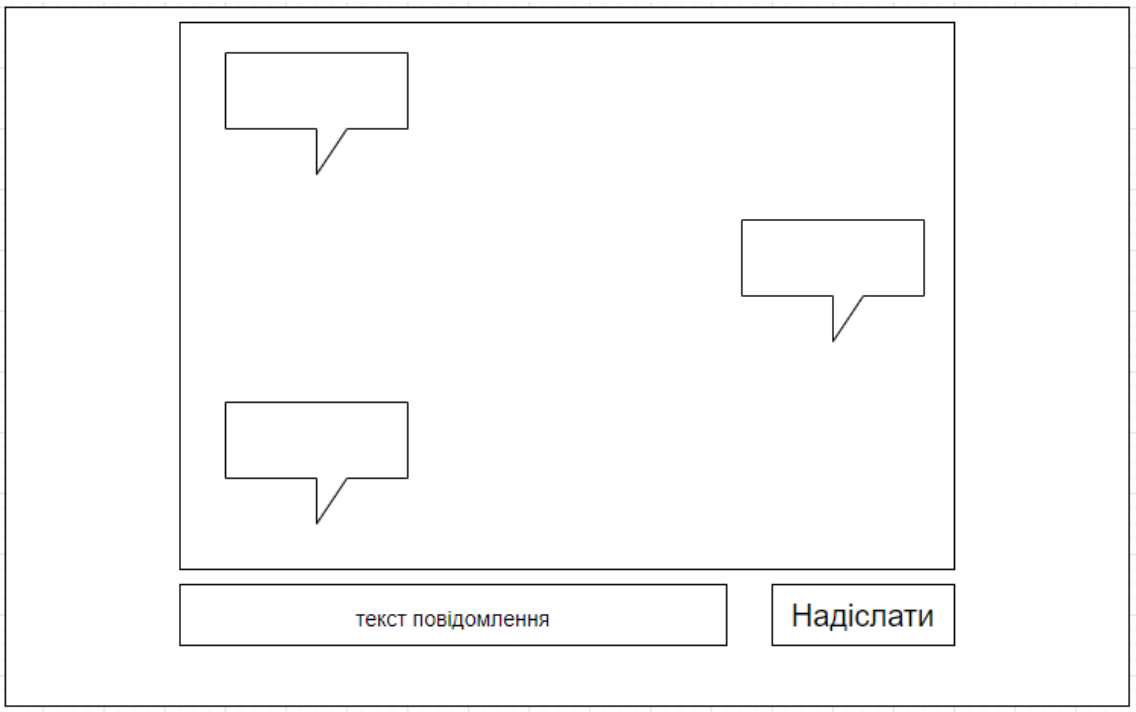

Рисунок 3.10 - Дизайн кімнати чату

3.3 Практична реалізація

Розробка практичної реалізації буде проходити двома етапами:

- Розробка серверної частини
- Розробка клієнтської частини

3.3.1 Стурктура серверної частини

Перш за все при розробці будь-якого додатку спочатку потрібно налаштувати середовище розробки.

Для встановлення окремих пакетів використовувався стандартний вбудований в Node.js пакетний менеджер npm.

Для розробки серверної частини я використовував такі сторонні бібліотеки:

- body-parser для обробки вхідних запитів та трансформації тіла запиту у JavaScript object.

- express сам фреймворк за допомогою якого розроблялася серверна частина додатку. Детально Express я описав в одному з попередніх розділів.
- express-ws для полегшення роботи з WebSocket API за допомогою express.
- helmet допомагає захистити веб-додатки Express, додаючи різні заголовки HTTP до усіх запитів.
- jsonwebtoken я використовував для реалізації концепції JSON Web Token(JWT). Це стандарт, який використовується для передачі даних аутентифікації у клієнт-серверних додатках.
- moment я використовував для приведення дати до одного формату, що на клієнті, що на сервері.
- mongoose це ORM система для полегшення запитів та загалом роботи з MongoDB.
- aes-js я використовував для створення ключів AES на серверній частині.
- node-rsa я використовував для шифрування AES ключа публічним ключом RSA на серверній частині.

Також для пришвидшення розробки я використовував бібліотеку nodemon, яка слідкує за будь-якими змінами коду в серверній частині додатку та власноруч перезапускає платформу без моєї необхідності робити це самому.

Набір конфігурації серверної частини включає в себе стандартні файли для розробки під платформу Node.js, також присутніми є файли для налаштування локального репозиторію, форматерів коду та власне самого редактору коду. Набір конфігурацій можна побачити на рисунку 3.11.

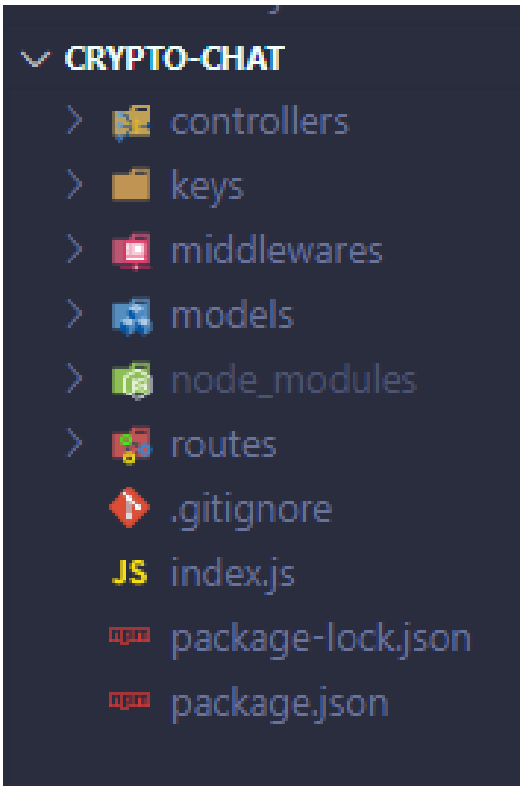

Рисунок 3.11 - Файлова структура серверної частини додатку

Реалізація створення AES ключів та їх шифрування на серверній частині виглядає так:

```
exports.sendKeys = (req, res) => {
    try {
        const id = req.params.id;
        const pubKey = new NodeRSA(req.body.key);
        const generateAES = (id) => seed(id)(32);
        const AESkey = generateAES(id);
        const encrypted = pubKey.encrypt(AESkey, "base64");
        res.send({ aesKey: encrypted });
    } catch (e) {
        console.log(e);
    }
};
    app.post("/api/room/:id/keys", [authJwt.verifyToken], controller.sendKeys);
```

3.3.2 Структура клієнтської частини

При розробці клієнтської частини додатку також спочатку потрібно налаштувати середовище розробки. Для ReactJS налаштувати стартову

конфігурацію досить легко, адже є спеціальна утиліта, яку розробив Facebook, Create React App. Ця утиліта швидко створює початкову версію SPA та усі потрібні налаштування.

React слідує компонентному підходу розробки додатків. Самі компоненти в моєму чаті діляться на декілька типів:

- Так звані структурні компоненти, які відповідають за структуру окремих сторінок.
- Нижче в ієрархії компонентів йдуть компоненти для логіки та відображення елементів користувацького інтерфейсу.
- Ще нижче в ієрархії стоять допоміжні компоненти, які можуть використовуватися в декількох компонентах. Прикладом такого компоненту може бути кнопка.

Не рахуючи основну бібліотеку для розробки клієнтської частини – React, в моєму додатку є дуже багато допоміжних пакетів, які були встановлені за допомогою пакетного менеджера npm. Розглянемо ключові з них:

- Найважливішою бібліотекою є redux. Ця бібліотека потрібна для менеджменту даних у всій клієнтській частині додатку. Ця бібліотека реалізує концепт подій. Весь додаток обгорнутий у обробник подій. Це дозволяє викликати певні події з будь-якого місця додатку та записувати дані в глобальний об’єкт Store. Ця бібліотека є надзвичайно корисною, так як React в своєму ядрі реалізує лише односторонню передачу даних. Компоненти можуть отримувати дані лише з компонентів, які є для них батьківськими. Redux вирішує цю проблему, так як до Store можна звертатися з будь-якого місця додатку.
- node-rsa використовується для створення ключів RSA на клієнті та розшифровування ключа AES.
- aes-js використовується для шифрування та розшифрування повідомлень ключом AES.

- reconnecting-websocket це бібліотека, яка допомагає повторно приєднатися до WebSocket в разі втрати зв'язку.

Набір конфігурації можна побачити на рисунку 3.12.

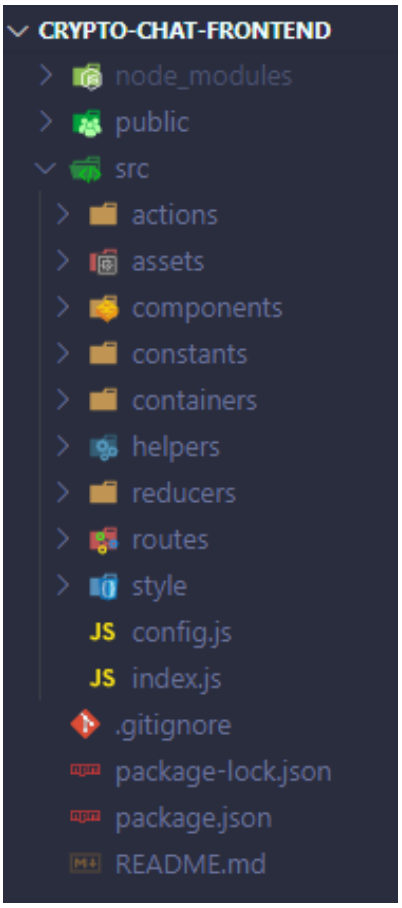

Рисунок 3.12 - Файлова структура клієнтської частини додатку

Реалізація процесу підключення до кімнати, під час якого відбувається отримання ключів, отримання історії повідомлення виглядає так:

```
componentDidMount() {
    const {
      getChatHistory, getInvites, match: {params: { id }, },getNewMessage, history, postKey,
      saveDecrypted,
    } = this.props;
    const { date } = this.state;
    getInvites(id).then((res) => {
      if (res.payload && res.payload.status && res.payload.status === 200){
        if (res && res.payload &&res.payload.data &&res.payload.data.isValid ) {
          const key = new NodeRSA({ b: 1024 });
          const pubKeyStr = key.exportKey("pkcs8-public");
          postKey(id, { key: pubKeyStr }).then((res) => {
```

```
                if (res.payload && res.payload.status &&res.payload.status === 200) {
                    if (res.payload && res.payload.data &&res.payload.data.aesKey) {
                        const decryptedStr = key.decrypt(res.payload.data.aesKey, "hex");
                        const decrypted = Buffer.from(decryptedStr, "hex");
                        this.setState({decryptKey: decrypted, });
                        saveDecrypted(decryptedStr);
                        this.rws.addEventListener("open", () => {console.log("Connected"); });
                        this.rws.addEventListener("message",
                            function (event) {
                                const obj = JSON.parse(event.data);
                                getNewMessage({...obj, text: decrypt(obj.text, decrypted), });
                            }
                        );
                        getChatHistory(id, moment(date).format("yyyy-MM-DDTHH:mm:ss"), 1
                        ).then((res) => {
                            if (res.payload && res.payload.status &&res.payload.status === 200) {
                                this.setState({ loading: false });
                            } else {
                                toast("Something went wrong. Try again, please!",
                                    {progressClassName: "red-progress",}
                                );
                            }
                        });
                    }
                } else {
                    history.push("/main");
                    toast("Something went wrong. Try again, please!", {
                        progressClassName: "red-progress",
                    });
                }
            });
        } else {
            history.push("/main");
            toast("У вас нету доступа к этой комнате!", {progressClassName: "red-progress", });
        }
    } else {
        history.push("/main");
        toast("ID комнаты неправильное!", {progressClassName: "red-progress",});
    }
});
}
```

