КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від _____, протокол №__.
Завідувач кафедри доктор фіз.-мат. наук, професор
_____

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«Програма моніторингу локальної мережі з можливістю обміну повідомленнями між користувачами»

Виконав:
студент 4-го курсу
денної форми навчання
спеціальності 172 – Телекомунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»

_____ Сіренко Максим

Науковий керівник:
канд. фіз.-мат. наук, асистент
_____ Богданов Р.В.

Рецензент:
канд. техн. наук, доцент
_____ Загороднюк С. П.

ЗМІСТ

РЕФЕРАТ

Дипломна робота: 45 с., 3 табл., 23 рис., 13 джерел.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБМІНУ ПОВІДОМЛЕННЯМИ, ЛОКАЛЬНА МЕРЕЖА, МЕСЕНДЖЕР, DJANGO, WEBSOCKET.

Об’єкт розроблення - програмне забезпечення для обміну повідомленнями в локальній мережі з функціями моніторингу та адміністративного керування.

Мета роботи - розробити веб-застосунок для локальної мережі, який забезпечує обмін текстовими повідомленнями та файлами в групових і приватних чатах, а також включає адміністративну панель на базі Django для керування чатами, повідомленнями та користувачами.

У роботі проведено аналіз сучасних месенджерів, зокрема Slack, Microsoft Teams, Telegram, WhatsApp, Viber, з акцентом на їхні функціональні можливості, безпеку та підтримку локальних мереж. Розроблено веб-застосунок на основі фреймворку Django з використанням бібліотеки Channels для реалізації асинхронного обміну повідомленнями через протокол WebSocket. Застосунок підтримує аутентифікацію користувачів за допомогою бібліотеки django-allauth, створення профілів з аватарами та відображенням імен, відправлення текстових повідомлень і файлів, а також створення групових і приватних чатів.

Адміністративна панель дозволяє створювати, редагувати та видаляти чати, видаляти повідомлення та користувачів, а також моніторити активність у локальній мережі. Для розробки використано SQLite як базу даних із можливістю переходу на PostgreSQL для масштабування. Інтерфейс застосунку реалізовано з використанням Tailwind CSS, HTMX та Alpine.js для забезпечення інтерактивності та зручності користувача.

ВСТУП

У сучасному світі організації дедалі частіше потребують надійних і безпечних засобів для внутрішньої комунікації в межах локальних мереж. Існуючі хмарні месенджери, такі як Slack чи Telegram, попри їхню зручність, не завжди відповідають вимогам ізольованих середовищ через залежність від глобальної мережі та недостатній контроль над даними. Натомість локальні месенджери забезпечують автономне спілкування, ефективно захищаючи інформацію від зовнішніх загроз. Крім того, у закритих мережах критично важливим є централізоване адміністративне керування, що включає модерацію чатів, контроль користувачів та моніторинг активності. Наразі спостерігається дефіцит комплексних рішень, які б поєднували ці можливості, що зумовлює актуальність розробки такого програмного забезпечення.

Ця робота зосереджена на створенні месенджера для локальної мережі, доповненого адміністративною панеллю. Розроблений застосунок забезпечить можливість обміну текстовими повідомленнями та файлами, створення групових і приватних чатів, а також надасть адміністраторам засоби для управління контентом та відстеження активності в мережі. Ключовою особливістю програми є її здатність функціонувати без постійного доступу до інтернету, що є визначальним фактором для організацій із підвищеними вимогами до безпеки даних.

Об’єктом дослідження є організація ефективних комунікацій у локальній мережі та систем адміністративного керування. Предметом дослідження виступає програмне забезпечення, що реалізує функції месенджера, адміністративної панелі та механізмів мережевого моніторингу. Мета роботи полягає у розробці веб-застосунку на базі фреймворку Django з використанням протоколу WebSocket для організації чатів, а також інтеграції адміністративних функцій та можливостей моніторингу мережевої активності.

У роботі використано методи системного аналізу літературних джерел, принципи об'єктно-орієнтованого програмування на Python, методики

проєктування реляційних баз даних на основі SQLite та комплексне тестування програмного продукту.

РОЗДІЛ І. АНАЛІТИЧНИЙ ОГЛЯД СУЧАСНИХ МЕСЕНДЖЕРІВ

1.1. Аналіз функціональних можливостей та архітектури месенджерів

Проведено аналіз функціоналу та архітектури провідних месенджерів, які широко використовуються як для персональних, так і для корпоративних комунікацій. До цих платформ належать Slack, Microsoft Teams, Telegram, WhatsApp та Viber, кожна з яких має свої архітектурні особливості та функціональні обмеження в контексті автономної роботи.

Slack позиціонується як інструмент для командної роботи, що надає широкі можливості для створення тематичних каналів, обміну файлами та інтеграції зі сторонніми сервісами, такими як Google Drive, Jira або Zoom. Його архітектура базується на хмарній інфраструктурі, що є визначальним фактором залежності від зовнішнього доступу до мережі Інтернет. Це обмежує його застосування в автономних локальних мережах (Рис. 1.1 - Логотип Slack).

Рисунок 1.1 - Логотип Slack

Переваги Slack

- Структуровані канали для спілкування.
- Інтеграція зі сторонніми сервісами.
- Кросплатформність (мобільні, декстоп, веб).

Недоліки Slack

- Залежність від інтернету.
- Обмеження безкоштовної версії.
- Висока вартість платних планів.

- Складність для новачків.

Microsoft Teams, інтегрований у екосистему Microsoft 365, орієнтований на корпоративних користувачів. Його функціонал охоплює групові чати, відеоконференції, обмін файлами та взаємодію з OneDrive та SharePoint. Подібно до Slack, Microsoft Teams є суто хмарним рішенням, що вимагає постійного інтернет-з’єднання для повноцінного функціонування (Рис. 1.2 - Логотип Microsoft Teams).

