МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ МЕСЕНДЖЕРУ ЗА КОНЦЕПЦІЄЮ WEB3

Здобувач освіти гр. ІН.м-13 Максим МИКИТЧЕНКО

Науковий керівник, Надія ТИРКУСОВА
доцент, кандидат фізико-математичних наук

В.о. завідувача кафедри Ігор ШЕЛЕХОВ
доцент, кандидат технічних наук

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122-Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Микитченко Максиму Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проектування месенджеру за концепцією Web3*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Інформаційний огляд 2) Вибір методів реалізації 3) Практична реалізація
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1	*Інформаційний огляд*	*01.09 – 26.09*	
2	*Вибір методів реалізації*	*27.09 – 30.10*	
3	*Практична реалізація*	*03.11 – 10.12*	

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 49 сторінок, 25 рисунків, 1 додаток, 15 джерел.

Об’єкт дослідження — концепція Web3, технологія блокчейн, розробка веб-додатків, криптовалюта.

Мета роботи — розробка месенджеру за концепцією Web3.

Результати — було розроблено месенджер за концепцією Web3. Це онлайн чат, з авторизацією через криптогаманець, з можливостями взаємодії з криптовалютою. Для клієнтської частини було використано фреймворк React JS, для серверної – фреймворк Express JS.

WEB3, МЕСЕНДЖЕР, КРИПТОГАМАНЕЦЬ, КРИПТОВАЛЮТА, БЛОКЧЕЙН, ВЕБ-ДОДАТКИ, ETHEREUM, EVM, NODE.JS, EXPRESS.JS, REACTJS, MONGODB, WEBSOCKET API.

ЗМІСТ

ВСТУП

Сьогодні важко уявити своє життя без інтернету. Кожного дня ми заходимо до мережи, щоб поспілкуватися зі знайомими, подивитися новини чи замовити собі якийсь одяг. Заходячи кожного разу на новий сайт, ми повинні зареєструватися для того, щоб виконувати якісь дії. Це займає якийсь час на реєстрацію, а також ми кожного разу надаємо свої дані якимось сайтам, що не є дуже безпечно в наш час. Зараз мережа працює за концепцією Web2, цей етап почався досить давно, з появою соцмереж. Але ми все ближче до нової версії інтернету – Web3.

Web3 може дуже полегшити наше життя, нам не потрібно буде реєструватися на різних сайтах та передавати свої дані стороннім розробникам. Ми будемо мати один криптогаманець та так сказати один профіль для усієї мережі. Наші дані будуть децентралізовані.

В цій роботі я більш детально розгляну концепцію Web3, розповім, що таке криптовалюта та блокчейн, зроблю огляд криптогаманців.

Також ми кожного дня використовуємо месенджери для спілкування у мережі. Там ми можемо спілкуватися з друзями, родичами, тощо.

У цій роботі я вирішив поєднати звичайний месенджер та концепцію Web3.

Спочатку я опишу вибрані технології для розробки цього додатку, розкажу, який блокчейн та який криптогаманець, я обрав для роботи з цим додатком.

Після цього я розроблю веб-додаток, це буде месенджер, в якому авторизація буде реалізована за допомогою криптогаманця. Також у цьому месенджері кожний користувач зможе надсилати криптовалюту будь-якому іншому користувачу дуже легко. Користувачі також зможуть створювати платний контент, який потім можна буде розблокувати іншим користувачам за криптовалюту.

Я вважаю, що зовсім скоро ми повністю перейдемо на мережу Web3. Наприклад, Telegram вже робить перші кроки в цьому напрямку.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Постановка задачі

В цій кваліфікаційній роботі мені потрібно розробити веб-додаток. Цей веб-додаток повинен бути месенджером за концепцією Web3.

Необхідний функціонал додатку:

- Авторизація за допомогою криптогаманця
- Можливість користувач створювати кімнати для спілкування та приєднуватися до них
- Можливість спілкування користувачів у режимі реального часу
- Можливість користувачів надсилати криптовалюту в додатку
- Можливість користувачів створювати платний контент, який можна розблокувати іншим користувачам за криптовалюту

1.2 Що таке Web3

Централізація допомогла мільярдам людей підключитися до всесвітнього павутиння та створила стабільну та надійну інфраструктуру, на якій вона живе. У той же час жменька централізованих організацій зміцнилася на великих ділянках всесвітнього павутиння, в односторонньому порядку вирішуючи, що можна, а що не можна. Web3 є відповіддю на цю дилему. Замість Інтернету, монополізованого великими технологічними компаніями, Web3 використовує децентралізацію та створюється, керується та належить користувачам. Web3 віддає владу до рук окремих осіб, а не корпорацій. Перш ніж ми поговоримо про Web3, розглянемо, як ми до цього дійшли.

Багато людей вважають, що мережа це незамінна річь в сучасному житті — вона була розроблена і існує досі з тих часів. Сьогодні інтернет відомий усім з нас, досить сильно відрізняється від початкової версії. Щоб зрозуміти наскільки, історію Інтернету потрібно розбити на такі етапи — Веб 1.0 і Веб 2.0.

1989 року в CERN у Женеві Тім Бернерс-Лі займався розробкою протоколів, які згодом стали всесвітнім павутинням. Його ідея? Створити відкриті децентралізовані протоколи, що дозволяють обмінюватися інформацією в будь-якій точці Землі. Перший винахід Бернерса-Лі, тепер відоме як «Веб 1.0», відбулося приблизно між 1990 та 2004 роками. Веб 1.0 був переважно статичними веб-сайтами, що належать компаніям, і взаємодія між користувачами була практично нульовою — окремі особи мало коли створювали контент — що призводило до того, що Веб 1.0 відомий як мережа тільки для читання.

Період Веб 2.0 розпочався 2004 року з появою соціальних мереж. Замість режиму лише для читання мережа стала доступною для читання та запису. Замість того, щоб компанії надавали контент користувачам, вони також почали надавати платформи для обміну контентом користувача та взаємодії між користувачами. У міру того, як все більше людей виходило в мережу, жменька провідних компаній почала контролювати непропорційно великий обсяг трафіку та цінності, що створюється в мережі. Веб 2.0 також породив модель доходів, що базується на рекламі. Хоча користувачі могли створювати контент, вони не володіли ним і не отримували вигоди від його монетизації.

Передумова «Веб 3.0» була вигадана співзасновником Ефіріуму Гевіном Вудом невдовзі після запуску Ефіріуму у 2014 році. Гевін висловив словами рішення проблеми, яку відчували багато ранніх прихильників криптографії: Мережа вимагає надто великої довіри. Тобто більшість Мережі, яку люди знають і використовують сьогодні, заснована на довірі до жменьки приватних компаній, які діють на користь суспільства.

Web3 став універсальним терміном для позначення нового, найкращого Інтернету. За своєю суттю Web3 використовує блокчейни, криптовалюти та NFT, щоб повернути владу користувачам у формі власності. У повідомленні 2020 року в Твіттері це сказано найкраще: Web1 був доступний тільки для читання, Web2 - для читання-запису, Web3 - для читання-запису-власності.

Головні ідеї Web3:

- Web3 децентралізований: замість великих ділянок Інтернету, що контролюються та належать централізованим організаціям, право власності розподіляється між його творцями та користувачами.
- Web3 не має дозволів: усі мають рівний доступ до участі в Web3, і ніхто не виключається.
- Web3 має нативні платежі: він використовує криптовалюту для витрат і відправки грошей в Інтернеті замість того, щоб покладатися на застарілу інфраструктуру банків і платіжних систем.
- Web3 не вимагає довіри: він працює, використовуючи стимули та економічні механізми, замість покладатися на довірених третіх осіб.[1]

1.2.1 Володіння

Web3 безпрецедентним чином дає вам право власності на цифрові активи. Наприклад, скажімо, ви граєте у гру web2. Якщо ви купуєте внутрішньоігровий предмет, він безпосередньо прив'язується до вашого облікового запису. Якщо творці гри видалять ваш обліковий запис, ви втратите ці предмети. Або якщо ви припините грати в гру, ви втратите цінність, яку вклали у свої внутрішньоігрові предмети. Web3 допускає пряме володіння через невзаємозамінні токени (NFT). Ніхто навіть творці гри не мають права позбавити вас права власності. А якщо ви припините грати, ви зможете продати або обміняти свої ігрові предмети на відкритих ринках і відшкодувати їх вартість.

1.2.2 Опір цензурі

У Web3 ваші дані зберігаються у блокчейні. Коли ви вирішите залишити платформу, ви можете взяти з собою свою репутацію, підключивши її до іншого інтерфейсу, який чіткіше відповідає вашим цінностям.

Web 2.0 вимагає, щоб творці контенту довіряли платформам і не змінювали правила, але стійкість до цензури є природною особливістю платформи Web3.[2]

1.2.3 Децентралізовані автономні організації

Окрім володіння своїми даними у Web3, ви можете володіти платформою як колективом, використовуючи токени, які діють як акції компанії. DAO дозволяють вам координувати децентралізоване володіння платформою та приймати рішення про її майбутнє. Технічно DAO визначаються як узгоджені смарт-контракти, які автоматизують децентралізоване ухвалення рішень щодо пулу ресурсів (токенів). Користувачі з токенами голосують за те, як витрачаються ресурси, а код автоматично виконує результати голосування. Проте люди визначають багато спільнот Web3 як DAO. Всі ці спільноти мають різні рівні децентралізації та автоматизації за допомогою коду. В даний час ми вивчаємо, що таке DAO і як вони можуть розвиватись у майбутньому.

1.2.4 Особистість

Традиційно ви створюєте обліковий запис для кожної використовуваної платформи. Наприклад, у вас може бути обліковий запис Twitter, обліковий запис YouTube та обліковий запис Reddit. Бажаєте змінити відображене ім'я або зображення профілю? Ви повинні зробити це для кожного облікового запису. У деяких випадках ви можете використовувати вхід через соціальні мережі, але це створює знайому проблему – цензуру. Одним клацанням миші ці платформи можуть заблокувати вас від усього вашого онлайн-життя. Найгірше, багато платформ вимагають, щоб ви довіряли їм особисту інформацію для створення облікового запису. Web3 вирішує ці проблеми, дозволяючи вам контролювати свою цифрову ідентифікацію за допомогою адреси Ethereum та профілю ENS. Використання адреси Ethereum забезпечує єдиний вхід на різних платформах, який є безпечним, стійким до цензури та анонімним.

1.2.5 Нативні платежі

Платіжна інфраструктура Web2 спирається на банки та платіжні системи, за винятком людей без банківських рахунків або тих, хто випадково проживає в

межах інших країн. Web3 використовує токени, такі як ETH, для відправки грошей безпосередньо у браузері і не вимагає довіреної третьої сторони.

Web3 — молода екосистема, що розвивається. Гевін Вуд придумав цей термін у 2014 році, але багато з цих ідей стали реальністю лише нещодавно. Лише за останній рік стався значний сплеск інтересу до криптовалюти, удосконалення рішень для масштабування другого рівня, масштабні експерименти з новими формами управління та революції у цифровій ідентифікації.[3]

1.3 Криптовалюта

Як можна було зрозуміти, Web3 має дуже багато спільного з криптовалютою. Сьогодні про криптовалюту можна почути звідусіль, останні роки це стало трендом. Кожного дня криптовалюта стає все популярнішою, але багато людей досить погано розуміють, що це таке. Я спробую вас ознайомити з поняттями криптовалюта та блокчейн.[4]

1.3.1 Що взагалі таке криптовалюта

Криптовалюта – це цифрова валюта, яка заснована на технології блокчейну. Порівнюючи з звичайними фіатними валютами, такими як гривня, долар, немає головного органу, який регулює вартість криптовалюти. Цим займаються звичайні користувачі та холдери криптовалюти в інтернеті.

За допомогою криптовалюти можна сплачувати послуги та товари, якщо це дозволяє робити виробник. Сьогодні ж більшість користувчів лише інвестують у криптовалюту, не використовуючи інші функції.

Я сказав, що криптовалюта заснована на технології блокчейн. Що ж це таке? Якщо дослівно перекладати слово блокчейн - це неперервний ланцюг блоків. У цьому ланцюгу містяться записи про угоди користувачів. Порівнюючи зі звичайними базами даних, редагувати ці записи не можна, є можливість лише створити нові.[5]

Блокчейн також називають спеціальною технологією дискретних реєстрів, у блокчейні ланцюг угод та список власників транзакцій мають на власних комп'ютерах усі користувачів цього блокчейну в інтернеті. У випадку коли декілька комп'ютерів користувачів не будуть працювати, інформація не зникне. Програма блокчейну зберігає кожну транзакцію користувача, коли вона надходить, також усі копії блокчейну оновлюється разом з надходженням нової інформації. При такому налаштуванні повністю всі записи будуть ідентичними.

Для запобігання шахрайства, кожна транзакція користувача перевіряється, використовуючи один основних методів перевірки блокчейну: PoS або PoW.

Щодо різниці між криптовалютою та блокчейном. За криптовалютою біткоїн стоїть блокчейн, відомий як Bitcoin. Альткоїн Ефіріум працює в базі даних блокчейну під назвою Ethereum. Альткоїн Litecoin має власний блокчейн, який є похідним від відкритого блокчейну Bitcoin. Стейблкоін Tether не має власного блокчейну. Токени Tether існують у блокчейні Ethereum, і саме там реєструються транзакції Tether.[6]

1.3.2 POW and POS

Proof-of-Work (PoW) використовується тоді, коли технічне обладнання майнера вирішує складні математичні завдання. За додавання перевіреного блоку в блок-ланцюжок майнер отримує винагороду у вигляді криптовалюти. Пошук рішень – це складний процес, що потребує значних обчислювальних потужностей. Як тільки комп'ютер знаходить рішення, він надсилає повідомлення іншим комп'ютерам спільноти для перевірки. Рішення легко перевірити, тому що іншим комп'ютерам надається відповідь.

На відміну від Proof-of-Work, де алгоритм винагороджує майнерів, які проводять обчислення для валідації транзакцій та створення нових блоків, у Proof-of-Stake творець нового блоку вибирається системою заздалегідь на підставі його стану, тобто частки загальної кількості криптовалюти.

Ідея PoS вирішує проблеми PoW, пов'язані з витратами великої кількості електроенергії. Замість обчислювальних потужностей учасників має значення

кількість криптовалюти, яка перебуває у них на рахунку. Так, замість використання великої кількості електроенергії для вирішення завдання PoW, учасник PoS обмежений відсотком можливих перевірок транзакцій. Обмеження відповідає кількості криптовалюти, що знаходиться на рахунку учасника.[7]

1.3.3 Різниця Coin та Token

Криптовалюту можна поділити на такі типи: Coin та Token.

Монетами називають будь-які криптовалюти, які працюють на базі окремого незалежного блокчейну. Такі криптовалюти створюються з нуля, причому їх мережі проектуються спеціально для досягнення певної мети.