3.4 Результати розробки додатку

Результатом розробки став додаток, який реалізує весь функціонал, який планувався на стадії моделювання.

Сторінки, які увійшли до фінальної версії додатку:

- Сторінка авторизації зображена на рисунку 3.13.

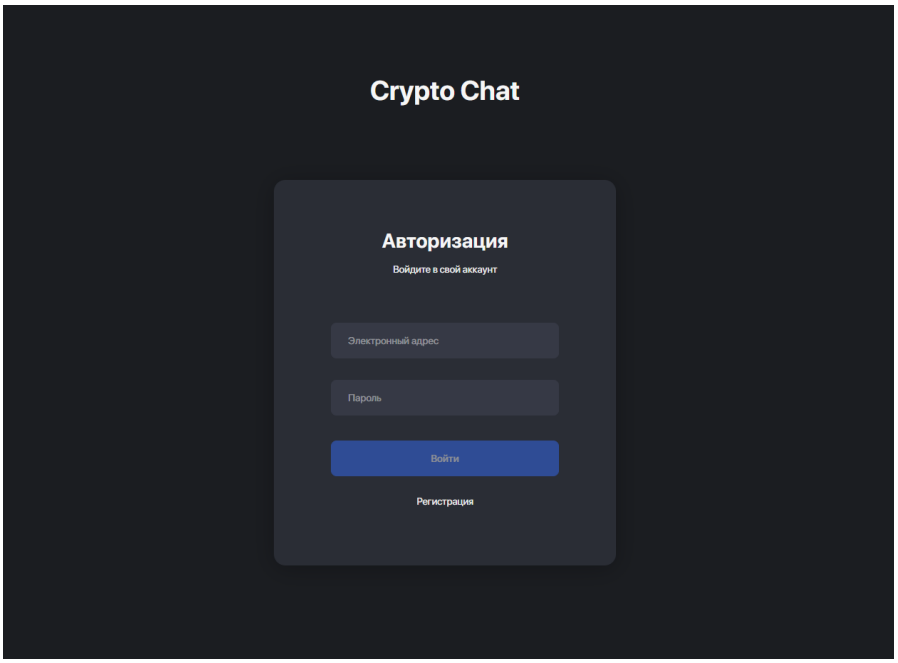

Рисунок 3.13 - Сторінка авторизації

- Сторінка реєстрації зображена на рисунку 3.14.

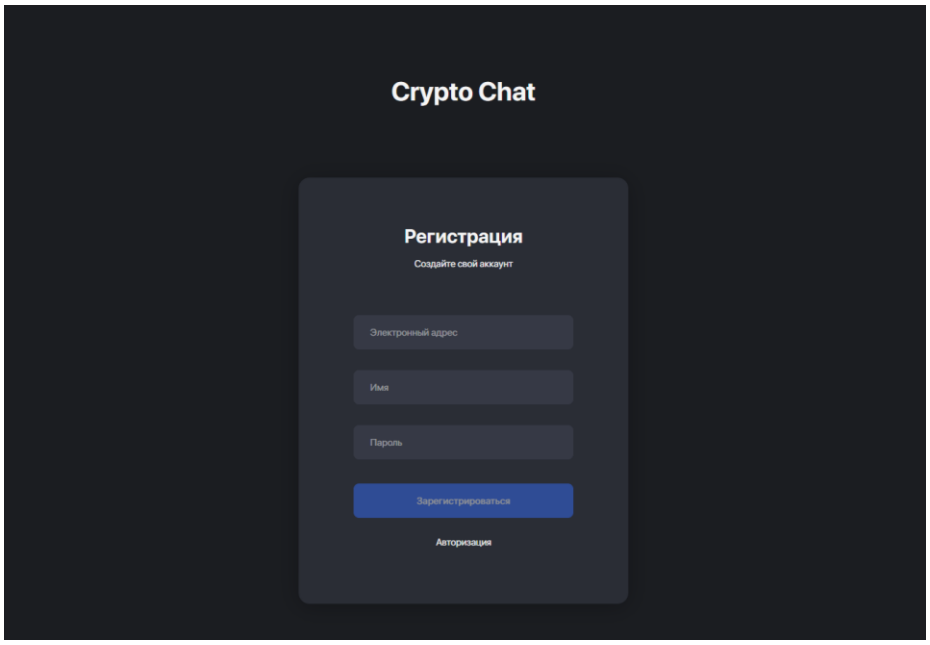

Рисунок 3.14 - Сторінка реєстрації

- Головна сторінка зображена на рисунку 3.15.

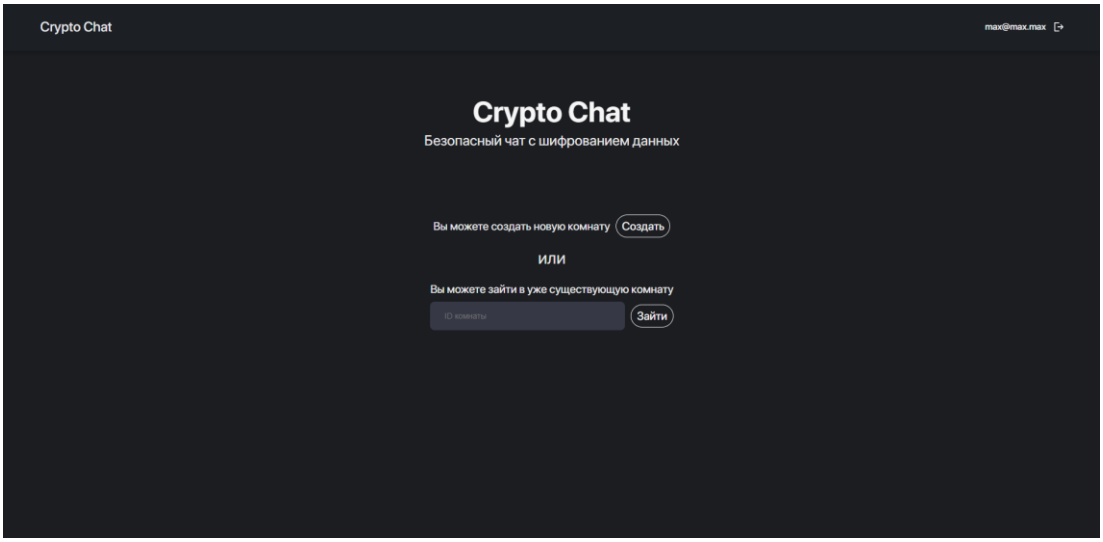

Рисунок 3.15 - Головна сторінка

- Модальне вікно на головній сторінці для створення кімнати зображене на рисунку 3.16.

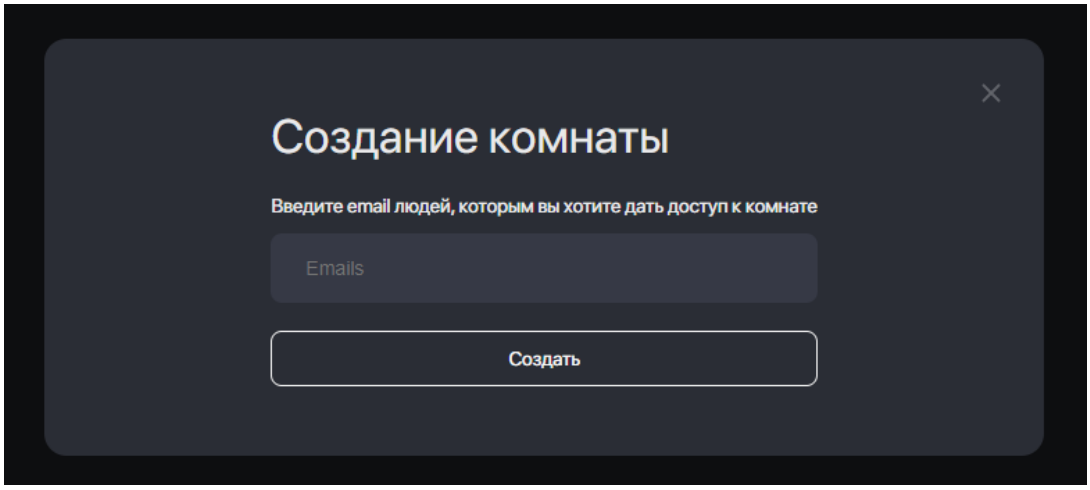

Рисунок 3.16 - Модальне вікно створення кімнати

- Сторінка чату зображена на рисунку 3.17.