Рисунок 1.2 - Логотип Microsoft Teams

Переваги Microsoft Teams

- Інтеграція з Microsoft 365 (Outlook, OneDrive, Word тощо).
- Підтримка відеоконференцій із великою кількістю учасників.
- Кросплатформність (мобільні, десктоп, веб).
- Зручний інтерфейс для командної роботи.

Недоліки Microsoft Teams

- Обмеження безкоштовної версії
- Високе споживання ресурсів на слабких пристроях
- Складність налаштування для нових користувачів

Telegram відомий своєю швидкістю та акцентом на конфіденційність, підтримуючи групові чати зі значною кількістю учасників, приватні повідомлення та функцію "секретних чатів" з наскрізним шифруванням. Архітектура Telegram побудована на розподілених хмарних серверах, що забезпечує високу доступність

та синхронізацію даних, однак унеможливлює його розгортання та автономну роботу в локальних мережах без зовнішнього доступу (Рис. 1.3 - Логотип Telegram).

Рисунок 1.3 - Логотип Telegram

Переваги Telegram

- Підтримка груп і каналів із великою кількістю учасників.
- Доступність на всіх популярних платформах (мобільні, десктоп, веб).
- Можливість приховати номер телефону.
- Хмарне зберігання чатів із доступом із будь-якого пристрою.

Недоліки Telegram

- Наскрізне шифрування не ввімкнено за замовчуванням (лише для секретних чатів).
- Для реєстрації потрібен номер телефону.
- Протокол шифрування не перевірений незалежними експертами.

WhatsApp є одним з найпопулярніших месенджерів, що використовує наскрізне шифрування для всіх типів чатів. Він підтримує текстові повідомлення, обмін файлами, зображеннями та голосові/відеодзвінки. Проте WhatsApp орієнтований на індивідуальних користувачів та повністю залежить від хмарних серверів, що робить його непридатним для використання в ізольованих корпоративних мережах (Рис. 1.4 - Логотип WhatsApp).

Рисунок 1.4 - Логотип WhatsApp

Переваги WhatsApp

- Наскрізне шифрування за замовчуванням.
- Підтримка голосових і відеодзвінків.
- Доступність на мобільних і десктопних платформах.
- Широка популярність у світі.

Недоліки WhatsApp

- Для використання потрібен номер телефону та доступ до контактів.
- Обмежений функціонал для групового спілкування порівняно з Telegram.
- Залежність від інтернету.

Viber має схожий функціонал з WhatsApp, пропонуючи наскрізне шифрування, групові чати та обмін мультимедійними повідомленнями. Як і попередні аналоги, Viber є хмарним рішенням і не може функціонувати без підключення до мережі інтернет (Рис. 1.5 - Логотип Viber).

Рисунок 1.5 - Логотип WhatsApp

Переваги Viber

- Наскрізне шифрування для чатів і дзвінків.
- Підтримка голосових і відеодзвінків високої якості.
- Доступність на мобільних, десктопних і веб-платформах.
- Можливість створювати групові чати та спільноти.

Недоліки Viber

- Для реєстрації потрібен номер телефону.
- Менша популярність порівняно з WhatsApp і Telegram.
- Наявність реклами в безкоштовній версії.
- Високе споживання ресурсів на деяких пристроях.

Таблиця 1.1.

Порівняння функціональних можливостей месенджерів

Месенджер	**Групові чати**	**Обмін файлами**	**Шифрування**	**Адмін-функції**	**Локальна мережа**
Slack	Так	Так	У транзиті/спокої	Обмежені (платні тарифи)	Ні

Microsoft Teams	Так	Так	E2EE (частково), GDPR	Керування доступом, контентом	Ні
Telegram	Так	Так	E2EE (секретні чати)	Базова модерація груп	Ні
WhatsApp	Так	Так	E2EE (усі чати)	Базова модерація груп	Ні
Viber	Так	Так	E2EE (усі чати)	Базова модерація груп	Ні

1.2. Висновки щодо застосовності існуючих рішень для локальних мереж

Проведений аналітичний огляд сучасних месенджерів свідчить про їхню суттєву залежність від хмарної інфраструктури. Ця особливість є критичним обмежувальним фактором, що унеможливлює повноцінне використання даних рішень в ізольованих локальних мережах. Окрім того, адміністративні функції у більшості розглянутих платформ або мають вкрай обмежений функціонал, або доступні виключно у платних версіях, які також зберігають хмарну залежність. Такий стан справ підкреслює нагальну потребу у розробці месенджера, який би функціонував автономно в межах локальної мережі та був оснащений потужною, гнучкою та централізованою адміністративною панеллю.

РОЗДІЛ ІІ. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОЄКТУВАННЯ МЕСЕНДЖЕРА

Розробка месенджера, призначеного для локальної мережі та доповненого адміністративною панеллю, є прямою відповіддю на актуальні потреби організацій у забезпеченні безпечного, автономного та контрольованого середовища для комунікацій. У сучасному корпоративному, освітньому та державному секторах, де функціонують ізольовані локальні мережі, існує попит на програмне забезпечення, що здатне працювати без зовнішнього доступу до мережі Інтернет, гарантувати захист конфіденційних даних та надавати адміністраторам повний контроль над внутрішньою комунікацією. Таким чином, месенджер має забезпечувати не тільки швидкий обмін текстовими повідомленнями та файлами, а й підтримку групових та приватних чатів, надійну аутентифікацію користувачів та інтуїтивно зрозумілий користувацький інтерфейс. Ключовим елементом системи є адміністративна панель, яка дозволяє ефективно управляти чатами, модерувати контент, керувати обліковими записами користувачів та забезпечувати відповідність внутрішнім політиками безпеки.