Токени - це унікальне продовження платформ для смарт-контрактів на кшталт Ефіріуму. Останні дають можливість користувачам створювати, випускати та взаємодіяти з токенами, які в результаті виявляються похідними основного блокчейну.

Монети в свою чергу можна поділити на біткоїн та альткоїни.

Біткоїн – перша створена криптовалюта на основі блокчену, анонсована у 2008 році і створена та запущена у 2009 році. Серед переваг біткоїна, які успадкували інші криптовалюти:

- Анонімність криптовалюти;
- Незворотність криптовалюти;
- Прозорість криптовалюти;

Усі криптовалюти, які були створені після Bitcoin це альтернативні монети. Після створення першої у світі криптовалюти, велика кількість команд почали розробляти свої альтернативні проекти. Кожен новий проект повинен був виправляти якийсь недолік біткоїну або вирішувати якусь проблему в досить старій фінансовій системі. Але до сьогодні кількість альтернативних коїнів становила вже кілька десятків або сотень тисяч. В даний момент багато з цих проектів не переслідує корисних цілей, а використовуються просто для маніпуляцій ринком. Не всі альтернативні коїни мають схожості з Bitcoin.

Наприклад, другий по популярності блокчейн Ethereum був розроблений для створення децентралізованих додатків. Але зараз більшість криптовалют - це надбудови над Bitcoin або Ethereum та переосмислення їх вихідного коду.

Також нещодавно почали виокремлювати стрейблкоїни. Стейблкоїни Це криптовалюти з низьким рівнем волатильності. За своїм задумом вони повинні бути підкріплені іншими активами, якими і забезпечується стабільність курсу. Наприклад, найпопулярніший і найбільший за обсягом ринкової капіталізації стейблкоїн - Tether (USDT). Спочатку стверджувалося, що тезер повністю забезпечений американським доларом 1:1, проте пізніші звіти повідомляють про наявність також комерційних паперів, банківських депозитів та інших фінансових продуктів у резерві компанії Tether Limited, яка випускає USDT.

Зі зростанням популярності, а особливо зі збільшенням обсягу ринкової капіталізації USDT багато відомих і не дуже проектів стали створювати свої стейблкоїни. Цей тип криптовалюти дуже зручний для представлення ціни. У криптоіндустрії, де багато проектів взагалі не стикаються з фіатними грошима (традиційними валютами — американськими доларами, євро та іншими), набагато частіше можна зустріти значення курсу до USDT, ніж до USD, не кажучи про інші національні валюти. Але стейблкоїн це також альткоїн.

Шодо токенів. Токени існують у різних галузях, не тільки у блокчейн-технологіях. Взагалі токен — одиниця обліку, яку можна вважати спеціальною нотаткою у якійсь інформаційній системі. Криптотокени - це запис спеціального виду в блокчейні. Криптотокени відрізняються від популярних криптовалют тим, що їх архітектура будується не на власному блокчейні, а на блокчейнах інших криптовалют, наприклад, Ethereum або Solana. Криптотокени можна вважати цифровою реалізацією цінних паперів або активів іншого виду.

Криптотокени можна поділити на підвиди за призначенням або унікальністю. Токени бувають взаємозамінними або незамінними. Незамінні токени також називають NFT. NFT існують у одному екземплярі і можуть закріпляти право власності за будь-яким предметом. Усі інші токени, які випускаються в будь-якій кількості, але значення кожного токену дорівнює

значенню будь-якого іншого токену називаються взаємозамінні криптотокенами. Взаємозамінні токени дуже схожі на звичайні фіатні валюти.[8]

1.3.4 Способи отримання криптовалюти

Способи отримання криптовалюти можна поділити на майнінг, стейкінг та купівлю.

Майнінг це процес накопичення нових криптокоїнів у блокчейні та перевірка на правдивість нових транзакцій. Комп'ютери проводять перевірку та виконують документацію транзакцій, після цього отримують винагороду, тобто нові монети. Для того, щоб отримати коїни, комп'ютери мають вирішити певну криптографічну задачу. Комп'ютер користувача, який зробить це першим, отримає за це криптовалюту. Потужні комп'ютери мають більше шансів розв'язати задачу і створити новий блок. Майнінг потребує багато енергії. Майнінг можливий, якщо блокчейн працює на Proof of Work. Прикладом такого блокчейну є Bitcoin.

Стейкінг є більш екологічною альтернативою майнінгу, бо стейкінг не залежить від великих потужностей комп'ютера користувача. Стейкінг не потребує великої витрати електроенергії. Стейкінг заснований на механізмі proof of stake. Використовуючи прицип PoS є справедливим принцип: більша сума зберігається на гаманці у користувача - більше блоків має шанс створити користувач. За кожний створений новий блок користувач отримає винагороду – монети блокчейну. Наприклад Etherium нещодавно з Proof of Work перейшов на Proof of Stake.

Або є можливість купити криптовалюту. Це можна зробити на криптобіржах. Прикладами криптобірж є Binance та Coinbase. Для купівлі криптовалюти на біржі потрібно зареєструватися на одній з них, поповнити рахунок та вибрати, яку саме криптовалюту купити.[9]

1.3.5 Криптогаманці

Криптовалюту десь потрібно зберігати. Для цього використовують криптогаманці. За допомогою нього можна створювати та керувати адресами для зберігання та транзакцій цифрових активів. Це додаток з інтерфейсом і різними функціями управління адресою і криптоактивами, що зберігаються в ньому.

Під час створення адреси гаманець генерує ключі — криптографічні ідентифікатори, свого роду «посвідчення особи», за допомогою якого ви отримуєте доступ до коштів на своєму рахунку (адресі в блокчейні). Кожна адреса зазвичай має пару ключів — публічну та приватну. Вони пов'язані між собою та прив'язані до конкретної адреси.

Адреса в блокчейні є «стислою» версією публічного ключа, який може подивитися будь-який інший користувач. Приватний ключ використовується для створення цифрових підписів і перевірки транзакцій. Він відомий лише власнику адреси, тому що дає доступ до його коштів.

Сьогодні більшість гаманців підтримують ще один рівень аутентифікації користувачів – за допомогою сід-фрази. Це унікальна послідовність з 12 або 24 слів англійською мовою, яка є паролем для відновлення доступу до адреси або її перенесення в інший гаманець. Сід-фраза, як і приватний ключ — лише для власника адреси, більше їх передавати нікому не можна. Якщо сід-фраза буде втрачена або вкрадена, ви можете втратити доступ до активів.

Криптогаманці поділяються на кастодіальні та некастодіальні.

Кастодіальний гаманець — це додаток для зберігання та транзакцій криптовалют, особливість якого в тому, що його оператор (кастодіан) управляє адресами користувачів або має доступ до приватних ключів. Крім того, клієнти кастодіану мають проходити процедуру верифікації особи (KYC). Вбудований кастодіальний гаманець мають централізовані криптобіржі. Хоча в кожного клієнта окремий рахунок і баланс, усі кошти зберігають на невеликій кількості адресів, якими управляє біржа. Це спрощує роботу з торговими інструментами та дозволяє не платити за транзакції всередині платформи. Прикладом кастодіального гаманця можна вважати гаманці на біржах Binance або FTX.

Основний недолік кастодіальних криптогаманців – можливість для кастодіану отримати доступ до криптоактивів клієнтів. Криптобіржа - юридична організація, яка повинна підпорядковуватися законодавству та вимогам правоохоронних органів. За їх запитом вона може як надавати дані про клієнтів, так і заморожувати їх кошти в гаманці. Наприклад, у разі санкцій чи арешту майна за рішенням суду. Також активи можуть бути викрадені з біржи хакерами. Або ви можете не мати можливості для транзакцій в випадку технічних робіт на біржі.

Некастодіальний криптовалютний гаманець зберігає за творцем адреси повний контроль своїх коштів, оскільки не передає комусь їхні приватні ключі. Така програма не може заморожувати засоби користувачів або керувати ними, але при цьому не несе відповідальності за їх збереження.

Зазвичай це програма, яку можна завантажити на ПК, мобільний пристрій або браузер. Для створення адреси в блокчейні через некастодіальну програму не потрібно проходити KYC.

Некастодіальні гаманці також можна поділити на Гарячі та Холодні.

Гарячий гаманець існує лише як цифрова програма. Їх можна використовувати практично на будь-якій платформі та пристрої. Прикладами гарячих гаманців є Metamask або TrustWallet.

"Холодні", або апаратні гаманці - це пристрої розміром з карту флеш-пам'яті, в яких криптоактиви зберігаються офлайн. Це дає максимальний захист від зламів хакерів. Для здійснення транзакцій "холодний" гаманець потрібно синхронізувати з блокчейном через комп'ютер, підключений до Інтернету.

Прикладом холодних гаманців є Safepal s1 або Ledger.

1.3.6 Навіщо потрібна криптовалюта та її переваги

Перше, коли ви чуєте криптовалюта, вам на думку приходить інвестиції та зберігання активів. Криптовалюту можна використовувати у багатьох сферах, окрім інвестування.

Криптовалюту можна використовувати, як засіб платежу. Уявіть, що потрібно терміново відправити 1000 доларів другу, який живе в іншій країні. Провести такий платіж через банківську систему ціле випробування. Та й до того ж у подібних випадках не кожен може чекати на обробку перекладу від трьох до семи днів. Однак завдяки криптовалютам провести транзакцію вийде ще до завершення телефонної розмови, на початку якої друг попросив вас про допомогу. Це дуже швидко та беспроблемно.

Стояти в черзі для отримання кредиту чи заповнювати необхідні папери досить хворобливий процес. За допомогою криптовалюти можна отримати цифрову позику без необхідності заповнювати будь-яку форму. Платформи із сфери децентралізованих фінансів роблять саме це. Вони позбавляють посередників при кредитуванні та отриманні криптовалютних позик, завдяки чому користувачі можуть позичати гроші у формі криптовалют практично миттєво. Що особливо важливо, для криптовалюту не існує меж, тому подібні угоди з ними можна проводити буквально з будь-якої точки світу, де є інтернет.

Поява криптовалют дозволила проводити токенізацію активів із реального, тобто фізичного світу. При цьому зробити токеном можна будь-що: авторські права, об'єкти нерухомості, витвори мистецтва і навіть звичайні товари. Наприклад, завдяки блокчейну реально токенізувати елітне житло вартістю 10 мільйонів доларів США, перетворивши його на мільйон токенів вартістю 10 доларів кожен. Це забезпечує значно більшу ліквідність для неліквідного активу і полегшує проведення операцій із ним. До того ж продажі та купівлі такого активу стануть більш простими, швидкими та прозорими.

NFT вже серйозно змінюють ситуацію в ігровій індустрії. NFT у світі геймінгу - це криптовалютні токени, які є цифровим активом всередині гри. Оскільки кожен NFT-токен є чимось унікальним, вони мають різну вартість і відсутню властивість взаємозамінності. Відповідно, таким чином, кожен геймер отримує унікальний внутрішньоігровий предмет, який при цьому належить виключно йому.

Є також багато більш нішевих сфер, де використовується криптовалюта.[10]

2. ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

В цій кваліфікаційній роботі буде розроблено месенджер за концепцією Web3. В цьому додатку користувачі зможуть реєструватися та авторизовуватися за допомогою криптогаманця, спілкуватися між собою, відправляти один одному криптовалюту та створювати платний контент.

В цьому розділі я опишу обрані мною технології для розробки цього додатку.

Мій додаток буде розділений на дві частини: клієнтську та серверну.

Серверна частина буде зроблена на основі фреймворку Epress.js, який заснований на платформі Node.js.

Клієнтська частина буде зроблена на основі UI-бібліотеки ReactJS. Усі допоміжні бібліотеки будуть описані у наступному розділі.

Express.js та ReactJS використовують мову JavaScript, як основну, тобто, і клієнтська частина, і серверна частина будуть написані на мові програмування JavaScript.

Реалізацією бази даних буде MongoDB.

Мій додаток буде підтримувати блокчейн Ethereum, а саме основну мережу Mainnet та тестову мережу Goerli.

Криптогаманець, з яким буде взаємодіяти мій месенджер, я обрав Metamask.

Розглянемо ці технології більш детально.

2.1 UI бібліотека ReactJS

React.js був випущений інженером-програмістом, який працює у Facebook, Джорданом Уоці в 2011 році. React - це бібліотека JavaScript, орієнтована на створення декларативних інтерфейсів (UI) з використанням концепції, заснованої на компонентах. Він використовується для обробки шару відображення і може використовуватися для веб-додатків та мобільних додатків.

Основна мета React - бути широким, швидким, декларативним, гнучким і простим.

React – це не фреймворк, а саме бібліотека. Пояснення цьому в тому, що React займається тільки рендерингом інтерфейсу користувача і залишає багато на розсуд окремих проектів. Стандартний набір інструментів для створення програми з ReactJS часто називають стеком.

Об'єктна модель документа (DOM) – це API для дійсних документів HTML та правильно сформованих документів XML. Віртуальний DOM - це представлення реального DOM, яке створюється/керується браузерами. Розширені бібліотеки, такі як React, генерують дерево елементів у пам'яті, еквівалентне реальному DOM, яке формує віртуальний DOM у декларативний спосіб. Віртуальний DOM — одна з функцій, які роблять фреймворк таким швидким та надійним.

React використовує розширення синтаксису JavaScript під назвою JSX. Ми використовуємо її для створення «елементів». JSX використовує препроцесори Babel для перетворення HTML-подібного тексту у файлах JavaScript на об'єкти JavaScript для аналізу. React не вимагає використання JSX, але більшість розробників вважають, що це робить JavaScript більш зручним для користувача. Ми використовуємо JSX для створення компонентів React, тому він є важливою частиною ReactJS.

Таким чином, головне питання полягає в тому, чому ви повинні вибрати ReactJS як стека розробки інтерфейсу, в той час як є багато інших. Ось кілька причин:

- Швидкість. React дозволяє розробникам використовувати окремі частини своєї програми як на стороні клієнта, так і на стороні сервера, і будь-які внесені ними зміни не вплинуть на логіку програми. Це робить процес розробки надзвичайно швидким.
- Підтримка компонентів. Використання тегів HTML і кодів JS полегшує роботу з великим набором даних, що містять DOM. React діє як

посередник, який представляє DOM і допомагає вам вирішити, який компонент потребує змін для отримання точних результатів.

- Легко використовувати та вчитися. ReactJS неймовірно зручний для користувача і робить будь-який інтерфейс користувача інтерактивним. Це також дозволяє швидко та ефективно створювати програми, що економить час як клієнтів, так і розробників.
- SEO. Поширена проблема, на яку скаржиться більшість веб-розробників, полягає в тому, що традиційні фреймворки JavaScript часто мають проблеми із SEO. ReactJS вирішує цю проблему, допомагаючи розробникам легко орієнтуватися в різних пошукових системах завдяки тому факту, що ReactJS може працювати на сервері, а віртуальний DOM відображає і повертає його в браузер як веб-сторінку.[11]

2.2 Фреймворк Express.js

Express.js — це веб-фреймворк NodeJS, який використовується на серверній частині веб-сайтів та веб-додатків. Express є гнучким та мінімалістичним, а це означає, що він не має великої колекції непотрібних бібліотек та пакетів і не диктує, як вам слід створювати свою програму.