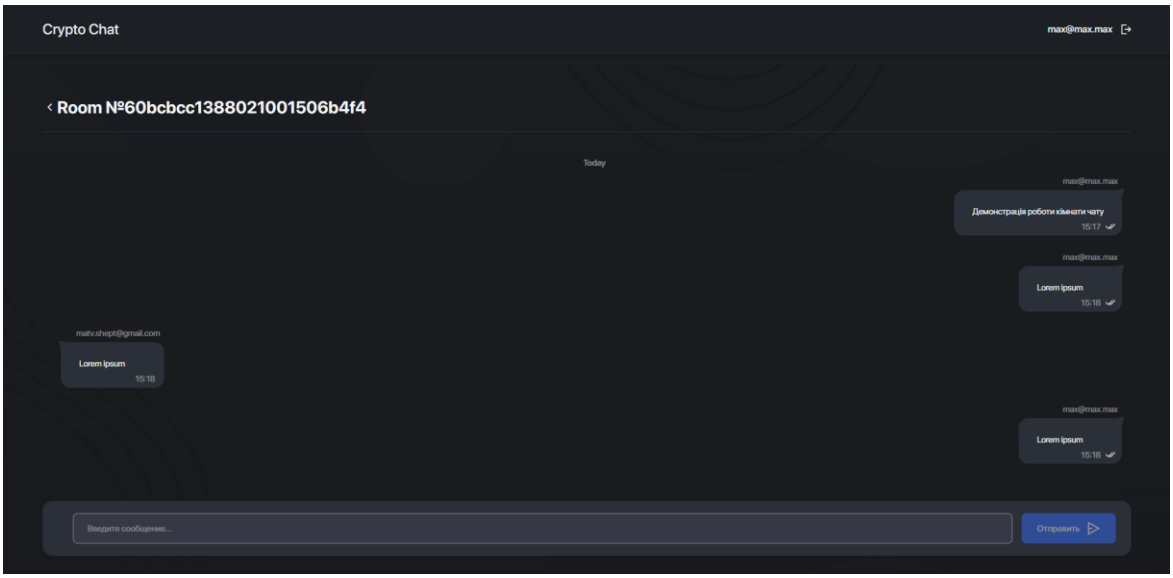

Рисунок 3.17 - Сторінка чату

Отже, ми отримали веб-додаток для обміну повідомленнями у реальному часі з шифруванням повідомлень та хешуванням паролів. Для розробки були використані: Node.js для серверної частини, ReactJS для клієнтської частини, MongoDB для баз даних, WebSocket API для з’єднання клієнтської та серверної частини. Були реалізовані усі моделі, які планувалися та усі події, які були описані. Також був розроблений користувацький інтерфейс за дизайном.

ВИСНОВКИ

Отже, в ході даної роботи були розглянуті вразливості веб-додатків та способи їх захисту. Детально було розібране шифрування даних у веб-додатках та хешування.

У ході огляду хешування був зроблений порівняльний огляд таких алгоритмів хешування, як SHA-1 та SHA-256. Звідки стало зрозуміло, що SHA-256 у наш час перемагає по всім параметрам.

Під час огляду шифрування даних були розглянуті різні види шифрування, такі як симетричне та асиметричне. Було зроблено порівняльний аналіз представників симетричного типу шифрування. Цими представниками були AES та DES. Порівнявши ці два алгоритми шифрування стало зрозуміло, що AES є новішим, значно надійнішим та кращим по всім параметрам, ніж DES. Далі були розглянуті представники асиметричного шифрування: RSA та ElGamal. В цьому порівнянні було не все так очевидно, так як RSA має значно вищу швидкість шифрування даних, а ElGamal може запропонувати кращу швидкість розшифрування даних. Ключовим фактором стала наявність детального стандарту для RSA. В цьому стандарті досконало розписаний цей алгоритм.

Для того, щоб підбити підсумки теоретичної частини, був спочатку змодельований, а потім і розроблений веб-додаток для обміну повідомленнями у реальному часі, в якому було наглядно реалізоване хешування паролів за допомогою SHA-256 та шифрування повідомлень за допомогою мого алгоритму, в якому я використав AES та RSA.

СПИСОК ЛІТЕРАТУРИ

1. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. – NY: Wiley, 2016. – С.114-120
2. Easttom W. Modern Cryptography: Applied Mathematics for Encryption and Information Security. – BER: Springer, 2020. – С.97-104.
3. Dooley J. F. History of Cryptography and Cryptanalysis: Codes, Ciphers, and Their Algorithms. – BER: Springer, 2018. - С.17-18.
4. Mihailescu M. I. Pro Cryptography and Cryptanalysis: Creating Advanced Algorithms with C# and .NET. – NY: Apress, 2020. – С.407-408.
5. Daemen J. The Design of Rijndael: The Advanced Encryption Standard (AES). – BER: Springer, 2020. – С.147-151.
6. Song Y. Y. Cryptanalytic Attacks on RSA – BER: Springer US, 2008. – С.19-26.
7. Takagi T. International Symposium on Mathematics, Quantum Theory, and Cryptography: Proceedings of MQC 2019 (Mathematics for Industry, 33) – BER: Springer, 2020. – С.102-108.
8. Aumasson P. Serious Cryptography: A Practical Introduction to Modern Encryption – NY: No Starch Press, 2017. – С.71-79.
9. Stallings W. Cryptography and Network Security: Principles and Practice (6th Edition) – NY: Pearson, 2013. – С.189-199.
10. Bernstein D. J. Post-Quantum Cryptography – BER: Springer, 2009. – С.204-206.
11. Janeczko P. B. Top Secret: A Handbook of Codes, Ciphers and Secret Writing. - NY: Candlewick, 2006. – С.19-22
12. Баричев С. Криптография без секретов – МСК: Горячая линия – Телеком, 2004. – С.25-31.
13. В чем преимущества Node.js? [Електронний ресурс] – Режим доступу до ресурсу: https://artjoker.ua/ru/blog/v-chem-preimushchestva-nodejs/

14. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Node.js
15. ExpressJS [Електронний ресурс] – Режим доступу до ресурсу: https://umbrellait.com/ru/blog/choosing-the-best-nodejs-framework/
16. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://mcs.mail.ru/blog/osobennosti-mongodb-kogda-baza-dannyh-vam-podhodit
17. ReactJS [Електронний ресурс] – Режим доступу до ресурсу: https://ru.reactjs.org/
18. WebSocket [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/WebSocket

ДОДАТОК

А.1 Серверна частина

index.js:

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const cors = require("cors");
const db = require("./models");
const app = express();
const keys = require("./keys");
const expressWs = require("express-ws")(app);
const helmet = require("helmet");

// const corsOptions = {
//     origin: "http://localhost:3000",
// };

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(helmet());

db.mongoose
    .connect(keys.MONGODB_URI, {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    })
    .then(() => {
        console.log("Successfully connect to MongoDB.");
    })
    .catch((err) => {
        console.error("Connection error", err);
        process.exit();
    });

app.get("/", (req, res) => {
    res.json({ message: "Welcome to crypto-chat backend." });
});

require("./routes/auth")(app);
require("./routes/user")(app);
require("./routes/chat")(app, expressWs);

const PORT = process.env.PORT || 8080;

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}.`);
});
```

auth.controller.js:

```
const keys = require("../keys");
const db = require("../models");
const User = db.user;

const jwt = require("jsonwebtoken");
const { createHash } = require("crypto");
```

```
exports.signup = (req, res) => {
    const user = new User({
        username: req.body.username,
        email: req.body.email,
        password: createHash("sha256", keys.SALT_SECRET)
            .update(req.body.password)
            .digest("hex"),
    });

    user.save((err, user) => {
        if (err) {
            res.status(500).send({ message: err });
            return;
        }
        user.save((err) => {
            if (err) {
                res.status(500).send({ message: err });
                return;
            }
            res.status(200).send({ message: "User was registered!" });
        });
    });
};

exports.signin = (req, res) => {
    User.findOne({
        email: req.body.email,
    })
        .populate("-__v")
        .exec((err, user) => {
            if (err) {
                res.status(500).send({ message: err });
                return;
            }
            if (!user) {
                return res.status(404).send({ message: "User wasn't found." });
            }
            const passwordIsValid =
                createHash("sha256", keys.SALT_SECRET)
                    .update(req.body.password)
                    .digest("hex") === user.password;

            if (!passwordIsValid) {
                return res.status(401).send({
                    accessToken: null,
                    message: "Wrong password!",
                });
            }
            const token = jwt.sign({ id: user.id }, keys.JWT_SECRET);

            res.status(200).send({
                id: user._id,
                username: user.username,
                email: user.email,
                accessToken: token,
            });
        });
};
```

user.controller.js:

```
const db = require("../models");
const User = db.user;
const Room = db.room;
const Message = db.message;
const jwt = require("jsonwebtoken");
const mongoose = require("mongoose");
const keys = require("../keys");
const moment = require("moment");
const NodeRSA = require("node-rsa");
const seed = require("random-bytes-seed");
const aesjs = require("aes-js");

exports.getHeader = (req, res) => {
    try {
        jwt.verify(
            req.headers["x-access-token"],
            keys.JWT_SECRET,
            (err, decoded) => {
                if (err) {
                    return res.status(401).send({ message: "Unauthorized!" });
                }
                User.findById(mongoose.Types.ObjectId(decoded.id.toString()))
                    .populate("-__v")
                    .exec((err, user) => {
                        res.status(200).send({ email: user.email });
                    });
            }
        );
    } catch (e) {
        console.log(e);
    }
};

exports.createRoom = (req, res) => {
    try {
        jwt.verify(
            req.headers["x-access-token"],
            keys.JWT_SECRET,
            (err, decoded) => {
                if (err) {
                    return res.status(401).send({ message: "Unauthorized!" });
                }
                User.findById(mongoose.Types.ObjectId(decoded.id.toString()))
                    .populate("-__v")
                    .exec((err, user) => {
                        const invitesArr = [...req.body.invites, user.email];
                        const room = new Room({
                            invites: Array.from(new Set(invitesArr)),
                            messages: [],
                        });
                        room.save((err, room) => {
                            if (err) {
                                res.status(500).send({ message: err });
                                return;
                            }
                            res.send({ roomId: room.id });
                        });
                    });
            }
        );
    } catch (e) {
```