Предметна область даного дослідження охоплює комплексне управління комунікаціями в локальній мережі, включаючи: створення та функціонування каналів зв'язку, передачу даних у реальному часі, архівацію та зберігання інформації про користувачів та чати, а також гнучке керування доступом до системи. Основні виклики, що постають під час реалізації такого проєкту, включають забезпечення стабільної та надійної роботи без залежності від зовнішніх серверів, захист від несанкціонованого доступу та розробку ергономічного дизайну, який буде доступним для користувачів з різним рівнем технічної підготовки. Для вирішення цих завдань було обрано оптимальні сучасні технології, які поєднують продуктивність, безпеку та гнучкість розробки.

2.1. Архітектурне рішення системи

Архітектура месенджера побудована на принципах клієнт-серверної моделі, що забезпечує високу стабільність та ефективність функціонування в умовах локальної мережі. Система інтегрує три ключові компоненти: клієнтську частину, серверну частину та підсистему зберігання даних, взаємодія між якими забезпечується через відповідні протоколи.

Клієнтська частина являє собою веб-інтерфейс, який функціонує безпосередньо у веб-браузері користувача. Її реалізація охоплює HTML-шаблони для формування візуального контенту сторінок авторизації, профілю та чатів. CSS-стилі відповідають за естетичне оформлення інтерфейсу, а JavaScript забезпечує інтерактивність та динамічне відображення елементів. Взаємодія клієнта із сервером здійснюється за допомогою двох основних протоколів: HTTP, який використовується для обробки статичних запитів, таких як завантаження профілю користувача, та WebSocket, що організує двосторонній асинхронний обмін повідомленнями в реальному часі. Використання WebSocket дозволяє серверу миттєво доставляти нові повідомлення всім учасникам чату без необхідності ініціювання запитів з боку клієнта, що значно покращує швидкість взаємодії.

Серверна частина реалізована на базі фреймворку Django, який відповідає за обробку HTTP-запитів, керування сесіями користувачів та забезпечення механізмів безпечної аутентифікації. Для асинхронного обміну повідомленнями інтегровано бібліотеку Django Channels, що розширює функціонал Django, дозволяючи йому працювати з протоколом WebSocket. Channels оперує "споживачами" (consumers) — спеціалізованими класами, які обробляють вхідні повідомлення через WebSocket та розсилають їх відповідним учасникам чату. Наприклад, при надсиланні повідомлення в груповий чат, consumer зберігає його у базі даних та негайно транслює всім учасникам через WebSocket. Додатково, сервер відповідає за безпечне зберігання файлів, завантажених користувачами, у локальній файловій системі після відповідної перевірки на безпеку.

Підсистема зберігання даних представлена базою даних SQLite, яка використовується для архівації інформації про користувачів, чати та повідомлення. Вибір SQLite обумовлений простотою розгортання та достатньою продуктивністю для підтримки локальної мережі з десятками користувачів. Основна структура бази даних включає чотири ключові таблиці: User, яка містить облікові дані користувачів; Profile, що розширює дані User через зв'язок "один до одного", включаючи аватар та відображуване ім'я; ChatGroup, яка містить інформацію про групові та приватні чати; та GroupMessage, що зберігає деталі повідомлень. Зв'язки між таблицями забезпечують цілісність даних, наприклад, каскадне видалення повідомлень при видаленні відповідного чату. Архітектура передбачає можливість подальшої міграції на PostgreSQL для підвищення масштабованості та підтримки більшої кількості одночасних користувачів. Безпека системи гарантується захистом WebSocket-з'єднань від неавторизованих спроб підключення та використанням стандартних механізмів безпеки Django для запобігання поширеним веб-вразливостям, таким як CSRF та XSS.

Таблиця 2.1.

Основні компоненти архітектури месенджера

Компонент	**Технології**	**Основна функція**
Клієнтська частина	HTML, CSS (Tailwind CSS), JavaScript (HTMX, Alpine.js)	Відображення інтерфейсу, користувацька взаємодія, динамічні оновлення
Серверна частина	Django, Django Channels, Python	Обробка запитів, аутентифікація, логіка чатів, зберігання файлів
База даних	SQLite (з можливістю міграції на PostgreSQL)	Зберігання даних про користувачів, чати, повідомлення
Протоколи	HTTP, WebSocket	Взаємодія клієнт-сервер, обмін повідомленнями в реальному часі

2.3. Вибір та обґрунтування технологічного стеку

Вибір технологічного стеку для реалізації месенджера базувався на критеріях продуктивності, безпеки, зручності розробки та масштабованості. Використані рішення оптимально поєднують ці аспекти для забезпечення ефективного функціонування в умовах локальної мережі.

Основним фреймворком для розробки серверної логіки обрано Django. Його переваги полягають у наявності потужного ORM для роботи з базою даних, гнучкої системи шаблонів для рендерингу веб-сторінок, а також вбудованих механізмів аутентифікації та захисту від типових веб-атак. Вибір Django обґрунтований його високою надійністю, розвиненою спільнотою та підтримкою асинхронних розширень, зокрема Channels.

За реалізацію функціоналу чатів у реальному часі відповідає Django Channels у поєднанні з протоколом WebSocket. Channels розширює можливості Django для обробки асинхронних протоколів, дозволяючи створювати постійні двосторонні з'єднання між клієнтом і сервером. Це суттєво зменшує затримки при обміні повідомленнями, забезпечуючи миттєве оновлення чату без повного перезавантаження сторінки.

Для управління системою аутентифікації користувачів інтегровано бібліотеку django-allauth. Вона забезпечує функціонал локальної реєстрації та авторизації, надійний захист паролів за допомогою криптографічного хешування та підтримку двофакторної аутентифікації, що підвищує безпеку облікових записів.