Платформа Express створює API-інтерфейси, які полегшують обмін даними через HTTP-запити та відповіді. Одна із чудових особливостей Express полягає в тому, що він дає розробникам повний контроль над запитами та відповідями, пов'язаними з кожним із методів його застосування.

Express.js - найпопулярніше серверне середовище для Node.js і велика частина екосистеми JavaScript.

Платформа надає набір інструментів для веб-додатків, HTTP-запитів та відповідей, маршрутизації та проміжного програмного забезпечення для створення та розгортання великомасштабних корпоративних додатків.

Він також надає інструмент інтерфейсу командного рядка (CLI) під назвою Node Package Manager (NPM), де розробники можуть одержувати розроблені

пакети. Це також змушує розробників дотримуватися принципу "Не повторюйся" (DRY).

Принцип DRY спрямований на зменшення повторення шаблонів програмного забезпечення, заміну його абстракціями або використання нормалізації даних, щоб уникнути надмірності.

Переваги Express.js:

- Гнучкий та швидкий. Express.js дуже простий у використанні і гнучкий, і він швидше, ніж будь-який інший фреймворк Node.js. Мінімалістичний фреймворк пропонує швидку розробку додатків та полегшує освоєння безлічі різних частин більшого фреймворку. Він також надає багато можливостей, такі як відмінна система маршрутизації, проміжне програмне забезпечення та узгодження контенту прямо з коробки.
- Масштабованість. За минулі роки Express.js зарекомендував себе як дуже масштабований через велику кількість компаній, які щодня використовують фреймворк на своїх серверах. Він ефективно обробляє запити та відповіді користувачів і практично не вимагає додаткового налаштування при розробці великомасштабного веб-додатку. Він має відмінні модулі, пакети та додаткові ресурси, які допомагають розробникам створювати надійні та масштабовані веб-програми.
- Підтримується двигуном Google V8. Express.js підтримує безліч пакетів двигуна Google V8, що робить платформу дуже потужною для створення та розгортання додатків реального часу, спільних та мережевих додатків на рівні підприємства. Двигун Google V8 – це високопродуктивний двигун JavaScript та WebAssembly з відкритим вихідним кодом, який підтримує високу швидкість та масштабованість для складних та ресурсомістких додатків. Коли ви використовуєте пакети, що використовують двигун Google V8, це значно підвищує продуктивність та масштабованість вашої серверної програми.

Потужна система маршрутизації. Платформа має найпотужнішу та найнадійнішу систему маршрутизації, вбудовану з коробки, яка допомагає

вашому додатку у відповіді на запит клієнта через певну кінцеву точку. За допомогою системи маршрутизації в Express.js можна розділити роздуту систему маршрутизації на керовані файли, використовуючи екземпляр маршрутизатора фреймворку. Система експрес-маршрутизації допомагає керувати структурою вашої програми, групуючи різні маршрути в одну директорію/каталог. Розробники створюють зручніші у супроводі коди, групуючи функції з маршрутизатором Express та уникаючи повторень.[12]

2.3 Нереляційна СУБД MongoDB

MongoDB – є документно-орієнтованою NoSQL базою даних яку використовують при зберіганні значної кількості даних. На відміну від реляційний баз даних які використовують таблиці та рядки, у MongoDB використані документи разом з колекціями. В MongoDB використовують документи які є парою ключ-значення я основну одиницю даних. Колекції містять набори документів та функцій, які еквівалентні таблицям реляційних баз даних. MongoDB - це база даних, що з'явилася приблизно в середині 2000-х років.

Переваги MongoDB:

- Орієнтованість на документи. Оскільки MongoDB є базою даних типу NoSQL, замість того, щоб мати дані у форматі реляційного типу, вона зберігає дані у документах. Це робить MongoDB дуже гнучкою та адаптованою до реальної ситуації та вимог ділового світу.
- MongoDB підтримує пошук за регулярними виразами та за полями. Запити можуть бути зроблені для повернення певних полів у документах.
- Індексування. Можна створювати індекси для підвищення продуктивності пошуку MongoDB. Будь-яке поле в документі MongoDB може бути проіндексовано.
- MongoDB забезпечує високий рівень доступу за допомогою копій наборів. Кожна частина набору копій може бути у ролі основної чи запасної копії у час. Основна копія – це основний сервер, який взаємодіє з клієнтом та виконує

всі операції читання/запису. Повторні копії підтримують копію даних основної копії за допомогою вбудованої реплікації. При збої основної копії набір копії автоматично перемикається на додаткову, потім стає основним сервером.

- Балансування навантаження - MongoDB використовує концепцію сегментування для горизонтального масштабування шляхом поділу даних між кількома екземплярами MongoDB. MongoDB має можливість балансувати навантаження та дублювати дані, працюючи на двох серверах одночасно, щоб система не впала у разі збоїв.[13]

2.4 Блокчейн Ethereum

Ethereum — це децентралізована глобальна програмна платформа, яка базується на технології блокчейн. Свою популярність він отримав завдяки цифровій валюті ефір (ETH). Будь-хто може використовувати блокчейн Ethereum для створення безпечної цифрової технології. Монета ETH призначена для оплати роботи, яка підтримує блокчейн, але учасники також можуть використовувати його для оплати матеріальних товарів і послуг.[14]

Ethereum має бути масштабованим, програмованим, безпечним і децентралізованим. Це блокчейн, який розробники та компанії обожнюють використовувати для створення технологій, які змінюють спосіб роботи багатьох галузей промисловості та наше повсякденне життя. Він повністю сумісний зі смарт-контрактами, це важливо для децентралізованих програм(Dapp).

Багато децентралізованих фінансових проектів використовують смарт-контракти в поєднанні з технологією блокчейн. Також Ethereum має тестові мережі, які добре розвинені та повністю функціональні. Крім того, у цих тестових мережах легко отримати криптовалюту для тестування вашої програми.[15]

2.5 Криптогаманець Metamask

Нещодавній крах найбільших криптовалютних бірж подарував нову хвилю популярності некастодіальним рішенням (гаманцям), які є воротами у світ web3. Not your key – not your money. Саме гаманці дозволяють повноцінно взаємодіяти із DAPP. Найбільш популярними криптогаманцями зараз є Metamask. SafePal, C98, Coinbase Wallet, Phantom. У кожного є свої особливості та обмеження. Phantom розроблений спеціально для блокчейну Solana. SafePal - мультивалютний гаманець (підтримує більшість мереж). Але мій вибір при розробці впав на Metamask. Тільки він вміє працювати із тестовими мережами.

При першому запуску програма запропонує створити або імпортувати ваш криптовалютний гаманець (за допомогою seed-фрази або privat key).

Важливе уточнення – Metamask підтримує виключно мережу EVM.

При великому завантаженні блокчейна іноді виникає помилка надсилання транзакцій, для цього в гаманцях часто можна вибрати ноду. Великі проекти можуть створювати свої приватні ноди для більшої незалежності та стабільної роботи.

Переваги MetaMask:

- Можна додавати тестові мережі, що найбільш потрібні для розробки
- Зручний інтерфейс
- Доступний вибір Gas + Gwei
- Підтримує всі популярні браузери

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Моделювання додатку

В цьому розділі будуть розглянуті основні моменти моделювання додатку.

3.1.1 Моделювання бази даних

Перше, що потрібно зробити, моделюючи базу даних для веб-месенджеру, це подумати про усі сутності, які будуть у цьому додатку.

Першою сутністю мого додатку є користувач. У моєму месенджері авторизація та реєстрація буде зроблена за допомогою криптогаманця, але у користувачів також будуть email та username для відображення певної інформації. Отже сутність користувача повинна мати такі поля: унікальний ідентифікатор криптогаманця, email, username.

Наступною сутністю є кімната месенджеру, в яку будуть мати доступ певні користувачі, яких запросив власник цієї кімнати. Отже сутність кімнати повинна мати такі поля: унікальний ідентифікатор, список запрошених користувачів.

Останньою сутністю буде сутність повідомлення. Повідомлення можуть бути двох видів: звичайні та платні. У звичайних повідомлень є поля: відправник(людина, яка відправила повідомлення), контент(тобто текст) та дата відправки. У платного повідомлення більше полів. Необхідним полем платного повідомлення є сума, за яку можливо розблокувати повідомлення. Також потрібно поле криптогаманець відправника, куди будуть приходити кошти за розблокування повідомлення.

Рисунок 3.1 демонструє, як повністю виглядатиме структура бази даних.

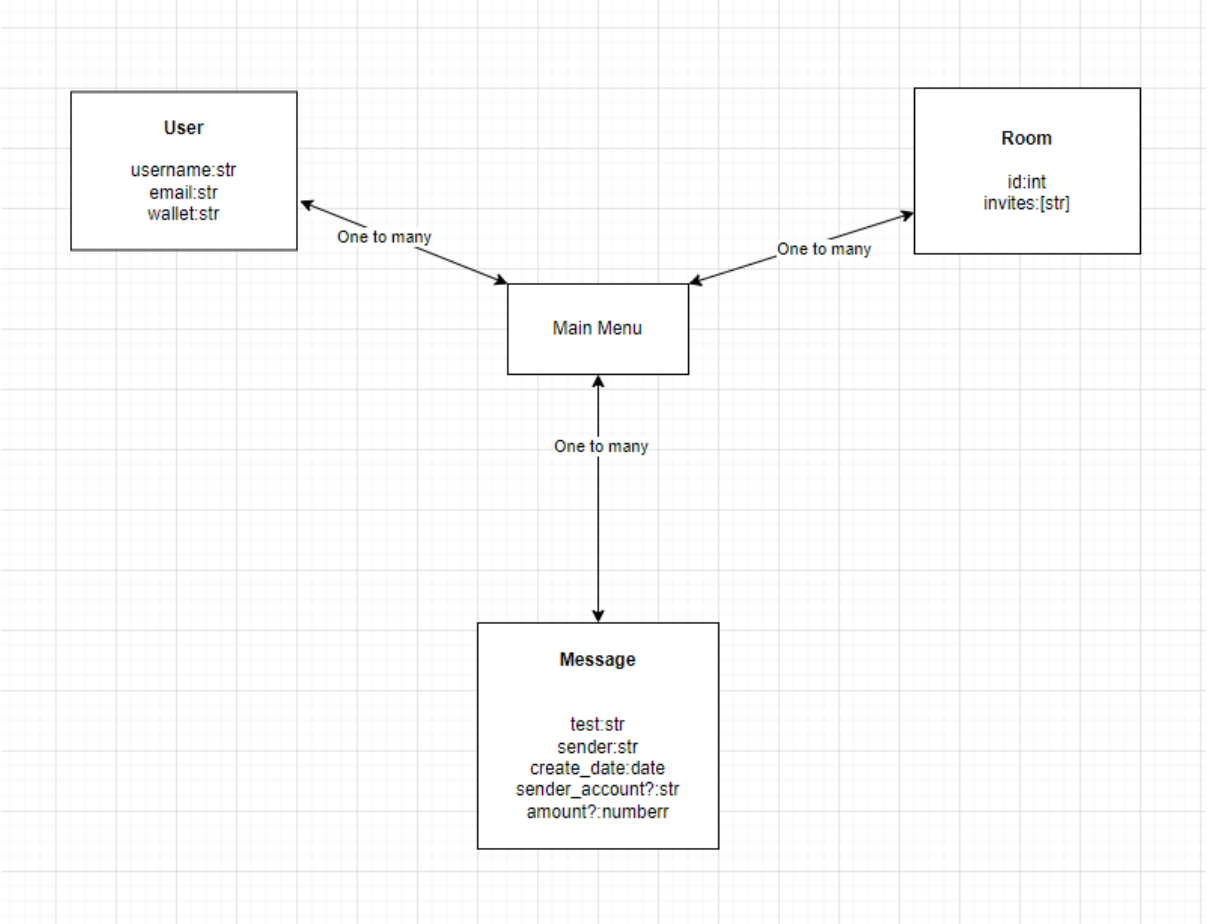

Рисунок 3.1 – Модель бази даних

3.1.2 Моделювання подій

В даному розділі будуть продемонстровані усі події, які будуть реалізовані в месенджері.

1. Першою подією буде реєстрація користувача. При першому відвідуванні месенджеру користувачу потрібно буде зареєструватися для подальшої можливості авторизації. Реєстрація та авторизація в моєму месенджері буде відбуватися за допомогою криптогаманця, але при реєстрації також користувач повинен буде вказати свій email та username. Модель цієї події зображена на рисунку 3.2.

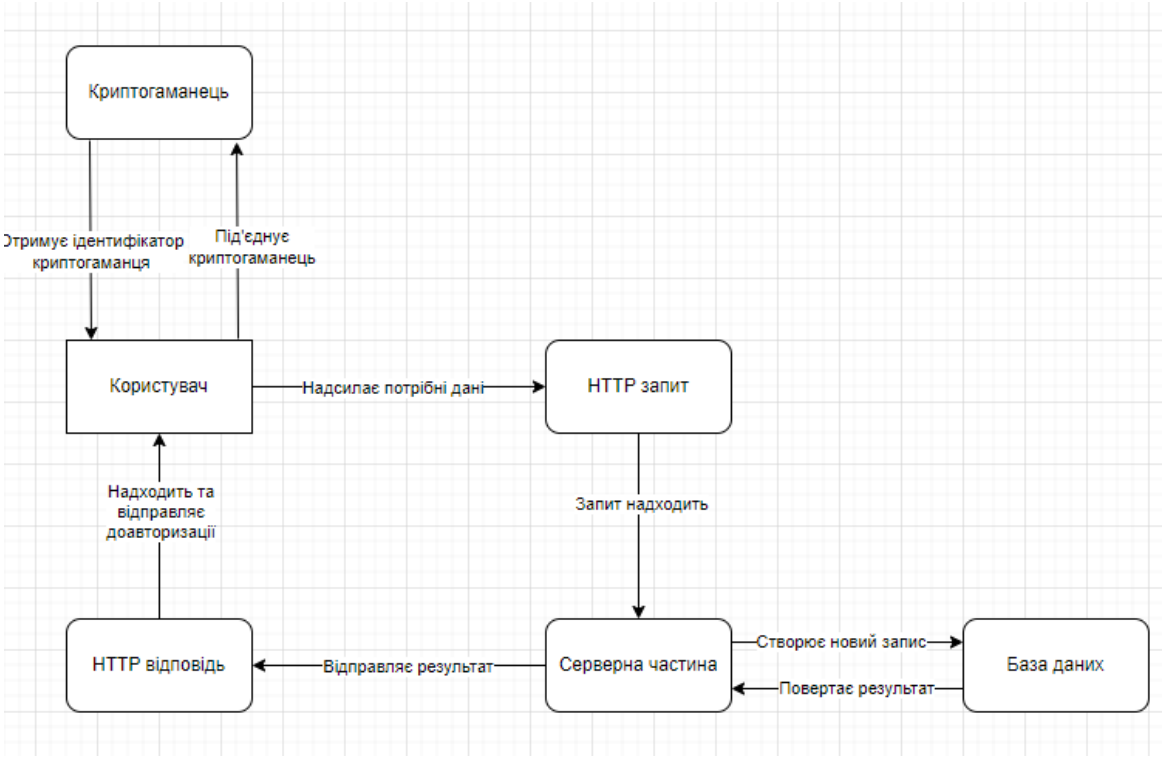

Рисунок 3.2 - Модель події реєстрації

2. Авторизація. В моєму додатку авторизація виконується за допомогою криптогаманця. Тут буде достатньо лише запиту до нього. Модель цієї події зображена на рисунку 3.3.