```
        console.log(e);
    }
};

exports.getHistory = (req, res) => {
    try {
        const date = req.query.create_date;
        const page = req.query.page;
        Room.findById(mongoose.Types.ObjectId(req.params.id))
            .populate("messages.messageId")
            .exec((err, room) => {
                if (err) {
                    res.status(500).send({ message: err });
                    return;
                }
                const messages = room.messages
                    .map((el) => el.messageId)
                    .filter((el) => new Date(el.create_date) < new Date(date))
                    .reverse();
                res.send({
                    results: messages.slice(+`${page - 1}0`, +`${page}0`),
                    pageAmount: Math.ceil(messages.length / 10),
                    messagesAmount: messages.length,
                    next: `/api/history/${
                        req.params.id
                    }?create_date=${date}&page=${+page + 1}`,
                });
            });
    } catch (e) {
        console.log(e);
    }
};

exports.getRoom = (req, res) => {
    try {
        jwt.verify(
            req.headers["x-access-token"],
            keys.JWT_SECRET,
            (err, decoded) => {
                if (err) {
                    return res.status(401).send({ message: "Unauthorized!" });
                }
                User.findById(mongoose.Types.ObjectId(decoded.id.toString()))
                    .populate("-__v")
                    .exec((err, user) => {
                        Room.findById(req.params.id)
                            .populate("-__v")
                            .exec((err, room) => {
                                if (err) {
                                    res.status(500).send({ message: err });
                                    return;
                                }

                                res.send({
                                    isValid: room.invites.some(
                                        (el) => el === user.email
                                    ),
                                });
                            });
                    });
            }
```

```
        );
    } catch (e) {
        console.log(e);
    }
};

exports.sendKeys = (req, res) => {
    try {
        const id = req.params.id;
        const pubKey = new NodeRSA(req.body.key);
        const generateAES = (id) => seed(id)(32);
        const AESkey = generateAES(id);
        const encrypted = pubKey.encrypt(AESkey, "base64");
        res.send({ aesKey: encrypted });
    } catch (e) {
        console.log(e);
    }
};
```

keys.dev.js:

```
module.exports = {
MONGODB_URI:"mongodb+srv://Max:tzqY77nndFA8Boks@cluster0.r0mjz.mongodb.net/cryptoChat",
    JWT_SECRET: "crypto-chat-secret-key",
    SALT_SECRET: "crypto-salt-test-hidden",
};
```

keys.prod.js:

```
module.exports = {
    MONGODB_URI: process.env.MONGODB_URI,
    JWT_SECRET: process.env.JWT_SECRET,
    SALT_SECRET: process.env.SALT_SECRET,
};
```

authJwt.js:

```
const jwt = require("jsonwebtoken");
const db = require("../models");
const User = db.user;
const keys = require("../keys");
const mongoose = require("mongoose");

verifyToken = (req, res, next) => {
    let token = req.headers["x-access-token"];

    if (!token) {
        return res.status(403).send({ message: "No token!" });
    }

    jwt.verify(token, keys.JWT_SECRET, (err, decoded) => {
        if (err) {
            return res.status(401).send({ message: "You are unauthorized!" });
        }
        User.findById(mongoose.Types.ObjectId(decoded.id.toString()))
            .populate("-__v")
            .exec((err, user) => {
                if (err || !user) {
                    return res.status(401).send({ message: "You are unauthorized!" });
                }
                req.userId = decoded.id;
                next();
            });
    });
```

```
};

const authJwt = {
    verifyToken,
};
module.exports = authJwt;

verifySignUp.js:
const db = require("../models");
const User = db.user;

checkDuplicateUsernameOrEmail = (req, res, next) => {
    User.findOne({
        username: req.body.username,
    }).exec((err, user) => {
        if (err) {
            res.status(500).send({ message: err });
            return;
        }

        if (user) {
            res.status(400).send({
                message: "Failed! Username is already in use!",
            });
            return;
        }

        // Email
        User.findOne({
            email: req.body.email,
        }).exec((err, user) => {
            if (err) {
                res.status(500).send({ message: err });
                return;
            }

            if (user) {
                res.status(400).send({
                    message: "Failed! Email is already in use!",
                });
                return;
            }

            next();
        });
    });
};

const verifySignUp = {
    checkDuplicateUsernameOrEmail,
};

module.exports = verifySignUp;

message.js:
const { model, Schema } = require("mongoose");

const Message = model(
    "Message",
    new Schema({
        text: {
```

```
            type: String,
            required: true,
        },
        sender: {
            type: String,
            required: true,
        },
        create_date: {
            type: Date,
            required: true,
        },
    })
);

module.exports = Message;
```

room.js:

```
const { model, Schema } = require("mongoose");
const mongoose = require("mongoose");

const roomSchema = new Schema({
    invites: [
        {
            type: String,
        },
    ],
    messages: [{ messageId: { type: Schema.Types.ObjectId, ref: "Message" } }],
});

roomSchema.methods.addMessage = function (message, next) {
    const messages = [...this.messages];

    messages.push({ messageId: mongoose.Types.ObjectId(message._id) });
    this.messages = messages;

    this.save();
    return next();
};

module.exports = model("Room", roomSchema);
```

user.js:

```
const { model, Schema } = require("mongoose");

const User = model(
    "User",
    new Schema({
        username: {
            type: String,
            required: true,
        },
        email: {
            type: String,
            required: true,
        },
        password: {
            type: String,
            required: true,
        },
    })
);
```

```
module.exports = User;

auth.js:
const { verifySignUp } = require("../middlewares");
const controller = require("../controllers/auth.controller");

module.exports = function (app) {
    app.use(function (req, res, next) {
        res.header(
            "Access-Control-Allow-Headers",
            "x-access-token, Origin, Content-Type, Accept"
        );
        next();
    });

    app.post(
        "/api/auth/signup",
        [verifySignUp.checkDuplicateUsernameOrEmail],
        controller.signup
    );

    app.post("/api/auth/signin", controller.signin);
};

chat.js:
const db = require("../models");
const Room = db.room;
const mongoose = require("mongoose");
const jwt = require("jsonwebtoken");
const keys = require("../keys");
const User = db.user;
const moment = require("moment");
const Message = db.message;

module.exports = function (app, expressWs) {
    app.use(function (req, res, next) {
        res.header(
            "Access-Control-Allow-Headers",
            "x-access-token, Origin, Content-Type, Accept"
        );
        next();
    });

    const activeConnections = {};

    app.ws("/:id", async function (ws, req) {
        const token = req.query.token;
        const id = req.params.id;
        let room;
        try {
            room = await Room.findById(id);
            if (!room) {
                throw new Error();
            }
        } catch (e) {
            console.log("room err");
            ws.terminate();
            return;
        }
```

```
        if (!token) {
            console.log("no token");
            ws.terminate();
            return;
        }
        let gUser;
        jwt.verify(token, keys.JWT_SECRET, (err, decoded) => {
            if (err) {
                console.log("wrong token");
                ws.terminate();
                return;
            }
            User.findById(mongoose.Types.ObjectId(decoded.id.toString()))
                .populate("-__v")
                .exec((err, user) => {
                    if (err || !user) {
                        console.log("no user");
                        ws.terminate();
                        return;
                    } else {
                        gUser = user;
                    }
                });
        });

        if (activeConnections[id]) {
            activeConnections[id].push(ws);
        } else {
            activeConnections[id] = [ws];
        }

        ws.on("message", function (msg) {
            const message = new Message({
                text: msg,
                sender: gUser.email,
                create_date: moment(new Date()).format("yyyy-MM-DDTHH:mm:ss"),
            });
            message.save((err, mes) => {
                if (err) {
                    ws.terminate();
                    return;
                }
                room.addMessage(mes, (error) => {
                    if (error) {
                        ws.terminate();
                        return;
                    }
                    activeConnections[id].forEach(function (client) {
                        client.send(JSON.stringify(message));
                    });
                });
            });
        });
        ws.on("close", function () {
            activeConnections[id] = activeConnections[id].filter(
                (el) => el !== ws
            );
        });
    });
};
```

user.js:
```
const { authJwt } = require("../middlewares");
const controller = require("../controllers/user.controller");

module.exports = function (app) {
    app.use(function (req, res, next) {
        res.header(
            "Access-Control-Allow-Headers",
            "x-access-token, Origin, Content-Type, Accept"
        );
        next();
    });

    app.get("/api/header", [authJwt.verifyToken], controller.getHeader);

    app.post("/api/room", [authJwt.verifyToken], controller.createRoom);

    app.get("/api/history/:id", [authJwt.verifyToken], controller.getHistory);

    app.get("/api/room/:id", [authJwt.verifyToken], controller.getRoom);

    app.post("/api/room/:id/keys", [authJwt.verifyToken], controller.sendKeys);
};
```

А2 Клієнтська частина

index.js:
```
import React from "react";
import ReactDOM from "react-dom";
import { Provider } from "react-redux";
import { createStore, applyMiddleware } from "redux";
import { createBrowserHistory } from "history";
import {
    routerMiddleware,
    ConnectedRouter,
} from "connected-react-router/immutable";
import { multiClientMiddleware } from "redux-axios-middleware";

import api from "./actions/api";
import routes from "./routes/routes";
import rootReducer from "./reducers";

const axiosMiddlewareOptions = {
    interceptors: {
        request: [
            (action, config) => {
                if (localStorage.token || localStorage.token_res) {
                    let token = localStorage.token
                        ? localStorage.token
                        : localStorage.token_res;
                    config.headers["x-access-token"] = token;
                }
                return config;
            },
        ],
        response: [
            {
                success: ({ dispatch }, response) => {
                    return response;
                },
                error: ({ dispatch }, error) => {
```

```
                if (error.response.status === 401) {
                    localStorage.clear();
                }
                return Promise.reject(error);
            },
        },
    ],
  },
};

const history = createBrowserHistory();
const appRouterMiddleware = routerMiddleware(history);
const createStoreWithMiddleware = applyMiddleware(
  multiClientMiddleware(api, axiosMiddlewareOptions),
  appRouterMiddleware
)(createStore);
const store = createStoreWithMiddleware(
  rootReducer(history),
  {},
  window.__REDUX_DEVTOOLS_EXTENSION__
    ? window.__REDUX_DEVTOOLS_EXTENSION__()
    : (f) => f
);

ReactDOM.render(
  <Provider store={store}>
    <ConnectedRouter history={history} children={routes} />
  </Provider>,
  document.getElementById("wrapper")
);
```

config.js:

```
let BASE_URL, SECOND_URL;

const host = window.location.host;

if (host === "localhost:3000") {
  BASE_URL = "http://localhost:8080/api/";
} else if (host.includes("localhost")) {
  BASE_URL = "http://localhost:8080/api/";
} else {
  BASE_URL = `https://crypto-chat-back.herokuapp.com/api/`;
}

SECOND_URL = "";

export const API_BASE_URL = BASE_URL;
export const API_SECOND_URL = SECOND_URL;
```

api.jsx:

```
import axios from "axios";
import * as apiUrl from "../config";

export const api = {
  default: {
    client: axios.create({
      baseURL: apiUrl.API_BASE_URL,
      responseType: "json"
    })
  },
  second: {
```