Як реляційна база даних для зберігання всіх даних системи обрана SQLite. Її переваги включають легкість розгортання, оскільки вона не потребує окремого сервера, та достатню продуктивність для сценаріїв використання в локальних мережах з помірною кількістю користувачів. Для майбутнього масштабування архітектура передбачає можливість переходу на PostgreSQL, що забезпечить більшу продуктивність та надійність для великих навантажень.

Стилізація користувацького інтерфейсу виконана за допомогою Tailwind CSS. Цей утилітарний CSS-фреймворк дозволяє швидко та ефективно створювати

адаптивний, сучасний та чистий дизайн. Використання готових класів прискорює розробку, забезпечуючи при цьому гнучкість у налаштуванні зовнішнього вигляду компонентів, що є критичним для забезпечення зручності користувачів.

Для забезпечення динамічних оновлень сторінок без повного перезавантаження інтегровано HTMX. Це дозволяє оновлювати лише необхідні фрагменти інтерфейсу, наприклад, список повідомлень у чаті, що значно покращує швидкість взаємодії та користувацький досвід. HTMX сприяє створенню відчуття "односторінкового застосунку" без складності повноцінних JavaScript-фреймворків.

Клієнтська інтерактивність додана за допомогою Alpine.js. Він використовується для реалізації таких елементів, як випадаючі меню, модальні вікна та перемикачі. Завдяки своїй мінімалістичності та низькому обсягу коду, Alpine.js ідеально підходить для проектів, де потрібна базова, але ефективна клієнтська логіка без додаткового навантаження на продуктивність.

Таким чином, обраний технологічний стек забезпечує оптимальне поєднання простоти розгортання з можливістю масштабування, що робить розроблений месенджер гнучким, ефективним та придатним для використання в різноманітних локальних мережах.

Таблиця 2.2.

Обґрунтування вибору технологій

Технологія	**Роль**	**Обґрунтування вибору**
Django	Серверний фреймворк	Надійність, ORM, система шаблонів, безпека, велика спільнота
Django Channels	Асинхронна комунікація, WebSocket	Розширення Django для реального часу, двосторонній зв'язок
django-allauth	Система аутентифікації	Локальна реєстрація/авторизація, хешування паролів

SQLite	База даних	Легкість розгортання, продуктивність для малих мереж
Tailwind CSS	Стилізація інтерфейсу	Швидка розробка адаптивного дизайну, утилітарні класи
HTMX	Динамічні оновлення (AJAX)	Оновлення фрагментів сторінки без перезавантаження, UX
Alpine.js	Клієнтська інтерактивність	Легкість, мінімальний обсяг коду, швидке завантаження

2.4. Архітектура та функціональні можливості системи

Розроблений веб-застосунок є комплексним рішенням для обміну інформацією в локальній мережі, що поєднує функціонал месенджера з розширеними можливостями адміністрування та моніторингу. Система побудована на фреймворку Django з використанням асинхронних комунікацій на базі WebSocket, що забезпечує обмін повідомленнями та файлами в реальному часі.

Застосунок реалізує повний цикл аутентифікації та авторизації користувачів, забезпечуючи надійний захист облікових записів за допомогою бібліотеки django-allauth. Кожен користувач може створити та налаштувати власний профіль, додавши аватар і обравши відображуване ім'я. Система дозволяє організовувати гнучку комунікацію як у групових, так і в приватних чатах, зберігаючи повну історію повідомлень. Центральною функцією є миттєва доставка текстових повідомлень, доповнена можливістю завантаження файлів різних форматів, при цьому забезпечується автоматична перевірка їхнього типу та розміру для запобігання несанкціонованому контенту. Для ефективного керування системою розроблено захищену адміністративну панель, яка надає повний контроль над чатами, повідомленнями та користувачами, а також функціонал моніторингу активності у мережі.

Безпека є невід'ємною частиною архітектури системи. Механізми аутентифікації захищені від брутфорс-атак, а всі WebSocket-з’єднання доступні виключно для авторизованих користувачів, що мінімізує ризики несанкціонованого доступу та потенційного перехоплення даних у локальній мережі. Фреймворк Django забезпечує вбудований захист від типових веб-вразливостей, таких як CSRF, XSS та SQL-ін’єкції, завдяки інтегрованим механізмам екранування даних, використання CSRF-токенів та ORM. Завантажувані файли проходять ретельну перевірку та зберігаються в ізольованій папці, а архітектура системи передбачає можливість подальшої інтеграції наскрізного шифрування повідомлень для підвищення конфіденційності. Додатково, передбачено можливість резервного копіювання бази даних для забезпечення швидкого відновлення інформації у випадку системних збоїв.

РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ МЕСЕНДЖЕРА ДЛЯ ЛОКАЛЬНОЇ МЕРЕЖІ

Даний розділ присвячено практичній реалізації месенджера для локальної мережі, доповненого адміністративною панеллю. На основі вимог та архітектури, детально описаних у попередньому розділі, було створено веб-додаток, який забезпечує ключовий функціонал: групові та приватні чати, ефективний обмін повідомленнями й файлами, надійну аутентифікацію користувачів, а також розширене адміністративне керування. Процес реалізації охоплював розробку як серверної, так і клієнтської частин, проєктування оптимальної структури бази даних, налаштування протоколу WebSocket для забезпечення обміну даними в реальному часі, створення інтуїтивно зрозумілого користувацького інтерфейсу та комплексне тестування функціональності системи в умовах локальної мережі. Для досягнення поставлених цілей були використані сучасні технології, такі як Django, Channels, SQLite, Tailwind CSS, HTMX та Alpine.js, що дозволило забезпечити стабільність, безпеку та зручність експлуатації розробленої системи.