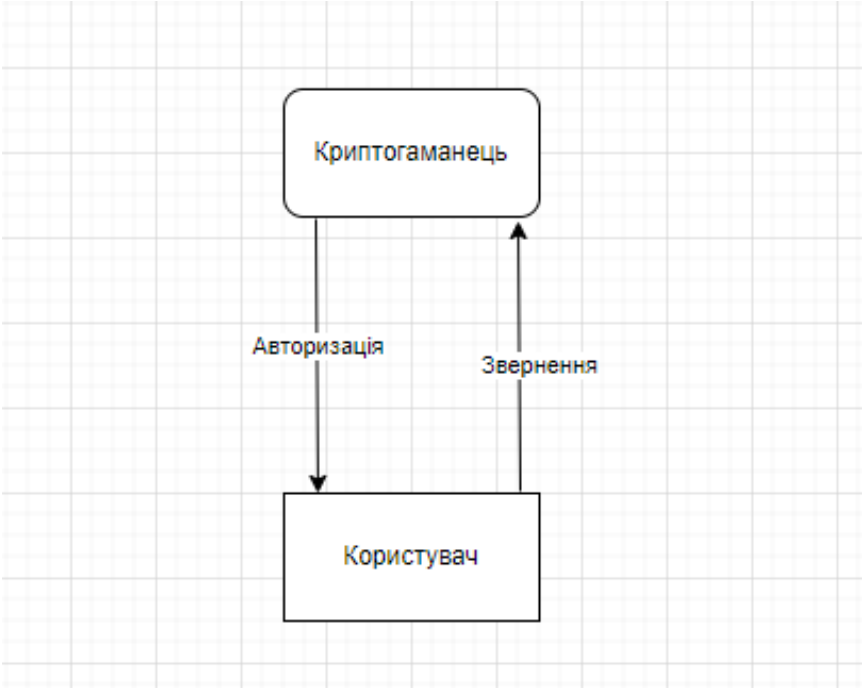

Рисунок 3.3 – Модель події авторизації

3. Наступною подією буде створення кімнати месенджеру. Створити кімнату може користувач, який пройшов етапи реєстрації та авторизації. Для цього потрібно зробити HTTP запит до серверної частини, відправивши список користувачів, яких хочеш запросити до кімнати. Модель цієї події зображена на рисунку 3.4.

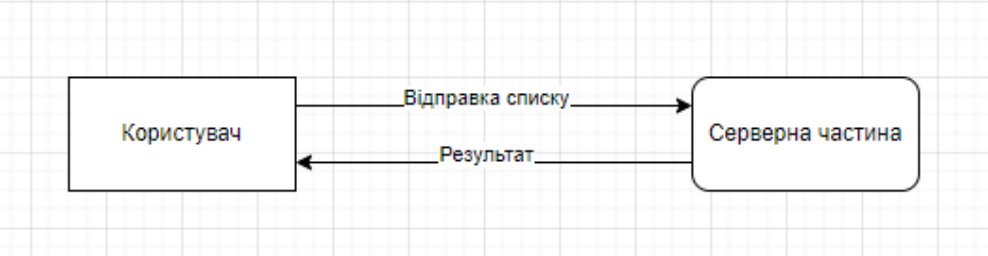

Рисунок 3.4 – Модель події створення кімнати

4. Підключення до кімнати месенджеру. В моєму месенджері використовується Websocket API для двостороннього з’єднання. При під’єднанні до кімнати, користувач повинен на клієнтській частині створити підключення та додати це підключення на серверній частині. При цьому процесі користувач змінює у себе на сторінці протокол обробки даних у мережі на Websocket. Модель цієї події зображена на рисунку 3.5.

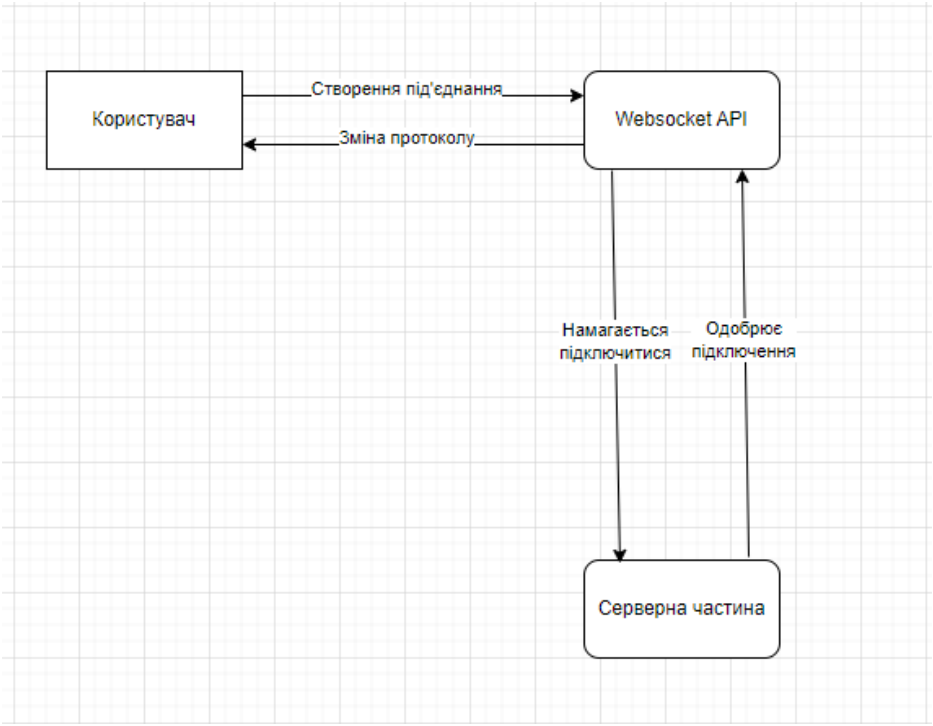

Рисунок 3.5 – Модель події під’єднання до кімнати

5. Подія відключення від кімнати месенджеру. Ця подія відбувається в зворотному порядку відносно підключення до кімнати. При виході з кімнати протокол змінюється назад на HTTP.

6. Подія надсилання повідомлення. Користувач надсилає повідомлення у кімнаті, тобто усі користувачі, які знаходяться в кімнаті повинні побачити це повідомлення. Модель цієї події зображена на рисунку 3.6.

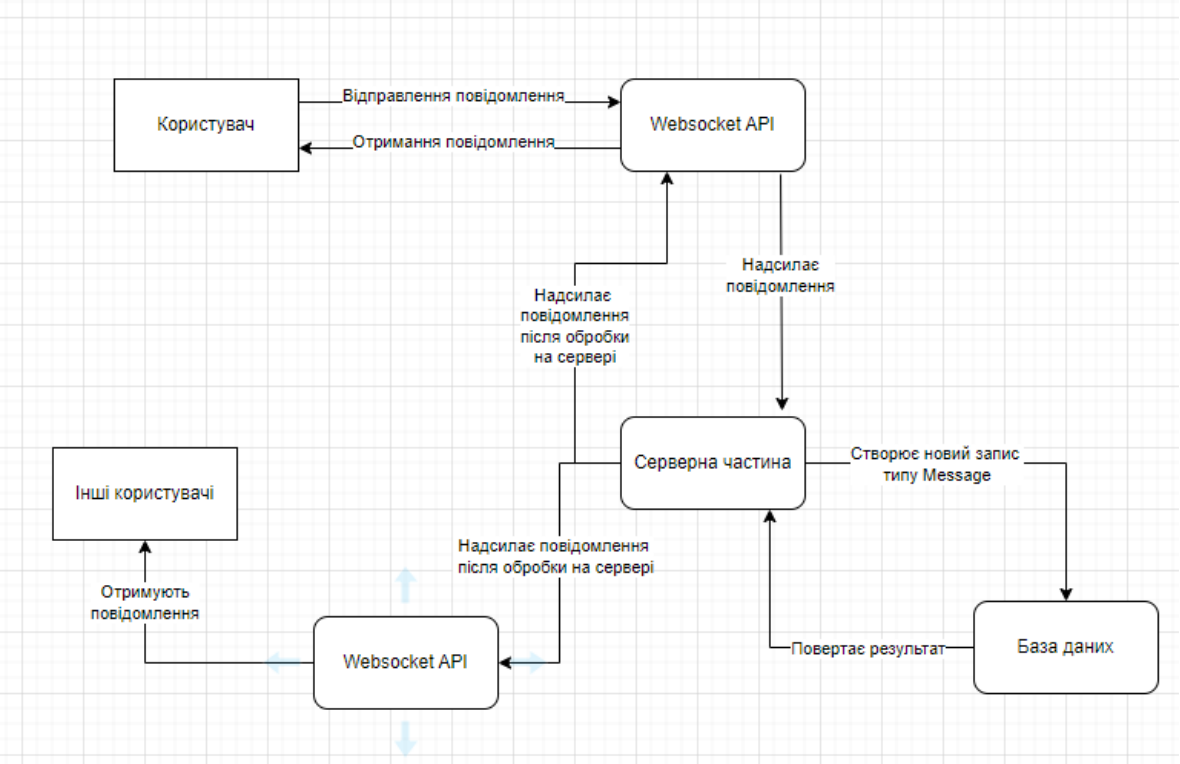

Рисунок 3.6 – Модель події надсилання повідомлення

7. Надсилання криптовалюти. Користувач обирає кількість криптовалюту та користувача зі списку усіх користувачів у кімнаті, після цього підтверджує транзакцію в криптогаманці. Модель цієї події зображена на рисунку 3.7.

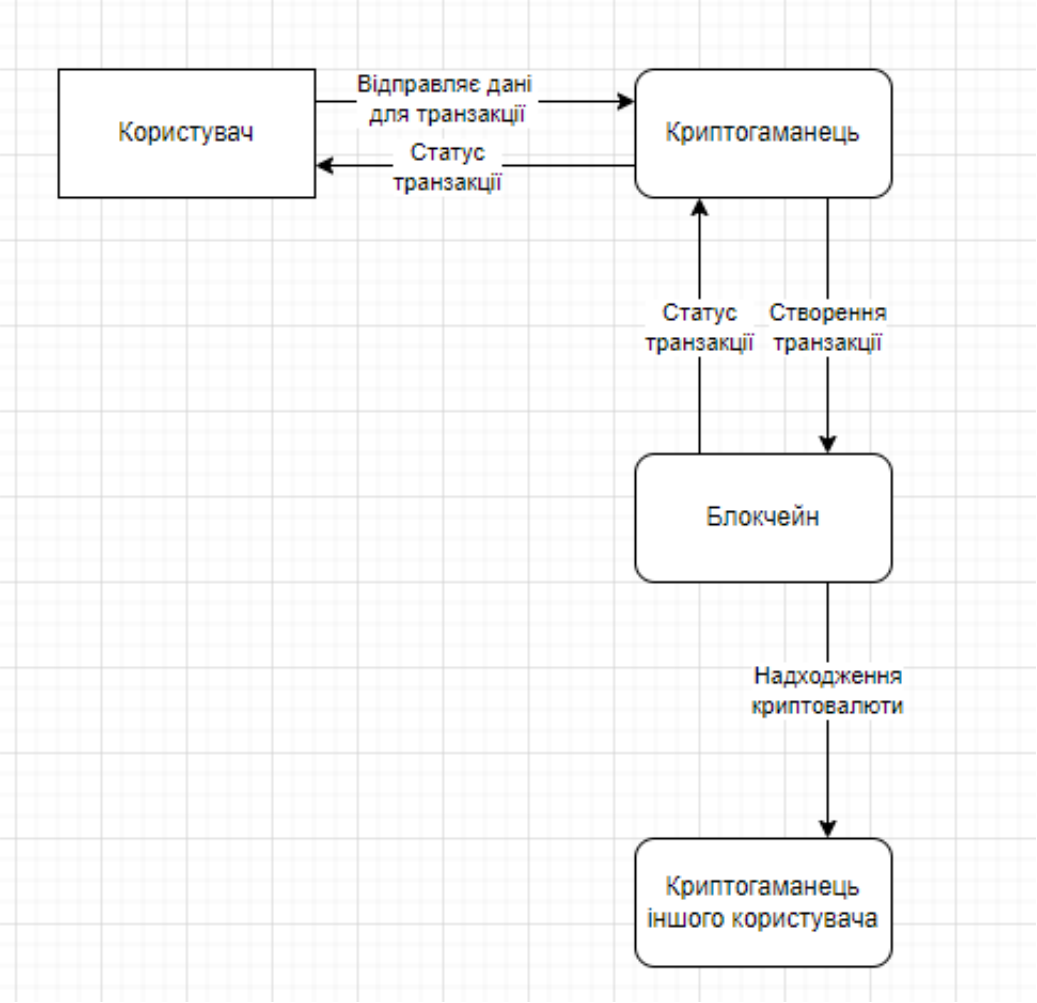

Рисунок 3.7 – Модель події надсилання криптовалюти

3.1.3 Моделювання інтерфейсу додатку

В цьому розділі змоделюємо попередній дизайн інтерфейсу мого месенджеру.

1. Сторінка реєстрації. На цій сторінці повинні бути поле для введення email, поле для введення username, кнопка для під’єднання криптогаманця та кнопка для підтвердження реєстрації. Wireframe сторінки реєстрації зображений на рисунку 3.8.