```
        client: axios.create({
            baseURL: apiUrl.API_SECOND_URL,
            responseType: "json"
        })
    }
};

export default api;
```

authActions.jsx:

```
import { AUTH } from "../constants";

export function postLogin(data) {
    return {
        type: AUTH.LOGIN,
        payload: {
            client: "default",
            request: {
                url: `auth/signin`,
                method: "post",
                data,
            },
        },
    };
}

export function register(data) {
    return {
        type: AUTH.REGISTER,
        payload: {
            client: "default",
            request: {
                url: `auth/signup`,
                method: "post",
                data,
            },
        },
    };
}

export function getHeaderInfo() {
    return {
        type: AUTH.GET_HEADER_INFO,
        payload: {
            client: "default",
            request: {
                url: `header`,
                method: "get",
            },
        },
    };
}

export function logout() {
    return {
        type: AUTH.LOGOUT,
        payload: {
            client: "default",
            request: {
                url: `auth/logout/`,
                method: "post",
```

```
            },
        },
    };
}
```

clientsActions.js:

```
import { CLIENTS } from "../constants";

export function getChatHistory(id, date, page) {
    return {
        type: CLIENTS.GET_CHAT_HISTORY,
        payload: {
            client: "default",
            request: {
                url: `/history/${id}/?create_date=${date}&page=${page}`,
                method: "get",
            },
        },
    };
}

export function getMoreChatMessages(url) {
    return {
        type: CLIENTS.GET_MORE_CHAT_MESSAGES,
        payload: {
            client: "default",
            request: {
                url: `/${url}`,
                method: "get",
            },
        },
    };
}

export function getNewMessage(message) {
    return {
        type: CLIENTS.GET_NEW_MESSAGE,
        payload: { message },
    };
}

export function sendMessage(id, data) {
    return {
        type: CLIENTS.SEND_MESSAGE,
        payload: {
            client: "default",
            request: {
                url: `/message/${id}`,
                method: "post",
                data,
            },
        },
    };
}

export function postKey(id, data) {
    return {
        type: CLIENTS.POST_KEY,
        payload: {
            client: "default",
            request: {
```

```
            url: `/room/${id}/keys`,
            method: "post",
            data,
        },
    },
  };
}

export function createRoom(data) {
  return {
    type: CLIENTS.CREATE_ROOM,
    payload: {
      client: "default",
      request: {
        url: `/room`,
        method: "post",
        data,
      },
    },
  };
}

export function getInvites(id) {
  return {
    type: CLIENTS.GET_INVITES,
    payload: {
      client: "default",
      request: {
        url: `/room/${id}`,
        method: "get",
      },
    },
  };
}

export function saveDecrypted(str) {
  return {
    type: CLIENTS.SAVE_DECRYPTED_STRING,
    payload: {
      str,
    },
  };
}

constants/index.jsx:
export const AUTH = {
  LOGIN: "LOGIN",
  LOGIN_SUCCESS: "LOGIN_SUCCESS",
  LOGIN_FAIL: "LOGIN_FAIL",
  REGISTER: "REGISTER",

  LOGOUT: "LOGOUT",

  GET_HEADER_INFO: "GET_HEADER_INFO",
  GET_HEADER_INFO_SUCCESS: "GET_HEADER_INFO_SUCCESS",
  GET_HEADER_INFO_FAIL: "GET_HEADER_INFO_FAIL",
};

export const APP = {
  LOADING: "LOADING",
  BUTTON_LOADING: "BUTTON_LOADING",
```

```
    ERROR_SNACK_OPEN: "ERROR_SNACK_OPEN",
    ERROR_SNACK_CLOSE: "ERROR_SNACK_CLOSE",

    SUCCESS_SNACK_OPEN: "SUCCESS_SNACK_OPEN",
    SUCCESS_SNACK_CLOSE: "SUCCESS_SNACK_CLOSE",
};

export const CLIENTS = {
    GET_CHAT_HISTORY: "GET_CHAT_HISTORY",
    GET_CHAT_HISTORY_SUCCESS: "GET_CHAT_HISTORY_SUCCESS",

    GET_MORE_CHAT_MESSAGES: "GET_MORE_CHAT_MESSAGES",
    GET_MORE_CHAT_MESSAGES_SUCCESS: "GET_MORE_CHAT_MESSAGES_SUCCESS",

    GET_NEW_MESSAGE: "GET_NEW_MESSAGE",

    SEND_MESSAGE: "SEND_MESSAGE",

    GET_ALL_CHATS: "GET_ALL_CHATS",
    GET_ALL_CHATS_SUCCESS: "GET_ALL_CHATS_SUCCESS",

    CREATE_ROOM: "CREATE_ROOM",

    GET_INVITES: "GET_INVITES",
    GET_INVITES_SUCCESS: "GET_INVITES_SUCCESS",

    POST_KEY: "POST_KEY",

    SAVE_DECRYPTED_STRING: "SAVE_DECRYPTED_STRING",
    SAVE_DECRYPTED_STRING_SUCCESS: "SAVE_DECRYPTED_STRING_SUCCESS",
};

reducers/index.js:
import { combineReducers } from "redux";
import { connectRouter } from "connected-react-router";
import AuthReducer from "./reduceAuth";
import AppReducer from "./reduceApp";
import ClientsReducer from "./reduceClients";
import { reducer as formReducer } from "redux-form";

const rootReducer = (history) =>
    combineReducers({
        router: connectRouter(history),
        form: formReducer,
        auth: AuthReducer,
        app: AppReducer,
        clients: ClientsReducer,
    });

export default rootReducer;

reduceApp.jsx:
import { APP } from "../constants";

const INITIAL_STATE = {
    loading: false,
    buttonLoading: false,
    errorSnack: false,
    errorSnackText: "",
    successSnack: "",
```

```
    successSnackText: ""
};

export default function(state = INITIAL_STATE, action) {
    switch (action.type) {
        case APP.LOADING:
            return { ...state, loading: action.payload };
        case APP.BUTTON_LOADING:
            return { ...state, buttonLoading: action.payload };
        case APP.ERROR_SNACK_OPEN:
            return { ...state, errorSnack: true, errorSnackText: action.payload };
        case APP.ERROR_SNACK_CLOSE:
            return { ...state, errorSnack: false };
        case APP.SUCCESS_SNACK_OPEN:
            return { ...state, successSnack: true, successSnackText: action.payload };
        case APP.SUCCESS_SNACK_CLOSE:
            return { ...state, successSnack: false };
        default:
            return state;
    }
}
```

reduceAuth.js:

```
import { AUTH } from "../constants";

const INITIAL_STATE = {
    error_auth: "",
    token: "",
    email: {},
};

export default function (state = INITIAL_STATE, action) {
    switch (action.type) {
        case AUTH.LOGIN_SUCCESS:
            localStorage.setItem("token", action.payload.data.accessToken);
            return {
                ...state,
                token: action.payload.data.token,
            };
        case AUTH.GET_HEADER_INFO_SUCCESS:
            return {
                ...state,
                email: action.payload.data,
            };
        case AUTH.GET_HEADER_INFO_FAIL:
            localStorage.removeItem("token");
            window.location.replace("/auth/login");
            return INITIAL_STATE;
        default:
            return state;
    }
}
```

reduceClients.js:

```
import { CLIENTS } from "../constants";
import { Buffer } from "buffer";
import { decrypt } from "../helpers/functions";

const INITIAL_STATE = {
    messages: {},
    loading: true,
```

```
    invites: [],
    decryptedString: "",
};

export default function (state = INITIAL_STATE, action) {
    switch (action.type) {
        case CLIENTS.SAVE_DECRYPTED_STRING:
            return {
                ...state,
                decryptedString: action.payload.str,
            };
        case CLIENTS.GET_CHAT_HISTORY_SUCCESS:
            const decrypted = Buffer.from(state.decryptedString, "hex");
            return {
                ...state,
                messages: {
                    ...action.payload.data,
                    results: action.payload.data.results.map((el) => ({
                        ...el,
                        text: decrypt(el.text, decrypted),
                    })),
                },
                loading: !state.loading,
            };
        case CLIENTS.GET_MORE_CHAT_MESSAGES_SUCCESS:
            const decryptedMore = Buffer.from(state.decryptedString, "hex");
            return {
                ...state,
                messages: {
                    ...state.messages,
                    ...action.payload.data,
                    results: [
                        ...state.messages.results,
                        ...[
                            ...action.payload.data.results.map((el) => ({
                                ...el,
                                text: decrypt(el.text, decryptedMore),
                            })),
                        ],
                    ],
                },
            };
        case CLIENTS.GET_NEW_MESSAGE:
            return {
                ...state,
                messages: {
                    ...state.messages,
                    ...action.payload,
                    results: [
                        {
                            ...action.payload.message,
                            admin: {
                                chat_username: action.payload.message.admin,
                            },
                        },
                        ...state.messages.results,
                    ],
                },
                loading: !state.loading,
            };
        case CLIENTS.GET_INVITES_SUCCESS:
```

```
            return {
                ...state,
                invites: action.payload.data,
            };
        default:
            return state;
    }
}

routes.js:
import React from "react";
import App from "../containers/App";
import { Route, Switch, Redirect } from "react-router-dom";
import Container from "../containers/Container/Container";
import AuthContainer from "../containers/AuthContainer/AuthContainer";
import NotFound from "../components/NotFound/NotFound";

export default (
    <App>
        <Switch>
            <Route
                path="/"
                exact
                render={() => (!!localStorage.token ? <Redirect to="/main" /> : <Redirect to="/auth/login" />)}
            />
            <Route path="/auth" component={AuthContainer} />
            <Route path="/main" component={Container} />
            <Route render={() => <NotFound />} />
        </Switch>
    </App>
);

App.jsx:
import React from "react";
import "../style/main.scss";
import { Context as ResponsiveContext } from "react-responsive";
import { toast } from "react-toastify";

toast.configure();

const App = props => {
    return <ResponsiveContext.Provider>{props.children}</ResponsiveContext.Provider>;
};

export default App;

Container.jsx:
import React, { Component, Fragment } from "react";
import { Switch, Route, Redirect } from "react-router-dom";
import { connect } from "react-redux";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import Header from "../../components/Header/Header";
import Chat from "../../components/Chat/Chat";
import NotFound from "../../components/NotFound/NotFound";
import { closeErrorSnack } from "../../actions/appActions";
import MainPage from "./../../components/MainPage/MainPage";

class Container extends Component {
    componentDidUpdate(prevProps, prevState, snapshot) {
        if (
```