3.1. Проєктування бази даних

База даних є фундаментальною основою для зберігання всієї необхідної інформації про користувачів, чати та повідомлення, забезпечуючи цілісність та доступність даних у месенджері. Для реалізації системи обрано реляційну базу даних SQLite, що обумовлено її легкістю розгортання та достатньою продуктивністю для локальної мережі з помірною кількістю користувачів. Структура бази даних включає чотири основні таблиці, які є ключовими для забезпечення повноцінного функціоналу месенджера.

Перша таблиця, *User*, створюється автоматично фреймворком Django і призначена для зберігання основних облікових даних користувачів. Вона містить інформацію про логін, хешований пароль, адресу електронної пошти та дату реєстрації. Ця таблиця є основою для системи аутентифікації, реалізованої за

допомогою бібліотеки django-allauth. Важливо зазначити, що паролі зберігаються у хешованому вигляді із застосуванням алгоритму PBKDF2, що гарантує їхню безпеку навіть у випадку компрометації бази даних.

Таблиця *Profile* розширює інформацію, що міститься в таблиці User, через зв'язок "один до одного". Вона включає додаткові поля, такі як шлях до зображення аватара користувача у файловій системі, відображуване ім’я, яке використовується для ідентифікації в чатах (наприклад, "Олег К." замість технічного логіна), та статус користувача (активний чи заблокований). Профіль відіграє важливу роль у візуальному представленні користувачів у інтерфейсі месенджера.

Третя таблиця, *ChatGroup*, призначена для представлення групових та приватних чатів у системі. Вона містить унікальний ідентифікатор чату, його назву (наприклад, "Проєкт 2025"), дату створення, посилання на власника чату (зв'язок з таблицею User) та список учасників, реалізований як зв'язок "багато до багатьох" з таблицею User. Для приватних чатів характерною особливістю є наявність лише двох учасників та відсутність явно заданої назви, оскільки в інтерфейсі замість неї відображається ім’я співрозмовника.

Остання ключова таблиця, *GroupMessage*, відповідає за зберігання всіх повідомлень, що надсилаються в чатах. Її поля включають унікальний ідентифікатор повідомлення, посилання на чат, до якого воно належить (ChatGroup), автора повідомлення (User), текстовий вміст повідомлення, шлях до прикріпленого файлу (якщо такий є) та точний час відправлення. Важливою особливістю є реалізація каскадного видалення: у випадку видалення чату, всі пов’язані з ним повідомлення автоматично видаляються з таблиці GroupMessage, що запобігає накопиченню "сміттєвих" даних та забезпечує цілісність бази даних.

Нижче наведено приклад (Рисунок 3.1 - ChatGroup з models.py) моделі *ChatGroup* із *models.py*, що ілюструє структуру даних та зв'язки:

```python
from django.db import models
from django.contrib.auth.models import User
import shortuuid

class ChatGroup(models.Model):
    group_name = models.CharField(max_length=128, unique=True, default=shortuuid.uuid)
    groupchat_name = models.CharField(max_length=128, null=True, blank=True)
    admin = models.ForeignKey(User, related_name='groupchats', blank=True, null=True, on_delete=models.SET_NULL)
    users_online = models.ManyToManyField(User, related_name='online_in_groups', blank=True)
    members = models.ManyToManyField(User, related_name='chat_groups', blank=True)
    is_private = models.BooleanField(default=False)

    def __str__(self):
        return self.group_name
```

Рисунок 3.1 - ChatGroup з models.py

ER-діаграма бази даних, що відображає зв’язки: User має один Profile, один користувач може бути власником кількох ChatGroup, а ChatGroup містить багато GroupMessage.

3.2. Реалізація серверної частини

Серверна частина месенджера розроблена на базі фреймворку Django, який відповідає за обробку HTTP-запитів, механізми аутентифікації користувачів та рендеринг веб-шаблонів. Для забезпечення функціоналу чатів у реальному часі використано бібліотеку Django Channels, що працює за протоколом WebSocket.

Аутентифікація користувачів реалізована за допомогою бібліотеки django-allauth. Ця бібліотека була налаштована для забезпечення локальної реєстрації та авторизації в системі. Користувачі створюють облікові записи, вказуючи логін, пароль та електронну пошту, після чого отримують доступ до свого профілю. Важливо, що всі форми реєстрації та входу захищені від Cross-Site Request Forgery (CSRF) атак за допомогою спеціальних токенів, а паролі користувачів хешуються, що підвищує безпеку даних. Приклад конфігурації локальної авторизації в файлі settings.py демонструє підключення необхідних додатків та налаштування методу аутентифікації (Рисунок 3.2 - Конфігурація локальної авторизації в settings.py):