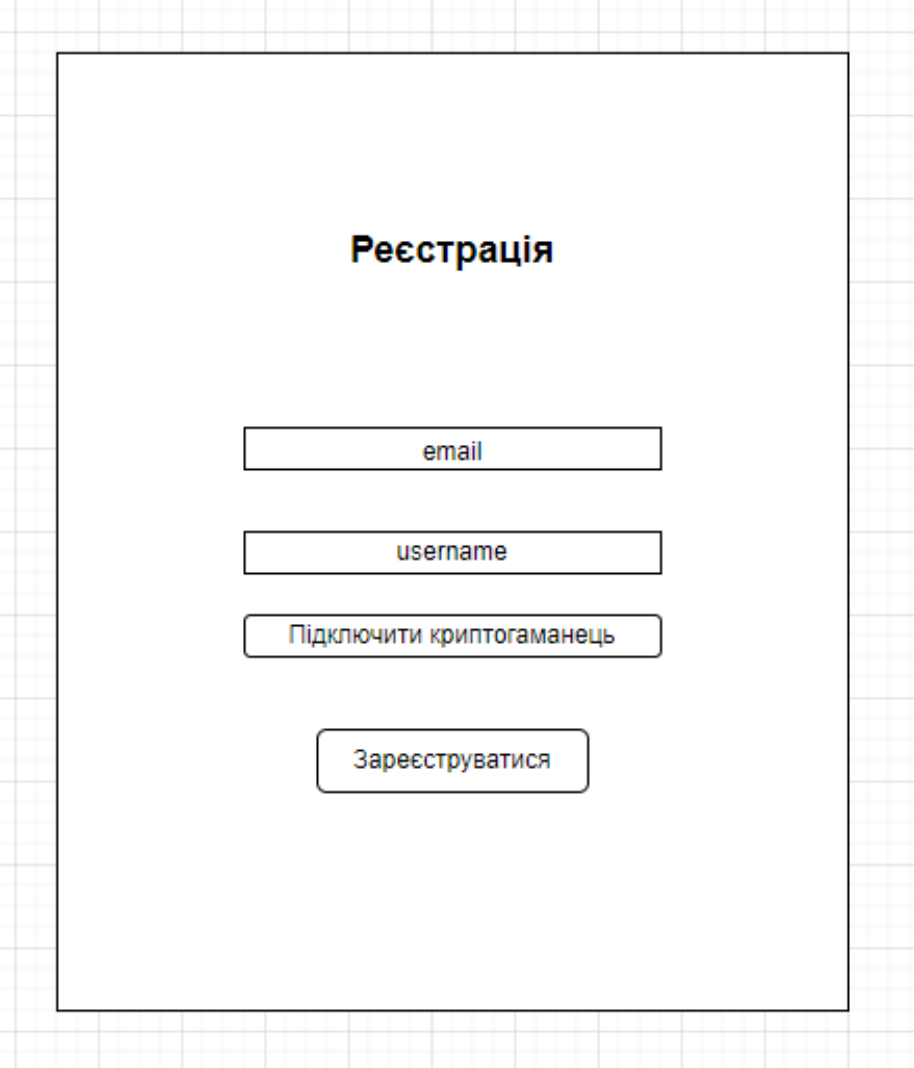

Рисунок 3.8 – Wireframe сторінки з реєстрацією

2. Сторінка авторизації. Так як в моєму додатку авторизація буде реалізована за допомогою криптогаманців, то на цій сторінці буде лише одна кнопка авторизації, яка буде викликати криптогаманець. Wireframe даної сторінки зображений на рисунку 3.9.

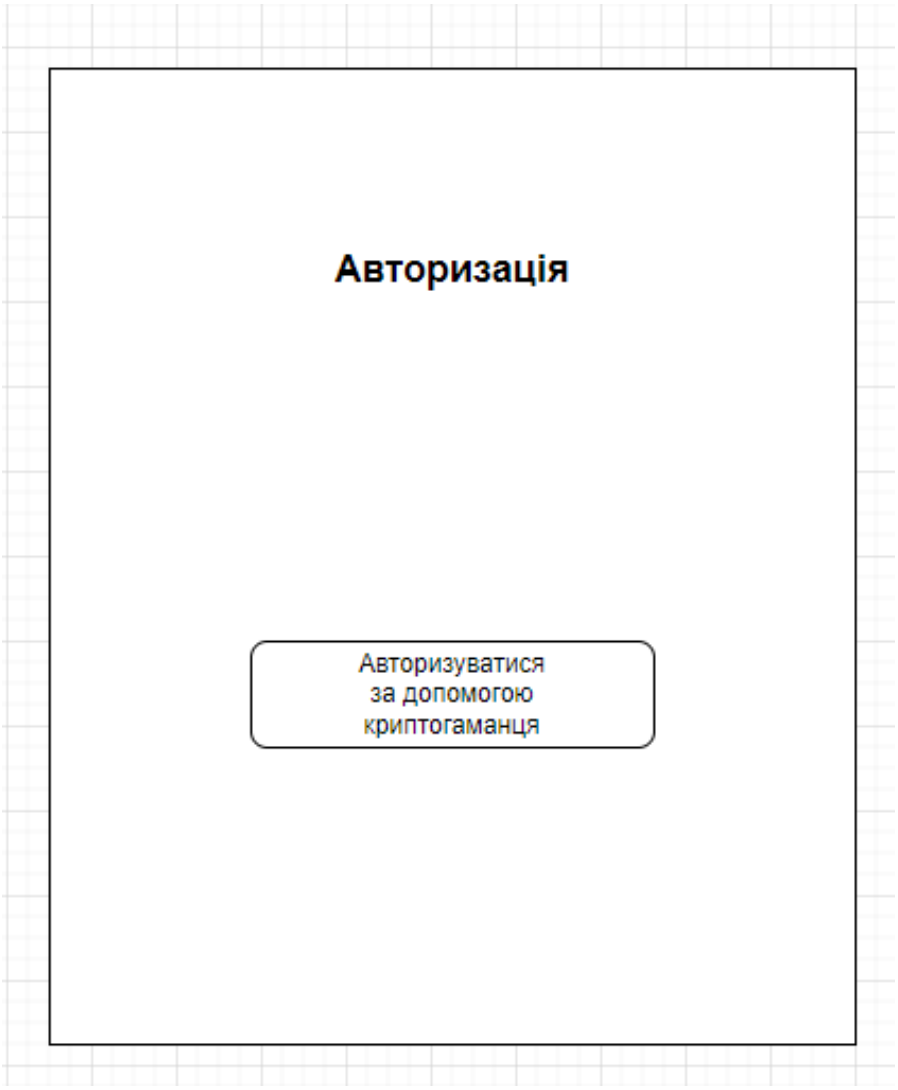

Рисунок 3.9 – Wireframe сторінки з авторизацією

3. Головна сторінка додатку. На цій сторінці є дві функції: створення нової кімнати та можливість зайти в вже існуючу кімнату. При натисканні кнопки створення кімнати відкривається модальне вікно, куди вносяться email людей, яких ти хочешь запросити до кімнати. Wireframe даної сторінки зображений на рисунку 3.10. Wireframe модального вікна зображений на рисунку 3.11.

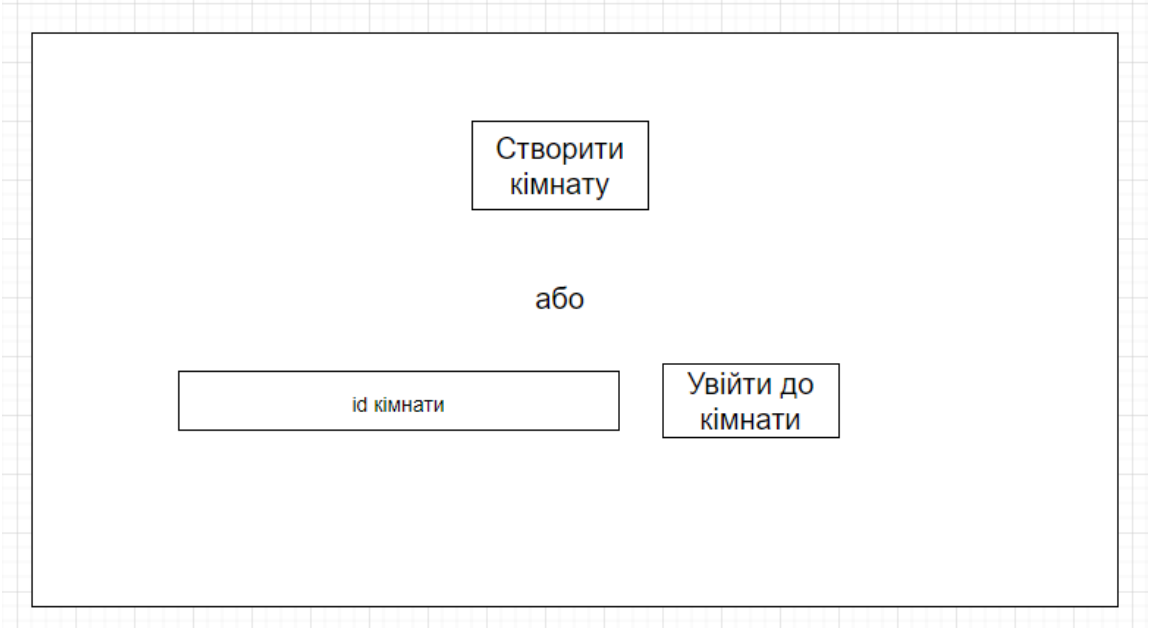

Рисунок 3.10 – Wireframe головної сторінки

Рисунок 3.11 – Wireframe модального вікна створення кімнати

4. Сторінка кімнати месенджеру. На даній сторінці користувач зможе відправляти повідомлення іншим користувачам, дивитися повідомлення інших користувачів, відправляти криптовалюту, створювати платний контент. Wireframe даної сторінки зображений на рисунку 3.12. Wireframe модального вікна відправлення криптовалюти зображений на рисунку 3.13. Wireframe модального вікна створення платного контенту зображений на рисунку 3.14.