```
            prevProps.errorSnack !== this.props.errorSnack &&
            this.props.errorSnack
        ) {
            this.notifyError();
        }
        if (
            prevProps.successSnack !== this.props.successSnack &&
            this.props.successSnack
        ) {
            this.notifySuccess();
        }
    }

    notifyError = () =>
        toast.error(this.props.errorSnackText, {
            position: "top-right",
            autoClose: 5000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
        });

    notifySuccess = () =>
        toast.success(this.props.successSnackText, {
            position: "top-right",
            autoClose: 5000,
            hideProgressBar: false,
            closeOnClick: true,
            pauseOnHover: true,
            draggable: true,
            progress: undefined,
        });

    render() {
        const { match } = this.props;
        if (!localStorage.token) return <Redirect to="/auth/login" />;
        return (
            <Fragment>
                <Header />
                <main className="page">
                    <Switch>
                        <Route path={match.url} exact component={MainPage} />
                        <Route
                            path={`${match.url}/chat/:id`}
                            component={Chat}
                        />
                        <Route render={() => <NotFound />} />
                    </Switch>
                    <ToastContainer />
                </main>
            </Fragment>
        );
    }
}

const mapStateToProps = ({ app }) => {
    return {
        errorSnack: app.errorSnack,
        errorSnackText: app.errorSnackText,
```

```
        successSnack: app.successSnack,
        successSnackText: app.successSnackText,
    };
};

const mapDispatchToProps = {
    closeErrorSnack,
};

export default connect(mapStateToProps, mapDispatchToProps)(Container);
```

AuthContainer.jsx:

```
import React, { Component, Fragment } from "react";
import { Switch, Route, Redirect } from "react-router-dom";
import { connect } from "react-redux";

import Login from "../../components/Auth/Login/Login";
import NotFound from "../../components/NotFound/NotFound";
import Register from "../../components/Auth/Register/Register";

import { closeErrorSnack } from "../../actions/appActions";
import logo from "../../assets/image/logo.svg";
import "./AuthContainer.scss";

class AuthContainer extends Component {
    render() {
        const { match } = this.props;
        if (!!localStorage.token) return <Redirect to="/main" />;
        return (
            <Fragment>
                <main className="auth">
                    <h1 className="auth__logo">Crypto Chat</h1>
                    <div className="auth__container">
                        <Switch>
                            <Route
                                path={`${match.url}/login`}
                                exact
                                component={Login}
                            />
                            <Route
                                path={`${match.url}/register`}
                                exact
                                component={Register}
                            />
                            <Route render={() => <NotFound />} />
                        </Switch>
                    </div>
                </main>
            </Fragment>
        );
    }
}

const mapStateToProps = ({ app }) => {
    return {
        errorSnack: app.errorSnack,
        errorSnackText: app.errorSnackText,
    };
};

const mapDispatchToProps = {
```

```
    closeErrorSnack,
};

export default connect(mapStateToProps, mapDispatchToProps)(AuthContainer);
```

AuthContainer.scss:

```
.auth {
    display: flex;
    flex-direction: column;
    align-items: center;
    padding: 105px 20px;
    height: calc(100vh - 210px);

    &__logo {
        margin-bottom: 105px;
        max-height: 60px;
        font-size: 38px;
    }

    &__container {
        padding: 65px 80px 80px;
        background-color: var(--gray);
        border-radius: 16px;
        box-shadow: 0 4px 32px 0 rgba(0, 0, 0, 0.25);
    }
}

.auth_form {
    max-width: calc(480px - 160px);
    text-align: center;

    a {
        margin-top: 25px;
        display: block;
    }

    &__title {
        margin-bottom: 10px;
    }

    &__subtitle {
        margin-bottom: 65px;
    }

    .custom_input_wrapper {
        margin-bottom: 30px;
    }

    &__btn {
        margin-top: 5px;

        .default_button_wrapper {
            width: 100%;

            .default_button {
                width: 100%;
                min-width: auto;
            }
        }
    }
```

```
    &__main_error {
        color: var(--yellow);
    }
}
```

functions.js:

```
import React from "react";
import aesjs from "aes-js";

export function decrypt(text, key) {
    const encryptedBytes = aesjs.utils.hex.toBytes(text);
    const aesCtr = new aesjs.ModeOfOperation.ctr(key, new aesjs.Counter(5));
    const decryptedBytes = aesCtr.decrypt(encryptedBytes);
    return aesjs.utils.utf8.fromBytes(decryptedBytes);
}
```

Header.jsx:

```
import React, { Component } from "react";
import "./Header.scss";
import logo from "../../assets/image/logo.svg";
import { Link } from "react-router-dom";
import { connect } from "react-redux";
import { logout, getHeaderInfo } from "./../../actions/authActions";
import { push } from "connected-react-router";

class Header extends Component {
    componentDidMount() {
        const { getHeaderInfo } = this.props;
        getHeaderInfo();
    }

    render() {
        const { email, push } = this.props;
        return (
            <header className="header">
                <Link to={`/`}>
                    <p className="header__logo">Crypto Chat</p>
                </Link>
                <div className="header__user">
                    <span className="header__email">
                        {email && email.email}
                    </span>
                    <button
                        className="header__logout good_hover"
                        onClick={() => {
                            localStorage.removeItem("token");
                            push("/auth/login");
                        }}
                    >
                        Выйти
                    </button>
                </div>
            </header>
        );
    }
}

const mapStateToProps = ({ auth }) => {
    return {
        email: auth.email,
    };
```

```
};

const mapDispatchToProps = {
    logout,
    getHeaderInfo,
    push,
};

export default connect(mapStateToProps, mapDispatchToProps)(Header);
```

Header.scss:

```
.header {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    z-index: 999;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 30px 65px;
    background-color: var(--dark-gray);
    box-shadow: 0 2px 8px 0 rgba(0, 0, 0, 0.25);

    &__logo {
        font-size: 24px;
        font-family: var(--medium-font);
    }

    &__user {
        display: flex;
        align-items: center;
    }

    &__email {
        margin-right: 15px;
        font-family: var(--medium-font);
        font-size: 16px;
        line-height: 24px;
    }

    &__logout {
        font-size: 0;
        height: 16px;
        width: 16px;
        background-image: url("../../assets/image/logout.svg");
    }
}
```

Login.jsx:

```
import React, { Component } from "react";
import { Field, reduxForm, SubmissionError } from "redux-form";
// import { Field, reduxForm, SubmissionError } from 'redux-form';
import { connect } from "react-redux";
import ErrorIcon from "@material-ui/icons/Error";
import { postLogin } from "../../../actions/authActions";
import RenderField from "../../HelperComponents/RenderField/RenderField";
import DefaultButton from "../../HelperComponents/Buttons/DefaultButton/DefaultButton";
import TooltipMessage from "../../HelperComponents/TooltipMessage/TooltipMessage";
import { Link } from "react-router-dom";
```

```
class Login extends Component {
    componentDidMount() {
        const { history } = this.props;
        if (localStorage.token) {
            history.push("/main");
        }
    }

    submitForm = (data) => {
        const { postLogin, history } = this.props;
        return postLogin(data).then((res) => {
            if (
                res.payload &&
                res.payload.status &&
                res.payload.status === 200
            ) {
                history.push("/main");
            } else {
                throw new SubmissionError({
                    ...res.error.response.data,
                    _error: res.error.response.data.message,
                });
            }
        });
    };

    render() {
        const {
            handleSubmit,
            submitting,
            pristine,
            valid,
            authError,
            loading,
            error,
        } = this.props;

        return (
            <form
                className="auth_form"
                onSubmit={handleSubmit(this.submitForm)}
            >
                <h1 className="auth_form__title">Авторизация</h1>
                <p className="auth_form__subtitle">Войдите в свой аккаунт</p>
                <Field
                    name="email"
                    type="text"
                    component={RenderField}
                    label="Электронный адрес"
                />
                <Field
                    name="password"
                    type="password"
                    component={RenderField}
                    label="Пароль"
                />
                <div className="auth_form__main_error">{error}</div>
                <div className="auth_form__btn">
                    <DefaultButton
                        variant="contained"
                        disabled={submitting || pristine || !valid}
```

```
                    loading={loading}
                    formAction
                >
                    Войти
                </DefaultButton>
                {authError ? (
                    <TooltipMessage
                        text={authError}
                        delay={200}
                        error
                        position="right"
                        classes=""
                    >
                        <ErrorIcon />
                    </TooltipMessage>
                ) : (
                    ""
                )}
            </div>
            <Link to={"/auth/register"}>Регистрация</Link>
        </form>
    );
  }
}

const validate = (values) => {
  const errors = {};
  if (!values.email) {
    errors.email = "Обязательное поле";
  }
  if (!values.password) {
    errors.password = "Обязательное поле";
  } else if (values.password.length < 3) {
    errors.password = "Должно быть 3 или более символов";
  }
  return errors;
};

Login = reduxForm({
  form: "LoginForm",
  validate,
})(Login);

const mapStateToProps = ({ auth, app }) => {
  return {
    authError: auth.error_auth,
    loading: app.loading,
  };
};
const mapDispatchToProps = {
  postLogin,
};

export default connect(mapStateToProps, mapDispatchToProps)(Login);

Register.jsx:
import React, { Component } from "react";
import { Field, reduxForm, SubmissionError } from "redux-form";
// import { Field, reduxForm, SubmissionError } from 'redux-form';
import { connect } from "react-redux";
import ErrorIcon from "@material-ui/icons/Error";
```

```
import { register } from "../../../actions/authActions";
import RenderField from "../../HelperComponents/RenderField/RenderField";
import DefaultButton from "../../HelperComponents/Buttons/DefaultButton/DefaultButton";
import TooltipMessage from "../../HelperComponents/TooltipMessage/TooltipMessage";
import { Link } from "react-router-dom";

class Register extends Component {
    componentDidMount() {
        const { history } = this.props;
        if (localStorage.token) {
            history.push("/main");
        }
    }

    submitForm = (data) => {
        const { register, history } = this.props;
        return register(data).then((res) => {
            if (
                res.payload &&
                res.payload.status &&
                res.payload.status === 200
            ) {
                history.push("/auth/login");
            } else {
                throw new SubmissionError({
                    ...res.error.response.data,
                    _error: res.error.response.data.message,
                });
            }
        });
    };

    render() {
        const {
            handleSubmit,
            submitting,
            pristine,
            valid,
            authError,
            loading,
            error,
        } = this.props;

        return (
            <form
                className="auth_form"
                onSubmit={handleSubmit(this.submitForm)}
            >
                <h1 className="auth_form__title">Регистрация</h1>
                <p className="auth_form__subtitle">Создайте свой аккаунт</p>
                <Field
                    name="email"
                    type="text"
                    component={RenderField}
                    label="Электронный адрес"
                />
                <Field
                    name="username"
                    type="text"
                    component={RenderField}
                    label="Имя"
```