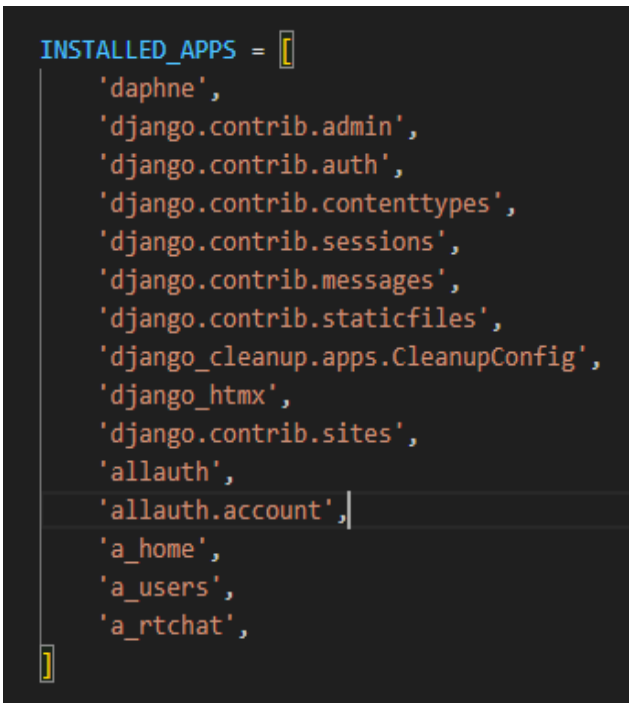

Рисунок 3.2 - Конфігурація локальної авторизації в settings.py

Функціонал чатів у реальному часі та обмін повідомленнями реалізовано за допомогою Django Channels. Асинхронний обмін повідомленнями забезпечується класом ChatConsumer, який міститься у файлі consumers.py і відповідає за обробку WebSocket-з’єднань. Коли користувач приєднується до певного чату, consumer додає його до відповідної групи Channels, що ідентифікується об'єктом ChatGroup. При відправленні повідомлення, consumer зберігає його вміст у базі даних, а потім розсилає всім учасникам цього чату через WebSocket, забезпечуючи миттєву доставку. Приклад реалізації ChatConsumer наведено нижче, демонструючи логіку підключення, прийому та розсилки повідомлень (Рисунок 3.3 - Приклад реалізації ChatConsumer):

```python
from channels.generic.websocket import WebsocketConsumer
from django.shortcuts import get_object_or_404
from django.template.loader import render_to_string
from asgiref.sync import async_to_sync
import json
from .models import *


class ChatroomConsumer(WebsocketConsumer):
    def connect(self):
        self.user = self.scope['user']
        self.chatroom_name = self.scope['url_route']['kwargs']['chatroom_name']
        self.chatroom = get_object_or_404(ChatGroup, group_name=self.chatroom_name)

        async_to_sync(self.channel_layer.group_add)(
            self.chatroom_name, self.channel_name
        )

        #add
        if self.user not in self.chatroom.users_online.all():
            self.chatroom.users_online.add(self.user)
            self.update_online_count()


        self.accept()

    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)(
            self.chatroom_name, self.channel_name
        )

        if self.user in self.chatroom.users_online.all():
            self.chatroom.users_online.remove(self.user)
            self.update_online_count()
```

Рисунок 3.3 - Приклад реалізації ChatConsumer

Обробка файлів, що завантажуються користувачами, здійснюється спеціально розробленим view-контролером. Після успішної перевірки файли зберігаються у локальній папці media/uploads, а шлях до завантаженого файлу додається до відповідного об'єкта GroupMessage у базі даних. Приклад реалізації UploadFileView для завантаження файлів (Рисунок 3.4 - Приклад реалізації UploadFileView):

```python
def chat_file_upload(request, chatroom_name):
    chat_group = get_object_or_404(ChatGroup, group_name=chatroom_name)

    if request.htmx and request.FILES:
        file = request.FILES['file']
        message = GroupMessage.objects.create(
            file = file,
            author = request.user,
            group = chat_group,
        )
        channel_layer = get_channel_layer()
        event = {
            'type': 'message_handler',
            'message_id': message.id,
        }
        async_to_sync(channel_layer.group_send)(
            chatroom_name, event
        )
    return HttpResponse()
```

Рисунок 3.4 - Приклад реалізації UploadFileView

3.3. Реалізація клієнтської частини

Клієнтська частина месенджера представляє собою веб-інтерфейс, який функціонує у веб-браузері користувача. Вона складається з HTML-шаблонів, які визначають структуру сторінок, стилів Tailwind CSS, що відповідають за візуальне оформлення, а також скриптів HTMX та Alpine.js, що забезпечують інтерактивність та динамічне оновлення контенту.

HTML-шаблони складають основу користувацького інтерфейсу. Серед них виділяються base.html, який слугує базовою структурою з навігацією; login.html для форми входу; chat.html, що відображає список чатів та вікно повідомлень; та admin_panel.html для інтерфейсу адміністратора. Наприклад, файл chat.html активно використовує бібліотеку HTMX для забезпечення динамічного оновлення повідомлень (Рисунок 3.5 - Приклад HTML-шаблону chat.html):

```
<div id="messages" hx-get="/chat/{{ chat.id }}/messages" hx-trigger="every 1s">
    {% for message in messages %}
        <p>{{ message.sender.username }}: {{ message.content }}</p>
        {% if message.file %}
            <a href="{{ message.file.url }}">Завантажити файл</a>
        {% endif %}
    {% endfor %}
</div>
<form hx-post="/chat/{{ chat.id }}/send" hx-target="#messages">
    <input type="text" name="message">
    <button type="submit">Надіслати</button>
</form>
```

Рисунок 3.5 - Приклад HTML-шаблону chat.html з HTMX

Tailwind CSS використовується для стилізації інтерфейсу, забезпечуючи сучасний та адаптивний дизайн. Завдяки Tailwind CSS, чати відображаються як візуально привабливі картки, що містять назву та останнє повідомлення. Адміністративна панель, у свою чергу, має чітко структуровані таблиці з кнопками для виконання різних дій. Приклад стилізації кнопки за допомогою Tailwind CSS-класів (Рисунок 3.6 - Приклад стилізації кнопки за допомогою Tailwind CSS-класів):

```
<button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounde
    Видалити
</button>
```

Рисунок 3.6 - Приклад стилізації кнопки за допомогою Tailwind CSS-класів

HTMX відіграє ключову роль у забезпеченні динамічних оновлень, дозволяючи оновлювати вміст чатів без повного перезавантаження сторінки. Наприклад, після надсилання нового повідомлення, воно з'являється в чаті в реальному часі за допомогою AJAX-подібного запиту, а відповідь сервера замінює вміст відповідного елемента div з ідентифікатором messages. Це значно покращує користувацький досвід, роблячи взаємодію з месенджером більш плавною та швидкою.