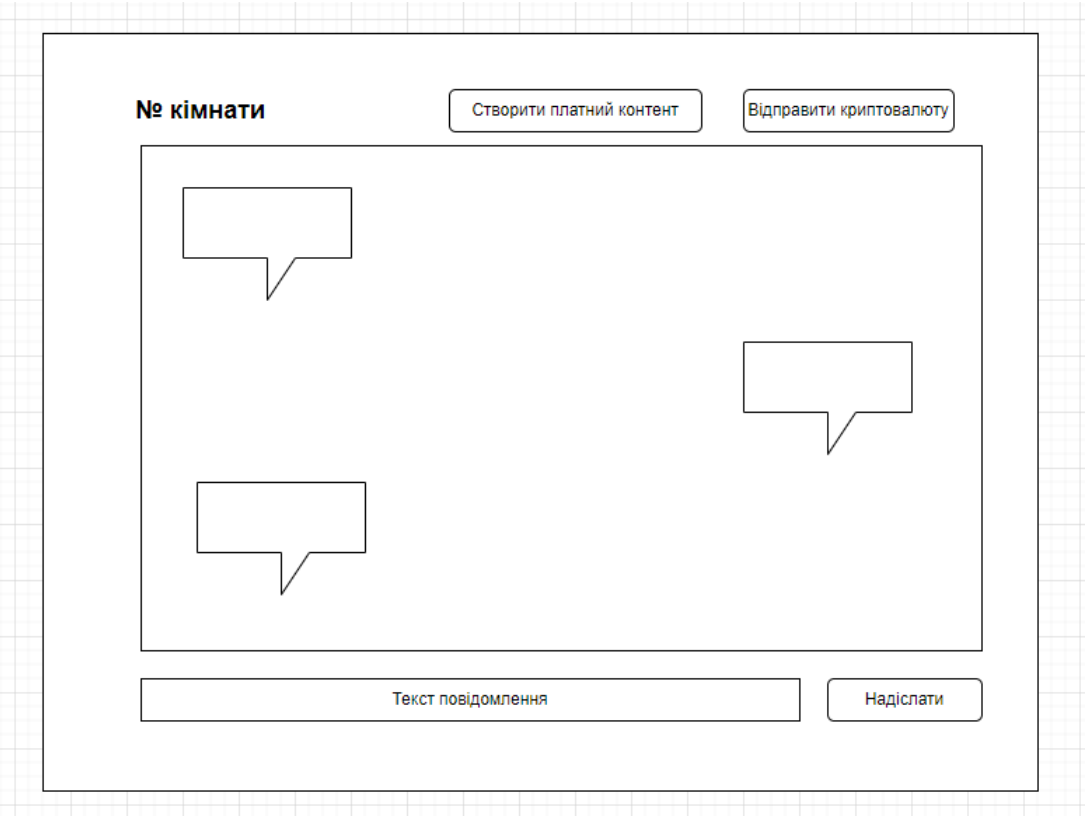

Рисунок 3.12 – Wireframe сторінки з кімнатою

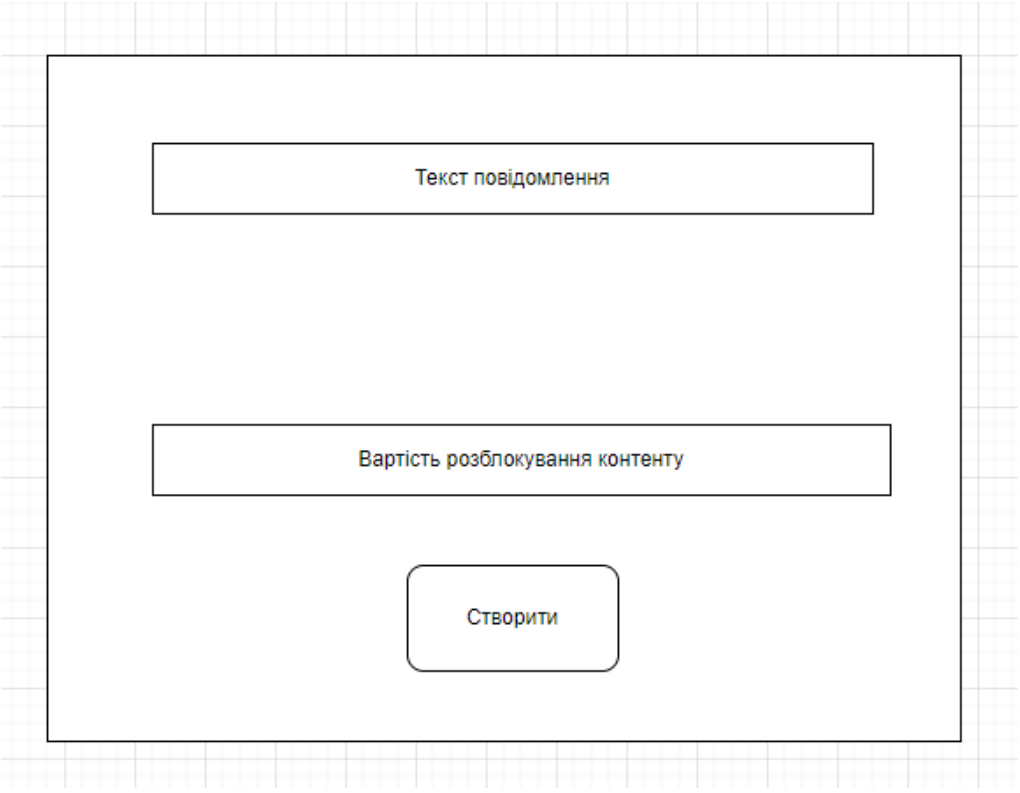

Рисунок 3.13 – Wireframe модального вікна створення платного контенту

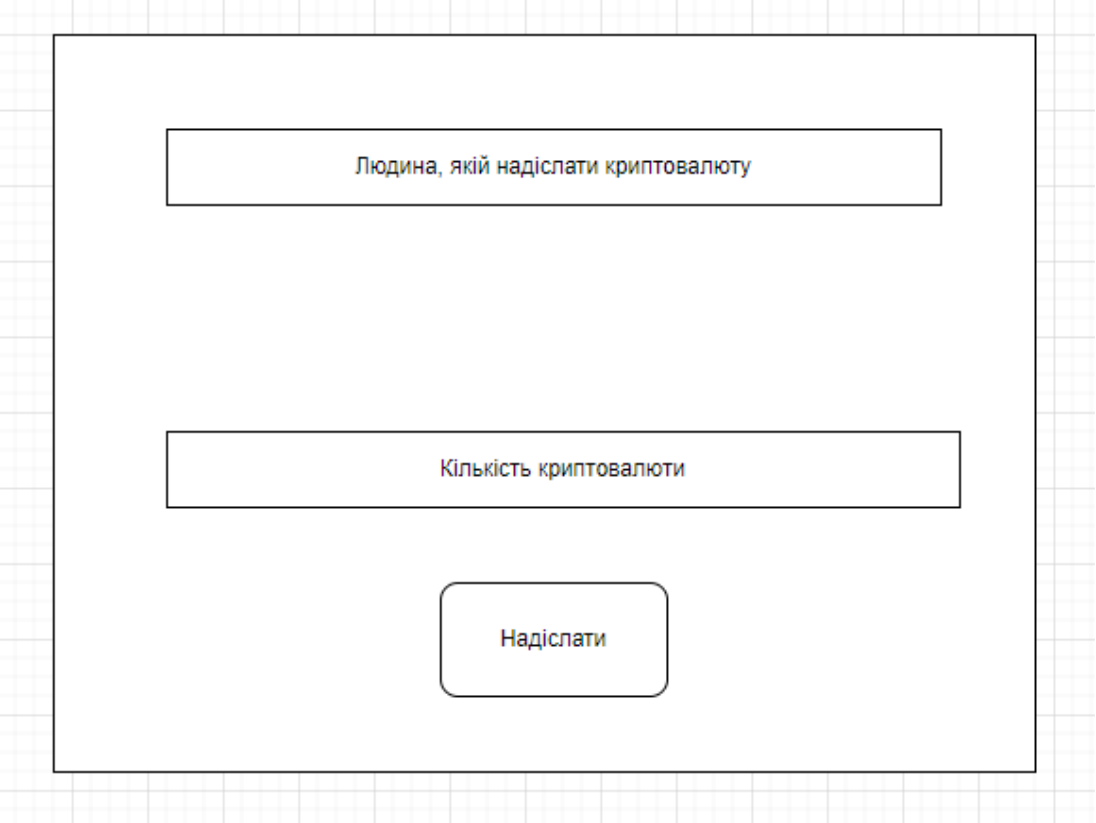

Рисунок 3.14 – Wireframe модального вікна відправки криптовалюти

3.2 Практична реалізація

При розробці додатку, реалізація була поділена на дві частини. Були розроблені клієнтська частина та серверна частина.

В даному розділі будуть розглянуті конфігурації обох частин та будуть продемонстровані фрагменти коду.

3.2.1 Структура серверної частини

Я використовував npm для встановлення пакетів. npm – звичайний менеджер пакетів, вбудований в Node.js.

Список пакетів, які я використовував для розробки серверної частини:

- express – головний фреймворк для розробки серверної частини в моєму додатку
- moment – пакет для роботи з часом та датами
- mongoose – пакет для роботи з MongoDB за допомогою express

- helmet – пакет для захисту веб-додатку. Цей пакет автоматично до всіх HTTP запитів додає спеціальні headers, які роблять ці запити більш захищеними
- body-parser – пакет, який оброблює усі HTTP запити та перетворює body запиту у валідний JavaScript об’єкт
- nodemon – пакет, який використовується для більш швидкої розробки додатку. Він дозволяє запускати серверну частину у режимі відслідковування, за допомогою цього режиму не потрібно після кожної зміни збирати додаток.

Дерево файлів серверної частини складається з файлів коду самого додатку, конфігурацій для продакшен та локальної версій додатку та конфігурацій форматування файлів. Дерево файлів можна побачити на рисунку.

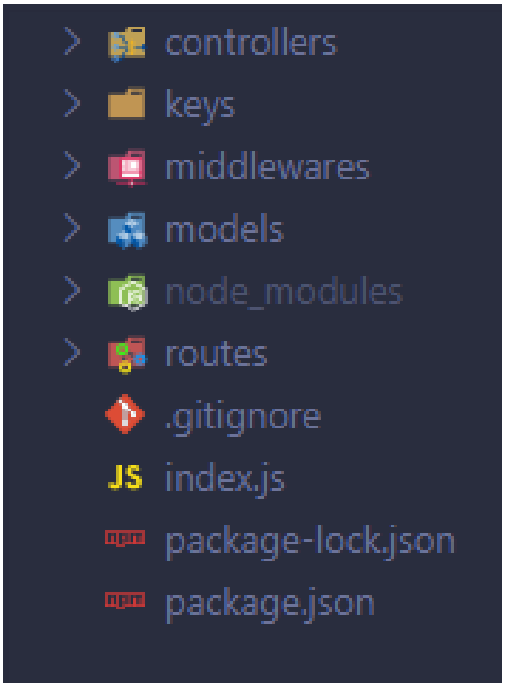

Рисунок 3.15 – Дерево файлів серверної частини

Приклад коду серверної частини, який демонструє ендпоінт для оплати контенту:

```
exports.payMessageFunc = (requestData, resultData) => {
  try {
    jsonwt.verify(
      requestData.headers[tokenHeader],
```

```
            (errorTracked, decodedData) => {
                if (errorTracked) {
                    return resultData.status(400).send("U are unauthorized!");
                }
                MessageObject.findByIdAndUpdate(
                    {
                        _id: requestData.params.id,
                    },
                    { paid: true },
                    function (errorDara, resultData) {
                        if (errorTracked) {
                            resultData.send(errorDara);
                        } else {
                            resultData.send(resultData);
                        }
                    }
                );
            }
        );
    } catch (errorSystem) {
        console.log(errorSystem);
    }
};
applicationBack.post("/api/pay_message/:id", controller.payMessageFunc);
```

3.2.2 Структура клієнтської частини

При розробці клієнтської частини я також використовував npm як пакетний менеджер для встановлення сторонніх бібліотек.

Для побудови клієнтської частини я використовував таку утіліту, як create-react-app, яка створює вже налаштовану конфігурацію для розробки react додатку.

Список пакетів, які я використовував для розробки клієнтської частини:

- react – основна бібліотека, яка використовувалася для розробки додатку
- redux – бібліотека, яка допомагає працювати з даними та робити нормальну структуру даних при роботі з реактом

- mui – пакет, який має в собі вже готові компоненти react, які пришвидшують розробку
- usedapp/core – пакет для роботи з криптогаманцями, криптовалютою та саме блокчейном ethereum. Ця залежність допомагає приєднувати криптогаманці, створювати транзакції, відслідковувати статуси транзакцій
- axios – бібліотека для роботи з HTTP запитами. Ця залежність полегшує взаємодію з серверною частиною
- buffer – залежність, яка допомагає копіювати дані в буфер обміну
- ethers – пакет, який має різні готові функції для взаємодії з блокчейном Ethereum
- moment - пакет для роботи з часом та датами
- react-toastify – бібліотека, яка має вже готові компоненти для сповіщень
- reconnecting-websocket – пакет для взаємодії з Websocket API

Дерево файлів для клієнтської частини можна побачити на рисунку:

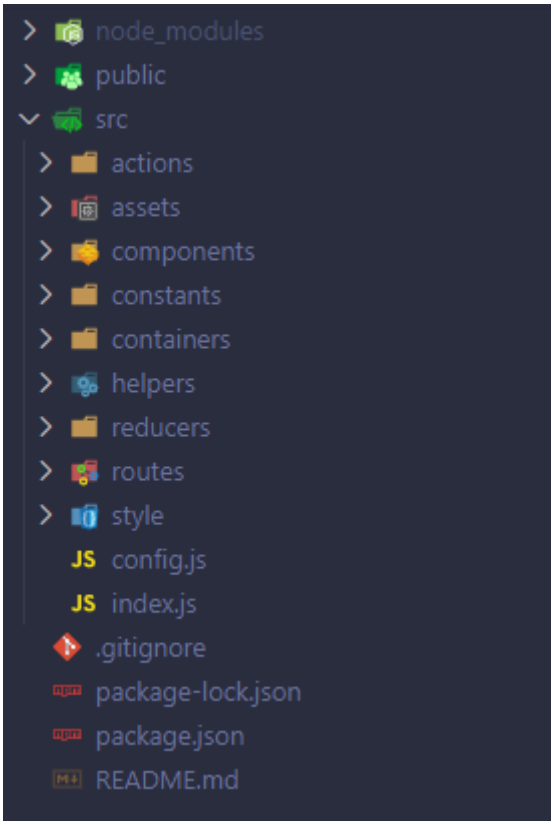

Рисунок 3.16 – Дерево файлів клієнтської частини

Дерево файлів складається з таких директорій:

- actions – директорія з файлами, які містять дії для redux
- assets – директорія, яка містить усі статичні файли, тобто картинки, іконки, шрифти
- components – основна директорія, яка містить усі компоненти, з яких складається клієнтська частина
- constants – директорія, де зберігаються усі константи
- containers – основні контейнери, які збирають компоненти до купи
- helpers – директорія, яка містить допоміжні кастомні функції, які використовуються в додатку
- reducers – спеціальні файли для бібліотеки redux
- style – директорія з CSS файлами, які задають стилі усім компонентам в додатку

Приклад коду клієнтської частини, який демонструє компонент модального вікна для відправки криптовалюти іншому користувачу:

```
useEffect(() => {
  if (stated.statuses === "Success") {
    setOpenDialog(false);
    setCryptoAmount("");
    setWallet("");
    setCryptoLoading(false);
    toast("Криптовалюту відправлено успішно!");
  }
}, [stated]);

<DialogComp
        isOpenNow={openDialog}
        onCloseDialog={() => {
          setOpenDialog(false);
          setCryptoAmount("");
          setWallet("");
          setCryptoLoading(false);
        }}
```

```
>
  <div className="container-dialog">
    <h2>Відправлення криптовалюти</h2>
    <p>
      Оберіть людину, якій ви хочете відправити криптовалюту
    </p>
    <FormControl className="forn-control"  fullWidth>
      <InputLabel>
        Email людини
      </InputLabel>
      <Select
        value={wallet}
        label="Wallet"
        onChange={handleChangeWallet}
      >
        {wallets?.length > 0 &&
          wallets?.map((el) => {
            if (el.email != email.email)
              return (
                <MenuItem value={el.account}>
                  {el.email}
                </MenuItem>
              );
          })}
      </Select>
    </FormControl>
    <p className="enter_crypto" style={{ marginTop: "30px" }}>
      Введіть кількість криптовалюти, яку хочете надіслати
    </p>
    <input
      style={{ marginBottom: "50px" }}
      placeholder="Amount"
      value={cryptoAmount}
      onChange={(e) => setCryptoAmount(e.target.value)}
    />
    <ButtonComp
      onClick={() => {
        sendTransaction({
          to: wallet,
          value: parseEther(cryptoAmount),
        });
```

```
            setCryptoLoading(true);
          }}
          loading={cryptoLoading}
          disabled={cryptoLoading}
          classes="full_wdith_btn"
        >
          Відправити
        </ ButtonComp >
      </div>
    </ DialogComp >
```

3.3 Результат розробки додатку

У результаті ми маємо додаток, який має весь функціонал, що було заплановано на стадії моделювання та планування.

Ось такі сторінки та компоненти ми маємо у результуючому додатку:

- Сторінка реєстрації показана на рисунку 3.17

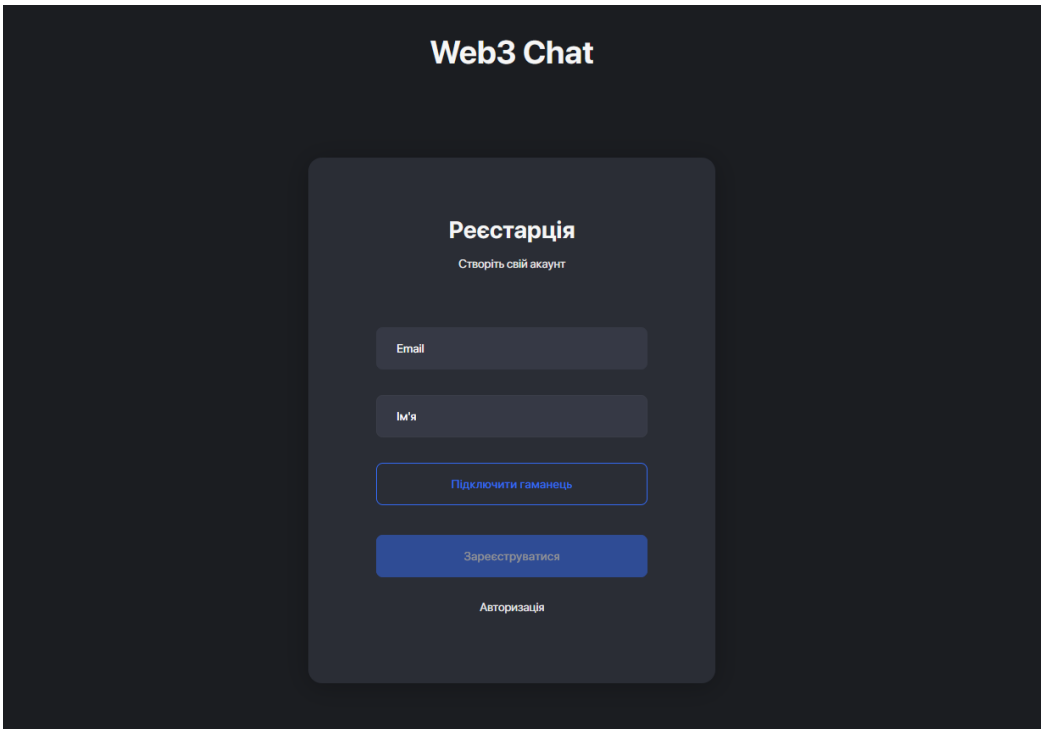

Рисунок 3.17 – Сторінка з реєстрацією месенджеру

- Сторінка авторизації показана на рисунку 3.18

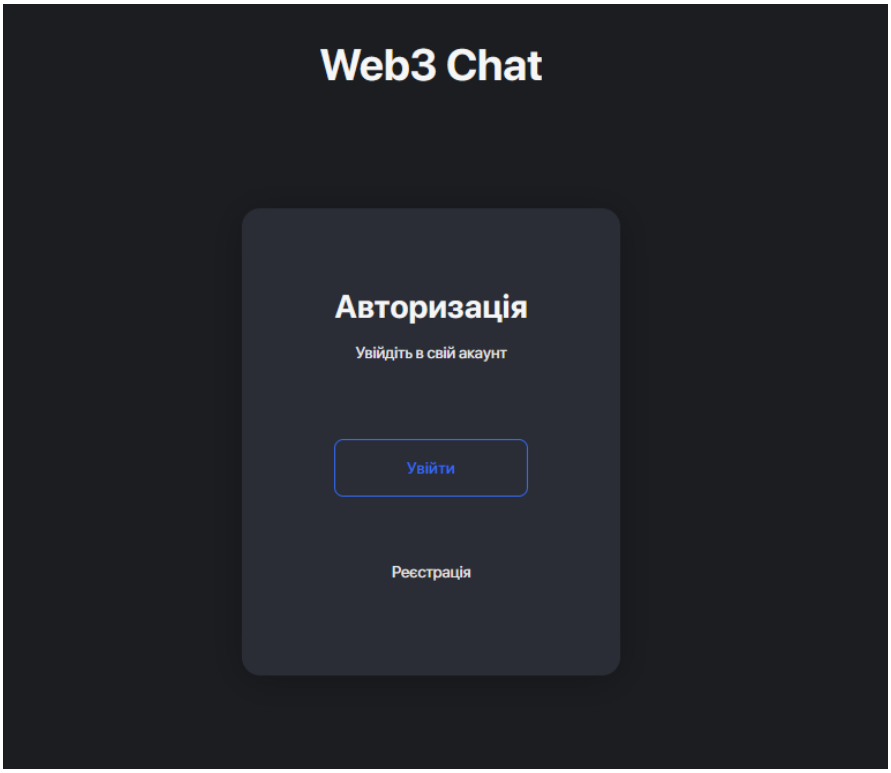

Рисунок 3.18 – Сторінка з авторизацією месенджеру

- Авторизація через криптогаманець показана на рисунках 3.19 та 3.20

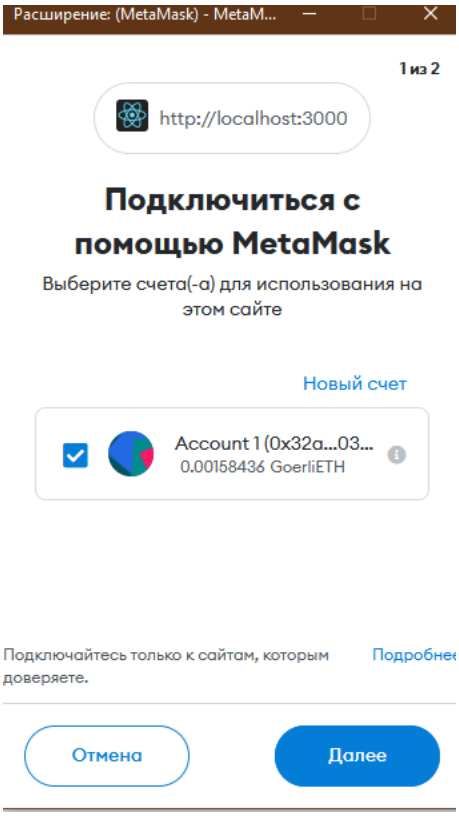

Рисунок 3.19 – Підключення криптогаманця

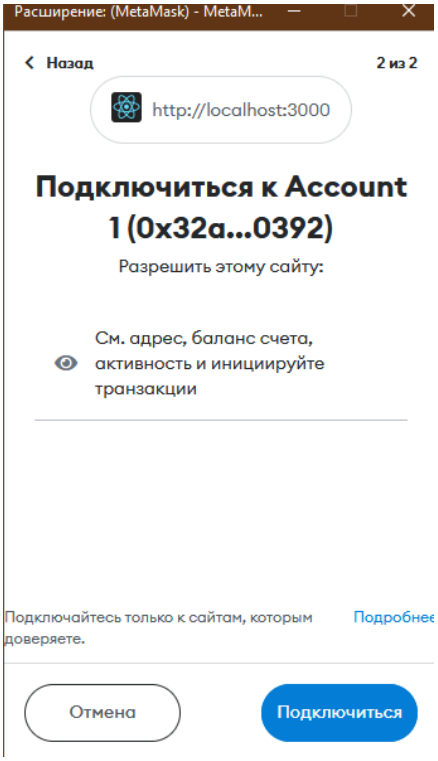

Рисунок 3.20 – Підключення криптогаманця

- Головна сторінка показана на рисунку 3.21

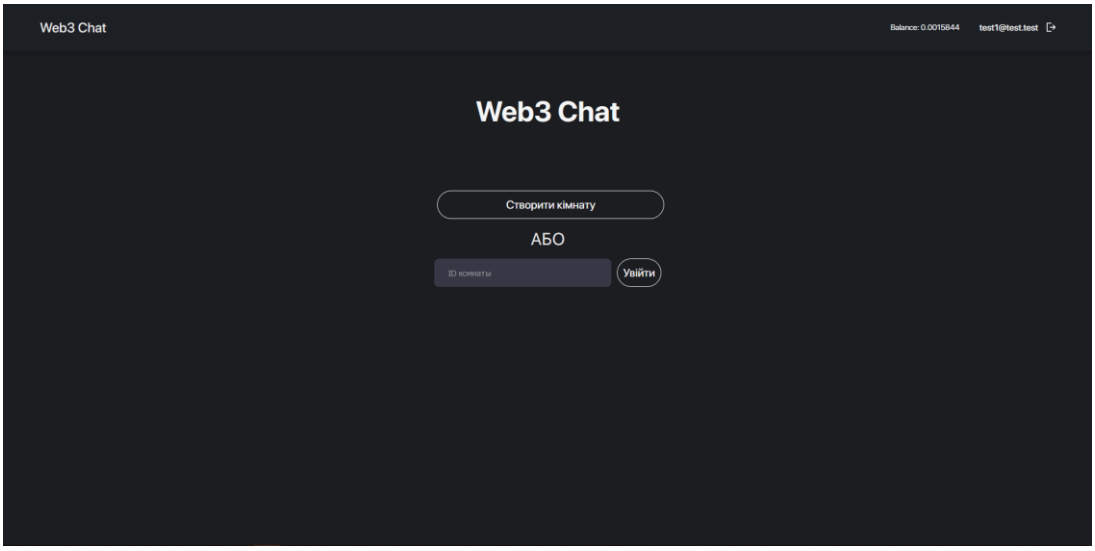

Рисунок 3.21 – Головна сторінка месенджеру

- Модальне вікно для створення кімнати показане на рисунку 3.22

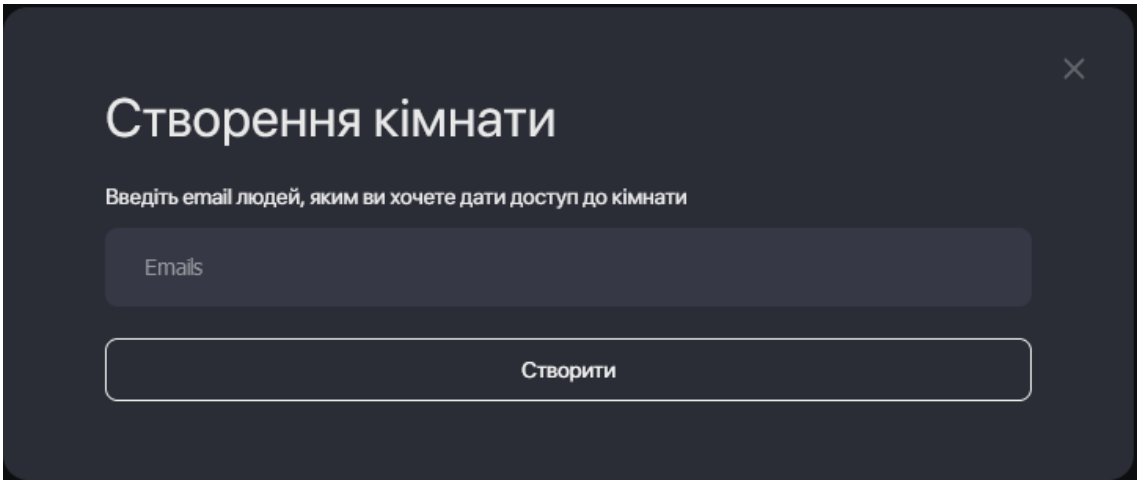

Рисунок 3.22 – Модальне вікно для створення кімнати

- Кімната месенджеру зображена на рисунку 3.23

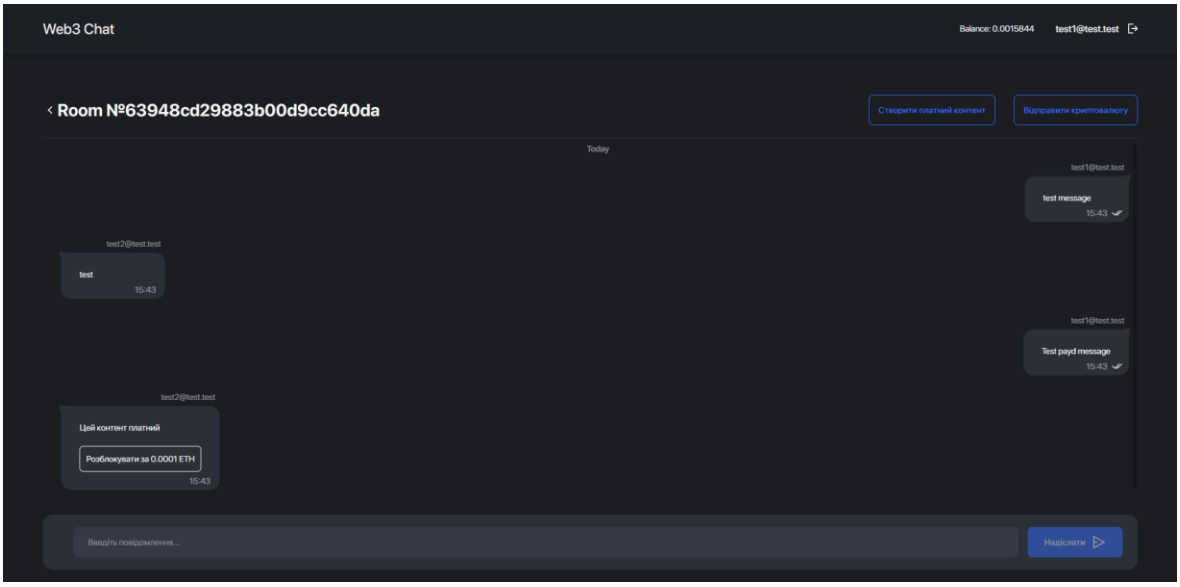

Рисунок 3.23 – Кімната месенджеру

- Модальне вікно для створення платного контенту зображене на рисунку 3.24

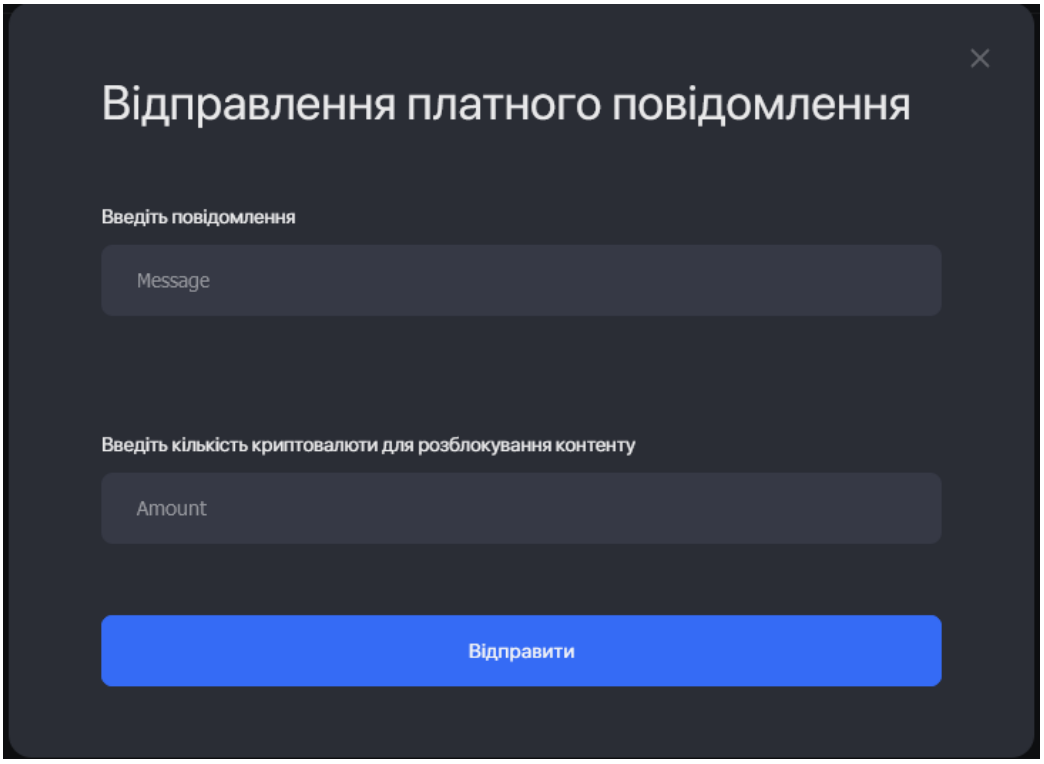

Рисунок 3.24 – Модальне вікно для створення платного контенту

- Модальне вікно для відправлення криптовалюти зображене на рисунку 3.25

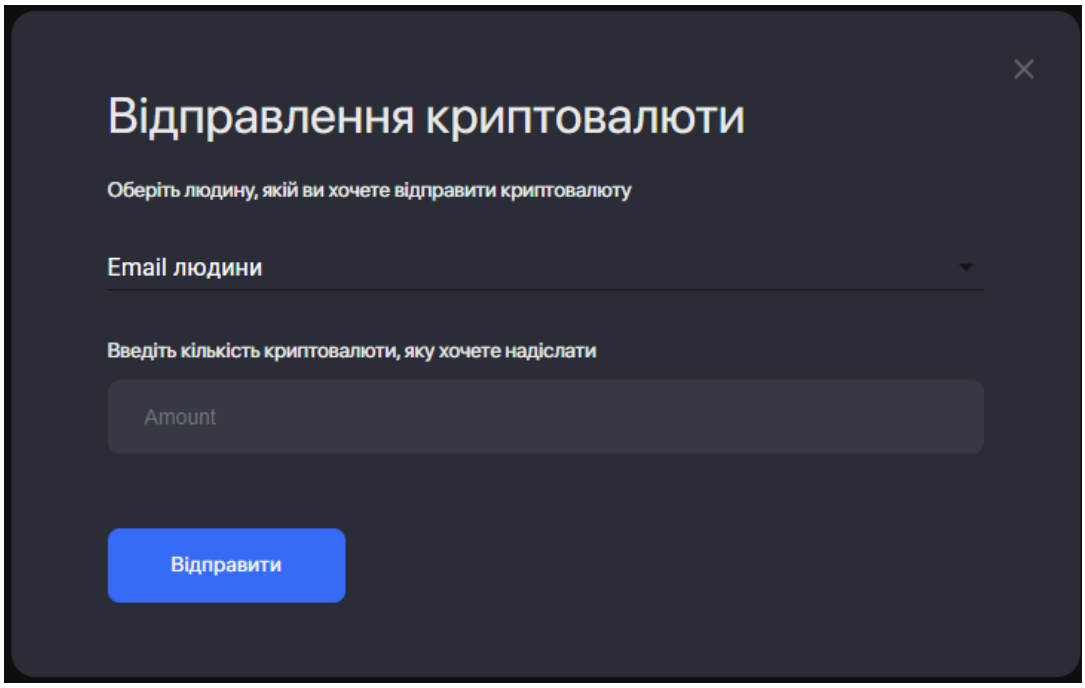

Рисунок 3.25 – Модальне вікно для відправлення криптовалюти

3.4 Можливі покращення додатку у майбутньому

В даний момент я бачу багато способів покращення та розвитку мого месенджеру.