```
                />
                <Field
                    name="password"
                    type="password"
                    component={RenderField}
                    label="Пароль"
                />
                <div className="auth_form__main_error">{error}</div>
                <div className="auth_form__btn">
                    <DefaultButton
                        variant="contained"
                        disabled={submitting || pristine || !valid}
                        loading={loading}
                        formAction
                    >
                        Зарегистрироваться
                    </DefaultButton>
                    {authError ? (
                        <TooltipMessage
                            text={authError}
                            delay={200}
                            error
                            position="right"
                            classes=""
                        >
                            <ErrorIcon />
                        </TooltipMessage>
                    ) : (
                        ""
                    )}
                </div>
                <Link to={"/auth/login"}>Авторизация</Link>
            </form>
        );
    }
}

const validate = (values) => {
    const errors = {};
    if (!values.email) {
        errors.email = "Обязательное поле";
    }
    if (!values.username) {
        errors.username = "Обязательное поле";
    }
    if (!values.password) {
        errors.password = "Обязательное поле";
    } else if (values.password.length < 3) {
        errors.password = "Должно быть 3 или более символов";
    }
    return errors;
};

Register = reduxForm({
    form: "RegisterForm",
    validate,
})(Register);

const mapStateToProps = ({ auth, app }) => {
    return {
        authError: auth.error_auth,
```

```
        loading: app.loading,
    };
};
const mapDispatchToProps = {
    register,
};

export default connect(mapStateToProps, mapDispatchToProps)(Register);
```

Chat.jsx:

```
import React, { Component } from "react";
import RenderField from "../HelperComponents/RenderField/RenderField";
import { Field, reduxForm, reset, formValueSelector } from "redux-form";
import DefaultButton from "../HelperComponents/Buttons/DefaultButton/DefaultButton";
import "./Chat.scss";
import Messages from "./Messages";
import { connect } from "react-redux";
import {
    getChatHistory,
    getMoreChatMessages,
    getNewMessage,
    sendMessage,
    getInvites,
    postKey,
    saveDecrypted,
} from "../../actions/clientsActions";
import { toast } from "react-toastify";
import moment from "moment";
import ReconnectingWebSocket from "reconnecting-websocket";

import sendIcon from "../../assets/image/send.svg";
import NodeRSA from "node-rsa";
import { Buffer } from "buffer";
import aesjs from "aes-js";
import { decrypt } from "../../helpers/functions";
import Loader from "./../HelperComponents/Loader/Loader";

class Chat extends Component {
    state = {
        scrollTo: undefined,
        date: new Date(),
        decryptKey: null,
        loading: true,
    };
    rws = new ReconnectingWebSocket(
        `wss://crypto-chat-back.herokuapp.com/${
            this.props.match.params.id
        }/?token=${localStorage.getItem("token")}`
    );

    componentDidMount() {
        const {
            getChatHistory,
            getInvites,
            match: {
                params: { id },
            },
            getNewMessage,
            history,
            postKey,
            saveDecrypted,
```

```
} = this.props;
const { date } = this.state;

getInvites(id).then((res) => {
    if (
        res.payload &&
        res.payload.status &&
        res.payload.status === 200
    ) {
        if (
            res &&
            res.payload &&
            res.payload.data &&
            res.payload.data.isValid
        ) {
            const key = new NodeRSA({ b: 1024 });
            const pubKeyStr = key.exportKey("pkcs8-public");
            postKey(id, { key: pubKeyStr }).then((res) => {
                if (
                    res.payload &&
                    res.payload.status &&
                    res.payload.status === 200
                ) {
                    if (
                        res.payload &&
                        res.payload.data &&
                        res.payload.data.aesKey
                    ) {
                        const decryptedStr = key.decrypt(
                            res.payload.data.aesKey,
                            "hex"
                        );
                        const decrypted = Buffer.from(
                            decryptedStr,
                            "hex"
                        );
                        this.setState({
                            decryptKey: decrypted,
                        });
                        saveDecrypted(decryptedStr);

                        this.rws.addEventListener("open", () => {
                            console.log("Connected");
                        });

                        this.rws.addEventListener(
                            "message",
                            function (event) {
                                const obj = JSON.parse(event.data);
                                getNewMessage({
                                    ...obj,
                                    text: decrypt(obj.text, decrypted),
                                });
                            }
                        );
                        getChatHistory(
                            id,
                            moment(date).format("yyyy-MM-DDTHH:mm:ss"),
                            1
                        ).then((res) => {
                            if (
```

```
                            res.payload &&
                            res.payload.status &&
                            res.payload.status === 200
                        ) {
                            this.setState({ loading: false });
                        } else {
                            toast(
                                "Something went wrong. Try again, please!",
                                {
                                    progressClassName:
                                        "red-progress",
                                }
                            );
                        }
                    });
                }
            } else {
                history.push("/main");
                toast("Something went wrong. Try again, please!", {
                    progressClassName: "red-progress",
                });
            }
        });
    } else {
        history.push("/main");
        toast("У вас нету доступа к этой комнате!", {
            progressClassName: "red-progress",
        });
    }
} else {
    history.push("/main");
    toast("ID комнаты неправильное!", {
        progressClassName: "red-progress",
    });
}
});
}

componentWillUnmount() {
    this.rws.close();
}

submitForm = (data) => {
    const { reset } = this.props;
    const { text } = data;
    const { decryptKey } = this.state;
    const textBytes = aesjs.utils.utf8.toBytes(text);
    const aesCtr = new aesjs.ModeOfOperation.ctr(
        decryptKey,
        new aesjs.Counter(5)
    );
    const encryptedBytes = aesCtr.encrypt(textBytes);
    const encryptedHex = aesjs.utils.hex.fromBytes(encryptedBytes);

    this.rws.send(encryptedHex);
    reset();
};

setScrollTo = (scrollTo) => this.setState({ scrollTo });

handleScroll = async (e) => {
```

```
    const { messagesList, getMoreChatMessages } = this.props;
    if (e.target.scrollTop === 0 && messagesList.next) {
      const previousHeight = e.target.scrollHeight;
      let nextUrl = messagesList.next.split("api/")[1];
      await getMoreChatMessages(nextUrl);
      this.setScrollTo(previousHeight);
    }
};

handleFile = (e) => {
    const file = e.target.files[0];
    if (file) {
      this.setState({ file });
    }
};

render() {
    const { scrollTo, file, decryptKey, loading } = this.state;
    const {
      client,
      messagesList,
      handleSubmit,
      textValue,
      history,
      email,
      match: {
        params: { id },
      },
    } = this.props;
    return (
      <div className="chat_page page_wrapper">
        <header className="chat_page__header section_header">
          <div
            className="back_link"
            onClick={() => history.goBack()}
            aria-label="Вернуться назад"
          />
          <h1 className="chat_page__title">Room №{id}</h1>
        </header>
        {loading ? (
          <div style={{ marginTop: "220px" }}>
            <Loader />
          </div>
        ) : (
          <Messages
            messages={messagesList}
            handleScroll={this.handleScroll}
            scrollTo={scrollTo}
            setScrollTo={this.setScrollTo}
            loading={loading}
            myEmail={email && email.email}
            decryptKey={decryptKey}
            decrypt={decrypt}
          />
        )}

        <div className="chat_page__send_wrap">
          <form
            className="chat_page__send"
            onSubmit={handleSubmit(this.submitForm)}
          >
```

```
                    <Field
                        name="text"
                        type="text"
                        component={RenderField}
                        placeholder="Введите сообщение..."
                    />
                    <DefaultButton
                        formAction
                        disabled={!textValue && !file}
                    >
                        <span>Отправить</span>
                        <img src={sendIcon} alt="Отправить" />
                    </DefaultButton>
                </form>
            </div>
        </div>
    );
  }
}

const validate = (values) => {
  const errors = {};

  return errors;
};

Chat = reduxForm({
  form: "ChatForm",
  validate,
})(Chat);

const selector = formValueSelector("ChatForm");

const mapStateToProps = (state) => {
  return {
    client: state.clients.singleClient,
    messagesList: state.clients.messages,
    loading: state.clients.loading,
    invites: state.clients.invites,
    textValue: selector(state, "text"),
    email: state.auth.email,
  };
};
const mapDispatchToProps = {
  getChatHistory,
  getMoreChatMessages,
  getNewMessage,
  sendMessage,
  getInvites,
  postKey,
  saveDecrypted,
  reset: () => reset("ChatForm"),
};

export default connect(mapStateToProps, mapDispatchToProps)(Chat);

Chat.scss:
.chat_file_preview  {
  margin-left: 10px;
}
```

```
.chat_page {
    position: relative;
    display: flex;
    flex-direction: column;
    padding-bottom: 30px;
    height: calc(100vh - 180px);
    background-image: url("../../assets/image/chat-bg.png");
    background-size: cover;
    background-position: center;

    &__header {
        margin-bottom: 25px;
        padding-bottom: 20px;
        border-bottom: 1px solid rgba(255, 255, 255, 0.1);

        .default_button_wrapper {
            margin-left: auto;
        }
    }

    &__avatar {
        margin: 0 15px;
    }

    &__messages {
        width: calc(100% - 35px);
        padding-top: 5px;
        padding-left: 30px;
        display: flex;
        flex-direction: column-reverse;

        &::-webkit-scrollbar {
            width: 3px;
            height: 5px;
        }

        &::-webkit-scrollbar-track {
            -webkit-border-radius: var(--small-radius);
            background: transparent;
        }

        &::-webkit-scrollbar-thumb {
            -webkit-border-radius: var(--small-radius);
            border-radius: var(--small-radius);
            background: var(--gray);
        }

        &::-webkit-scrollbar-thumb:window-inactive {
            background: var(--gray);
        }
    }

    &__send_wrap {
        margin-top: auto;
        padding-top: 40px;
    }

    &__send {
        display: flex;
        justify-content: space-between;
        align-items: center;
```