Alpine.js додає легку клієнтську інтерактивність до інтерфейсу. Він використовується для реалізації таких елементів, як випадаючі меню для вибору чатів або модальні вікна для редагування профілю чи видалення повідомлень. Легкість Alpine.js, порівняно з більш об'ємними JavaScript-фреймворками, забезпечує швидке завантаження сторінок та високу швидкодію. Приклад використання Alpine.js для випадаючого меню чатів (Рисунок 3.7 - Приклад використання Alpine.js для випадаючого меню чатів):

```
<div x-data="{ open: false }">
    <button @click="open = !open">Чати</button>
    <ul x-show="open" class="absolute bg-white shadow-md">
        {% for chat in user.chat_groups.all %}
            <li><a href="/chat/{{ chat.id }}">{{ chat.name }}</a></li>
        {% endfor %}
    </ul>
</div>
```

Рисунок 3.7 - Приклад використання Alpine.js для випадаючого меню чатів

РОЗДІЛ IV. ОЦІНКА ТА ТЕСТУВАННЯ РОБОТИ МЕСЕНДЖЕРА

Тестування програмного продукту є невід'ємним етапом його життєвого циклу, спрямованим на забезпечення надійності, стабільності та відповідності встановленим вимогам. Для месенджера, розробленого для роботи в локальній мережі з інтегрованою адміністративною панеллю, тестування набуває особливого значення. Це зумовлено необхідністю гарантувати безпечне та безперебійне спілкування, швидку доставку повідомлень та ефективне керування системою в ізольованому мережевому середовищі. Основні завдання тестування включали виявлення та усунення дефектів, підвищення загальної якості продукту, верифікацію відповідності програмного забезпечення сформульованим вимогам, а також оптимізацію продуктивності системи.

Тестування функціональних можливостей та продуктивності месенджера проводилося в модельованій локальній мережі. Серверна частина системи розгорталася на віртуальній машині з операційною системою Ubuntu Server, тоді як клієнтські взаємодії симулювалися з використанням пристроїв під керуванням Windows та Linux. Об'єктами перевірки були ключові компоненти, такі як серверна логіка на Django з Channels та SQLite, а також клієнтський інтерфейс, реалізований за допомогою Tailwind CSS, HTMX та Alpine.js. Особлива увага приділялася перевірці ключових функцій, зокрема процесів реєстрації та авторизації, обміну повідомленнями, роботі з файлами, а також можливостям адміністративного керування.

4.1. Тестування користувацького функціоналу

Тестування користувацького функціоналу охоплювало основні сценарії взаємодії кінцевих користувачів із системою, від реєстрації до обміну повідомленнями та управління власним профілем.

4.1.1. Реєстрація та авторизація користувачів

Процес реєстрації нового користувача розпочинається з вибору відповідної вкладки в меню, після чого з'являється форма для введення облікових даних. Користувач заповнює поля логіна, електронної пошти, пароля та його підтвердження (Рисунок 4.1 - Форма реєстрації користувача):