Перше, що можна буде зробити це підтримка інших криптогаманців, окрім Metamask. Є багато гаманців, якими користуються на даний момент. Я планую додати підтримку TrustWallet, SafepalWallet, Phantom. Це найпопулярніші гаманці на даний момент, окрім Metamask.

Також я планую додати підтримку інших блокчейнів, окрім Ethereum. Найімовірнішими варіантами я бачу блокчейни Solana, Polygon та Binance Smart Chain. Ці блокчейни зараз користуються великою популярністю.

Ще планується програмне розширення функцій додатку. Я хочу зробити декілька видів користувачів, додати тип користувача, який буду сплачувати щомісячну підписку та буде мати додаткові привілеї. Також я хочу додати різні види кімнат або каналів з різним функціоналом.

Ще можна буде додати додаткову монетизацію, яку можна буде отримати за допомогою комісій при кожній транзакції.

3.5 Аналіз результатів програмної реалізації

В даному розділі спочатку було проведено моделювання додатку, а саме бази даних, подій та користувацького інтерфейсу. Також була продемонстрована реалізація додатку. Були показані конфігурація додатку, приклади коду та результуючі сторінки.

Функціонал, який був реалізований: авторизація по криптогаманцю, створення кімнати для спілкування, спілкування в реальному часі між користувачами, відправка криптовалюти користувачами, створення платного контенту. Як можна побачити, увесь функціонал, що планувався в постановці задачі, працює.

ВИСНОВКИ

Отже, в даній роботі я розглянув концепцію Web3, порівнявши її з Web1 та Web2. Також я зробив детальний огляд понять криптовалюта та блокчейн. Розповів як можна здобувати криптовалюту та зберігати. Також я розглянув криптогаманці, які вони є та їх різницю. Ще в даній роботі я розглянув різницю між proof-of-work та proof-of-stake.

Також в цій роботі було розроблено месенджер за концепцією Web3. Спочатку я зробив огляд обраних технологій, які я використовував для розробки. Для клієнтської частини я використовував UI бібліотеку ReactJS. Для серверної частини я використовував фреймворк Express.js. Як базу даних я використовував нереляційну MongoDB. Для мережі я використовував протоколи HTTP та Websocket. Для роботи з криптовалютою я обрав блокчейн Ethereum. Як криптогаманець для свого додатку я обрав Metamask.

Після цього вже був розроблений додаток. Спочатку я зробив моделювання додатку, змоделював усі події, змоделював початковий дизайн. Потім я продемонстрував сам етап розробки, показавши фрагменти коду, конфігурацію та структуру додатку. Наступним кроком я показав результати розробки додатку, показавши усі готові сторінки. Останнім кроком я розповів, як можна покращити мій додаток та що можна додати. В останньому кроці виявилося, що мій додаток ще можна досить сильно покращити, додавши різні блокчейни, підтримку різних криптогаманців, різні типи користувачів, монетезацію.

Я вважаю, що мій додаток вийшов досить самодостатнім та цікавим.

СПИСОК ЛІТЕРАТУРИ

1. Web3: What Is Web3? Potential of Web 3.0 / Patrick Ejeke // Springer, 2022. - C.15-47.
2. BlockChain 2035 / Jared Tate // Springer, 2019. - C.84-101.
3. Everything About Web 3.0 / Arya Ghobadi // Apress, 2021.-C.12-74.
4. The Basics of Bitcoins and Blockchains / Antony Lewis // No Starch Press, 2022.-C.57-58.
5. The Bitcoin Standard: The Decentralized Alternative to Central Banking / Saifedean Ammous // Springer, 2017.-C.119-128.
6. Blockchain Wars: The Future of Big Tech Monopolies and the Blockchain Internet / Evan McFarland // No Starch Press, 2019.-C.66-90.
7. Mastering Bitcoin: Programming the Open Blockchain / Andreas M. Antonopoulos // Candlewick, 2019.-C.89-107.
8. Blockchain Basics: A Non-Technical Introduction in 25 Steps / Daniel Drescher // Springer, 2021.-C.46-69.
9. The Truth Machine: The Blockchain and the Future of Everything / Michael J. Casey and Paul Vigna // Wiley, 2020.-C.14-18.
10. The Book of Crypto: The Complete Guide to Understanding Bitcoin, Cryptocurrencies and Digital Assets by Henri Arslanian / Palgrave Macmillan // Wiley, 2022,-C.72-80.
11. Beginning React / Greg Lim // Springer, 2020,-C.14-18.
12. Web Development with Node and Express: Leveraging the JavaScript Stack / Ethan Brown // Apress, 2018,-C.28-42.
13. MongoDB Basics / Peter Membrey // Candlewick, 2019.-C.38-48.
14. Out of the Ether / Matthew Leising // No Starch Press, 2020.-C.89-93.
15. A No-nonsense Analysis of Ethereum / Stephen Satoshi // Apress, 2020,-C.91-144.

ДОДАТОК

user.controller.js

```
exports.payMessage = (req, res) => {
    try {
        jwt.verify(
            req.headers["x-access-token"],
            keys.JWT_SECRET,
            (err, decoded) => {
                if (err) {
                    return res.status(401).send({ message: "Unauthorized!" });
                }
                Message.findByIdAndUpdate(
                    {
                        _id: req.params.id,
                    },
                    { paid: true },
                    function (err, result) {
                        if (err) {
                            res.send(err);
                        } else {
                            res.send(result);       You, 5 days ago • unlock message
                        }
                    }
                );
            }
        );
    } catch (e) {
        console.log(e);
    }
};
```

auth.controller.js

```
exports.signup = (req, res) => {
    const user = new User({
        username: req.body.username,
        email: req.body.email,
        account: req.body.account,       You, 7 days ago • wallet auth …
    });

    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!" });
        });
    });
};
```

message.js

```
const Message = model(
    "Message",
    new Schema({
        text: {
            type: String,
            required: true,
        },
        sender: {
            type: String,
            required: true,
        },
        create_date: {
            type: Date,
            required: true,
        },
        paid: {
            type: Boolean,
            required: true,
        },
        sender_account: {
            type: String,
            required: true,
        },
        unlock_price: {       You, 5 days ago • paid messages
            type: Number,
            required: false,
        },
    })
```

room.js

```
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) });      Maxym Mykytchenko, 19 months ago •
    this.messages = messages;

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

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

user.js

```
const User = model(
    "User",
    new Schema({
        username: {
            type: String,
            required: true,
        },
        email: {
            type: String,
            required: true,
        },
        account: {
            type: String,
            required: true,
        },       Maxym Mykytchenko, 19 months ago • init commit …
    })
);
```

chat.js

```
app.post("/api/pay_message/:id", [authJwt.verifyToken], controller.payMessage);

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", e);
        ws.terminate();
        return;
    }

    if (!token) {      Maxym Mykytchenko, 19 months ago • init commit
        console.log("no token");
        ws.terminate();
        return;
    }
```

user.js

```
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);      Maxym Mykytchenko, 19 mo

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

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

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

Messages.js

```
useEffect(() => {
    if (state.status === "Success") {
        payMessage(chosenMessage?.id);
        unlockMessage(chosenMessage?.idx);
        toast("Контент розблоковано успішно!");
    }
}, [state]);

  You, 5 days ago • unlock paid content …
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.
```

```
                                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>
                                            {paid || myEmail === sender ? (
```

```
                                    <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>
    </>
);
```

Chat.jsx

```
const rws = new ReconnectingWebSocket(
    `ws://localhost:8080/${id}/?token=${localStorage.getItem("token")}`
);

const handleChangeWallet = (event) => {
    setWallet(event.target.value);
};

const { sendTransaction, state } = useSendTransaction();

useEffect(() => {
    if (state.status === "Success") {
        setOpenDialog(false);
        setCryptoAmount("");
        setWallet("");
        setCryptoLoading(false);
        toast("Криптовалюту відправлено успішно!");      You, 5 days ago • unlock paid content
    }
}, [state]);
```

```
const submitForm = (data) => {
    const { text } = data;
    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);

    rws.send(JSON.stringify({ text: encryptedHex, paid: true }));
    reset();
};

const sendPaidMessage = (text, paid, unlock_price) => {
    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);

    rws.send(JSON.stringify({ text: encryptedHex, paid, unlock_price }));
    reset();
};
```

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

```
return (
    <div className="chat_page page_wrapper">
        <header className="chat_page__header section_header">
            <div
                style={{
                    display: "flex",
                    alignItems: "center",
                    justifyContent: "space-between",
                    width: "100%",
                }}
            >
                <div style={{ display: "flex", alignItems: "center" }}>
                    <div
                        className="back_link"
                        onClick={() => history.goBack()}
                        aria-label="Вернуться назад"
                    />
                    <h1 className="chat_page__title">Room №{id}</h1>
                </div>
                <div style={{ display: "flex", alignItems: "center" }}>
                    <DefaultButton
                        variant="outlined"
                        onClick={() => setOpenMessageDialog(true)}
                    >
                        Створити платний контент
                    </DefaultButton>
                    <DefaultButton
                        variant="outlined"
                        onClick={() => setOpenDialog(true)}
                        classes="btn_send_crypto"
                    >
                        Відправити криптовалюту
                    </DefaultButton>
                </div>
            </div>
        </header>
        {loading ? (
            <div style={{ marginTop: "220px" }}>
                <Loader />
            </div>
        ) : (
```

```
            <Messages
                messages={messagesList}
                handleScroll={handleScroll}
                scrollTo={scrollTo}
                setScrollTo={setScrollTo}
                loading={loading}
                myEmail={email && email.email}
                decryptKey={decryptKey}
                decrypt={decrypt}
                payMessage={payMessage}
                unlockMessage={unlockMessage}
            />
        )}

        <div className="chat_page__send_wrap">
            <form
                className="chat_page__send"
                onSubmit={handleSubmit(submitForm)}
            >
                <Field
                    name="text"
                    type="text"
                    component={RenderField}
                    placeholder="Введіть повідомлення..."
                />
                <DefaultButton formAction disabled={!textValue}>
                    <span>Надіслати</span>
                    <img src={sendIcon} alt="Отправить" />
                </DefaultButton>
            </form>
        </div>
        <DialogComponent
            open={openDialog}
            onClose={() => {
                setOpenDialog(false);
                setCryptoAmount("");
                setWallet("");
                setCryptoLoading(false);
            }}
        >
            <div className="dialog-create">
                <h2>Відправлення криптовалюти</h2>
                <p>
                    Оберіть людину, якій ви хочете відправити криптовалюту
                </p>
```

```
                </FormControl>
                <p style={{ marginTop: "30px" }}>
                    Введіть кількість криптовалюти, яку хочете надіслати
                </p>
                <input
                    style={{ marginBottom: "50px" }}
                    placeholder="Amount"
                    value={cryptoAmount}
                    onChange={(e) => setCryptoAmount(e.target.value)}
                />
                <DefaultButton
                    onClick={() => {
                        sendTransaction({
                            to: wallet,
                            value: parseEther(cryptoAmount),
                        });
                        setCryptoLoading(true);
                    }}
                    loading={cryptoLoading}
                    disabled={cryptoLoading}
                    classes="full_wdith_btn"
                >
                    Відправити
                </DefaultButton>
            </div>
        </DialogComponent>
        <DialogComponent
            open={openMessageDialog}
            onClose={() => {
                setOpenMessageDialog(false);
                setPaidMessage("");
                setPaidMessagePrice("");
            }}
        >
            <div className="dialog-create">
                <h2>Відправлення платного повідомлення</h2>
                <p style={{ marginTop: "30px" }}>Введіть повідомлення</p>
                <input
                    style={{ marginBottom: "50px" }}
                    placeholder="Message"
                    value={paidMessage}
                    onChange={(e) => setPaidMessage(e.target.value)}
                />
                <p style={{ marginTop: "30px" }}>
                    Введіть кількість криптовалюти для розблокування
                    контенту
                </p>
                <input
```

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

    return errors;
};

const ChatForm = 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,
        wallets: state.clients.wallets.users,
    };
};
const mapDispatchToProps = {
    getChatHistory,
    getMoreChatMessages,
    getNewMessage,
    sendMessage,
    getInvites,
    postKey,
    saveDecrypted,
    getWallets,
    payMessage,
    unlockMessage,
    reset: () => reset("ChatForm"),
};

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

index.js

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

Header.jsx

```
const Header = ({ getHeaderInfo, email, push }) => {
    useEffect(() => {
        getHeaderInfo();
    }, []);

    const { account } = useEthers();
    const balance = useEtherBalance(account);

    return (
        <header className="header">
            <Link to={`/`}>
                <p className="header__logo">Web3 Chat</p>
            </Link>
            <div className="header__user">
                {balance && (
                    <span style={{ marginRight: "35px" }}>      You, 6 days ago • send crypto …
                        Balance: {(+formatEther(balance)).toFixed(7)}
                    </span>
                )}
                <span className="header__email">{email && email.email}</span>
                <button
                    className="header__logout good_hover"
                    onClick={() => {
                        localStorage.removeItem("token");
                        push("/auth/login");
                    }}
                >
                    Вийти
                </button>
            </div>
        </header>
    );
};

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