```
    padding: 20px 25px;
    width: calc(100% - 50px);
    background: #2a2f38;
    border-radius: 16px;

    .custom_input_wrapper {
        margin-left: 25px;
        margin-right: 15px;
    }

    .default_button {
        min-width: 155px;

        .MuiButton-label {
            display: flex;
            justify-content: center;

            img {
                margin-left: 10px;
            }
        }
    }
}

.time_separator {
    display: flex;
    justify-content: center;
    opacity: 0.5;
    font-size: 14px;
    font-family: var(--medium-font);
    margin: 10px 0 35px 0;
}

.message {
    position: relative;
    padding: 25px 30px 30px;
    margin-bottom: 50px;
    width: fit-content;
    max-width: calc(545px - 55px);
    background-color: #2a2f38;
    border-radius: 16px;
    max-width: calc(545px - 55px);
    min-width: 110px;
    > img {
        object-fit: contain;
        width: 100%;
        height: 100%;
        margin-bottom: 20px;
    }

    &:first-of-type {
        margin-bottom: 0;
    }

    &:before {
        content: "";
        position: absolute;
        top: -5px;
        left: -8px;
        transform: rotate(-45deg);
        width: 0;
```

```
    height: 0;
    border-style: solid;
    border-width: 0 10px 15px 10px;
    border-color: transparent transparent #2a2f38 transparent;
}

&--your {
    margin-left: auto;
    margin-right: 10px;

    &:before {
        left: auto;
        right: -8px;
        transform: rotate(45deg);
    }
}

&__author {
    position: absolute;
    top: -25px;
    right: 7px;
    color: var(--sub-color);
}

&__time {
    position: absolute;
    bottom: 5px;
    right: 15px;
    display: flex;
    align-items: center;
    color: var(--sub-color);
}

&__status {
    opacity: 0.5;
    display: inline-flex;
    margin-left: 5px;
    margin-right: -7px;
    width: 22px;
    height: 18px;
    background-image: url("../../assets/image/status-read.svg");

    &.read {
        opacity: 1;
    }
}
&__file {
    background: #363945;
    border-radius: 8px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    > span {
        word-wrap: break-word;
        word-break: break-all;
        padding: 12px 0;
    }
    .first_img {
        margin-right: 9px;
        padding-left: 19px;
    }
```

```
            .second_img {
                padding: 12px 16px;
                margin-left: 30px;
                border-left: 1.5px solid #2a2d35;
            }
        }
    }

    .with_image {
        padding: 0;
        padding-bottom: 30px;
        > img {
            border-top-left-radius: 16px;
            border-top-right-radius: 16px;
        }
        &:before {
            display: none;
        }
        .message__text {
            padding: 0 30px;
        }
    }
}

.dialog_custom_img {
    // width: 680px;
    padding: 56px 0 0 0;
    display: flex;
    justify-content: center;
    align-items: center;
    .image_dialog {
        width: 100%;
        display: flex;
        .big_icon {
            width: 100%;
            height: 100%;
            display: flex;
            justify-content: center;
            align-items: center;
            img {
                background-size: 100% 100%;
                width: 100%;
                height: 100%;
            }
        }
    }
}
```

Messages.js:

```
import React, { Fragment, useEffect, useRef, useState } from "react";
import { Scrollbars } from "react-custom-scrollbars";
import moment from "moment";
import "./Chat.scss";

import { Buffer } from "buffer";

const Messages = ({
    messages,
    handleScroll,
    scrollTo,
    setScrollTo,
```

```
    loading,
    myEmail,
    decryptKey,
    decrypt,
}) => {
    const scrollbarRef = useRef(null);
    useEffect(() => {
        scrollbarRef.current.scrollToBottom();
    }, []);

    useEffect(() => {
        scrollbarRef.current.scrollToBottom();
    }, [loading]);

    const handleUpdate = () => {
        if (scrollTo) {
            scrollbarRef.current.scrollTop(
                scrollbarRef.current.getScrollHeight() - scrollTo
            );
            setScrollTo(undefined);
        }
    };
    return (
        <>
            <Scrollbars
                onScroll={handleScroll}
                hideTracksWhenNotNeeded
                ref={scrollbarRef}
                onUpdate={handleUpdate}
            >
                <div className="chat_page__messages">
                    {messages &&
                        messages.results &&
                        messages.results.map(
                            (
                                { id, text, create_date, image, file, sender },
                                i
                            ) => {
                                const lastIndex = messages.results.length - 1;
                                const timeNow =
                                    moment(create_date).format("YYYY:MM:DD");
                                let timeNext = timeNow;
                                const today = moment().format("YYYY-MM-DD");
                                const isToday =
                                    today === create_date.slice(0, 10);
                                if (i + 1 <= lastIndex) {
                                    timeNext = moment(
                                        messages.results[i + 1].create_date
                                    ).format("YYYY:MM:DD");
                                }
                                return (
                                    <Fragment key={i}>
                                        <p
                                            key={id}
                                            className={`message ${
                                                myEmail !== sender
                                                    ? ""
                                                    : "message--your"
                                            } ${file ? "with_file" : ""} ${
                                                image ? "with_image" : ""
                                            }`}
```

```
                                >
                                    <span className="message__author">
                                        {sender}
                                    </span>
                                    <p className="message__text">
                                        {text}
                                    </p>
                                    <span className="message__time">
                                        {moment(create_date).format(
                                            "HH:mm"
                                        )}
                                        {myEmail === sender && (
                                            <span
                                                className="message__status read"
                                                aria-label="Статус сообщения"
                                            />
                                        )}
                                    </span>
                                </p>
                                {timeNow !== timeNext ||
                                i === lastIndex ? (
                                    <div className="time_separator">
                                        {isToday
                                            ? "Today"
                                            : moment(
                                                create_date
                                              ).format("MMMM DD, YYYY")}
                                    </div>
                                ) : null}
                            </Fragment>
                        );
                    }
                )}
            </div>
        </Scrollbars>
    </>
  );
};

export default Messages;

Header.jsx:
import React, { Component } from "react";
import "./Header.scss";
import logo from "../../assets/image/logo.svg";
import { Link } from "react-router-dom";
import { connect } from "react-redux";
import { logout, getHeaderInfo } from "./../../actions/authActions";
import { push } from "connected-react-router";

class Header extends Component {
  componentDidMount() {
    const { getHeaderInfo } = this.props;
    getHeaderInfo();
  }

  render() {
    const { email, push } = this.props;
    return (
      <header className="header">
        <Link to={`/`}>
```

```
                <p className="header__logo">Crypto Chat</p>
            </Link>
            <div className="header__user">
                <span className="header__email">
                    {email && email.email}
                </span>
                <button
                    className="header__logout good_hover"
                    onClick={() => {
                        localStorage.removeItem("token");
                        push("/auth/login");
                    }}
                >
                    Выйти
                </button>
            </div>
        </header>
    );
  }
}

const mapStateToProps = ({ auth }) => {
  return {
    email: auth.email,
  };
};

const mapDispatchToProps = {
  logout,
  getHeaderInfo,
  push,
};

export default connect(mapStateToProps, mapDispatchToProps)(Header);

Header.scss:
.header {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  z-index: 999;
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 30px 65px;
  background-color: var(--dark-gray);
  box-shadow: 0 2px 8px 0 rgba(0, 0, 0, 0.25);

  &__logo {
    font-size: 24px;
    font-family: var(--medium-font);
  }

  &__user {
    display: flex;
    align-items: center;
  }

  &__email {
    margin-right: 15px;
```

```
        font-family: var(--medium-font);
        font-size: 16px;
        line-height: 24px;
    }

    &__logout {
        font-size: 0;
        height: 16px;
        width: 16px;
        background-image: url("../../assets/image/logout.svg");
    }
}
```

MainPage.js:

```
import React, { Component } from "react";
import { connect } from "react-redux";
import "./MainPage.scss";
import DialogComponent from "../HelperComponents/DialogComponent/DialogComponent";
import { createRoom } from "./../../actions/clientsActions";
import { toast } from "react-toastify";

class MainPage extends Component {
    state = {
        value: "",
        openDialog: false,
        emailsValue: "",
    };
    render() {
        const { value, openDialog, emailsValue } = this.state;
        const { createRoom, history } = this.props;
        return (
            <div className="main-page">
                <h1>Crypto Chat</h1>
                <p className="main-page-desc">
                    Безопасный чат с шифрованием данных
                </p>
                <div className="main-page-create">
                    <span>Вы можете создать новую комнату</span>
                    <button
                        onClick={() => {
                            this.setState({ openDialog: true });
                        }}
                    >
                        Создать
                    </button>
                </div>
                <div className="main-page-or">или</div>
                <div className="main-page-find">
                    <p>Вы можете зайти в уже существующую комнату</p>
                    <div>
                        <input
                            placeholder="ID комнаты"
                            value={value}
                            onChange={(e) =>
                                this.setState({ value: e.target.value })
                            }
                        />
                        <button
                            onClick={() => history.push(`/main/chat/${value}`)}
                        >
                            Зайти
```

```
                    </button>
                </div>
            </div>
            <DialogComponent
                open={openDialog}
                onClose={() => {
                    this.setState({ openDialog: false, emailsValue: "" });
                }}
                classes="dialog_custom_img"
            >
                <div className="dialog-create">
                    <h2>Создание комнаты</h2>
                    <p>
                        Введите email людей, которым вы хотите дать доступ к
                        комнате
                    </p>
                    <input
                        placeholder="Emails"
                        value={emailsValue}
                        onChange={(e) =>
                            this.setState({ emailsValue: e.target.value })
                        }
                    />
                    <button
                        onClick={() => {
                            createRoom({
                                invites: [...emailsValue.split(', ')],
                            }).then((res) => {
                                if (
                                    res.payload &&
                                    res.payload.status &&
                                    res.payload.status === 200
                                ) {
                                    history.push(
                                        `/main/chat/${res.payload.data.roomId}`
                                    );
                                } else {
                                    toast(
                                        "Something went wrong. Try again, please!",
                                        {
                                            progressClassName:
                                                "red-progress",
                                        });}});
                        }}
                    >
                        Создать
                    </button>
                </div>
            </DialogComponent>
        </div>
    );
  }
}

const mapStateToProps = ({}) => {
  return {};
};

const mapDispatchToProps = { createRoom };

export default connect(mapStateToProps, mapDispatchToProps)(MainPage);
```