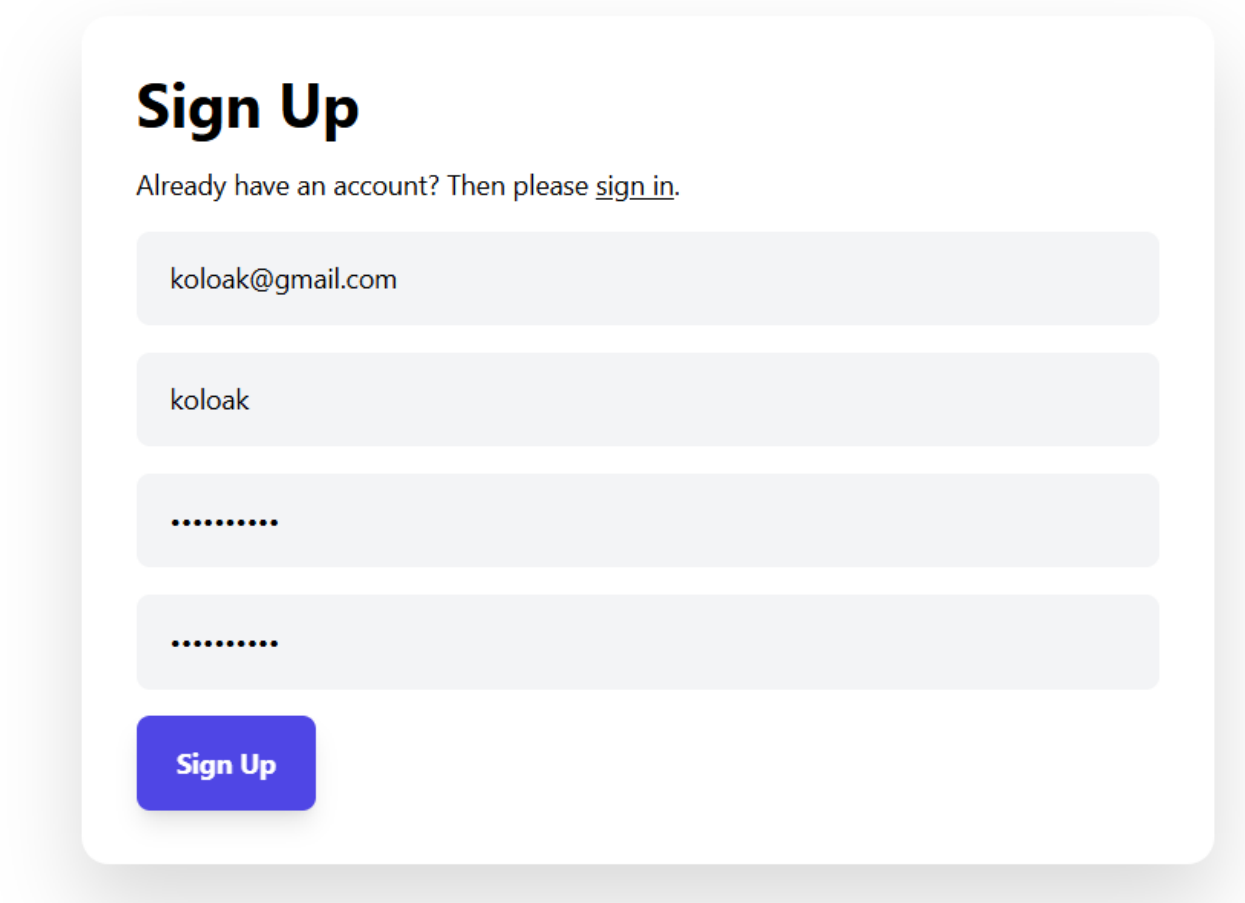

Рисунок 4.1 - Форма реєстрації користувача

Якщо дані введено некоректно (наприклад, email невалідний або паролі не збігаються), з’являється повідомлення про помилку (Рисунок 4.2 - Повідомлення про помилку під час реєстрації):

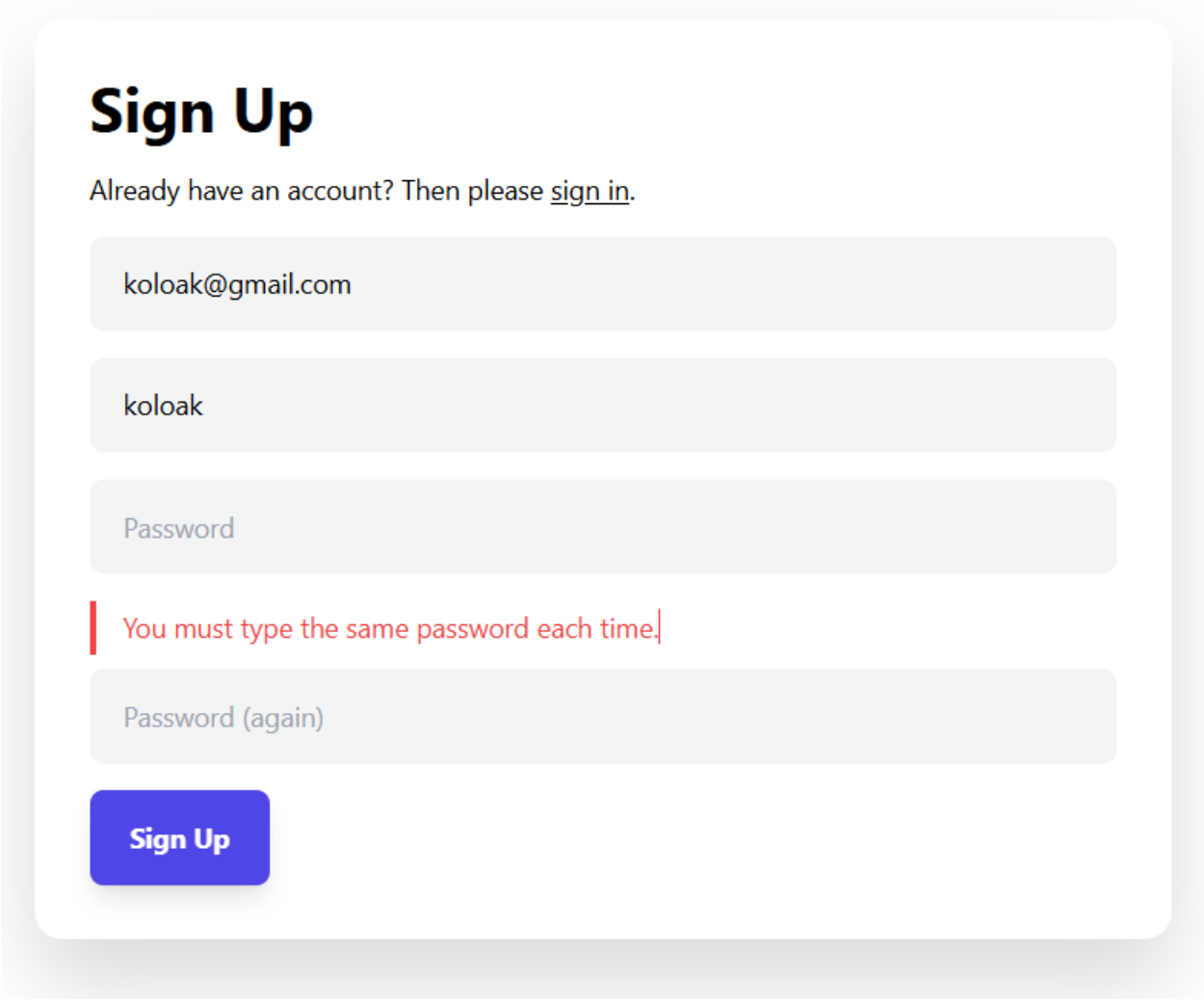

Рисунок 4.2 - Повідомлення про помилку під час реєстрації

Після успішної реєстрації та авторизації, користувач отримує доступ до свого особистого профілю, де може ознайомитися з можливостями взаємодії з іншими учасниками. Центральним елементом комунікації в системі є головний чат, який стає доступним одразу після входу. Цей чат функціонує як єдиний простір, де всі зареєстровані користувачі можуть вільно та оперативно обмінюватися повідомленнями в режимі реального часу (Рисунок 4.3 - Інтерфейс головного чату):

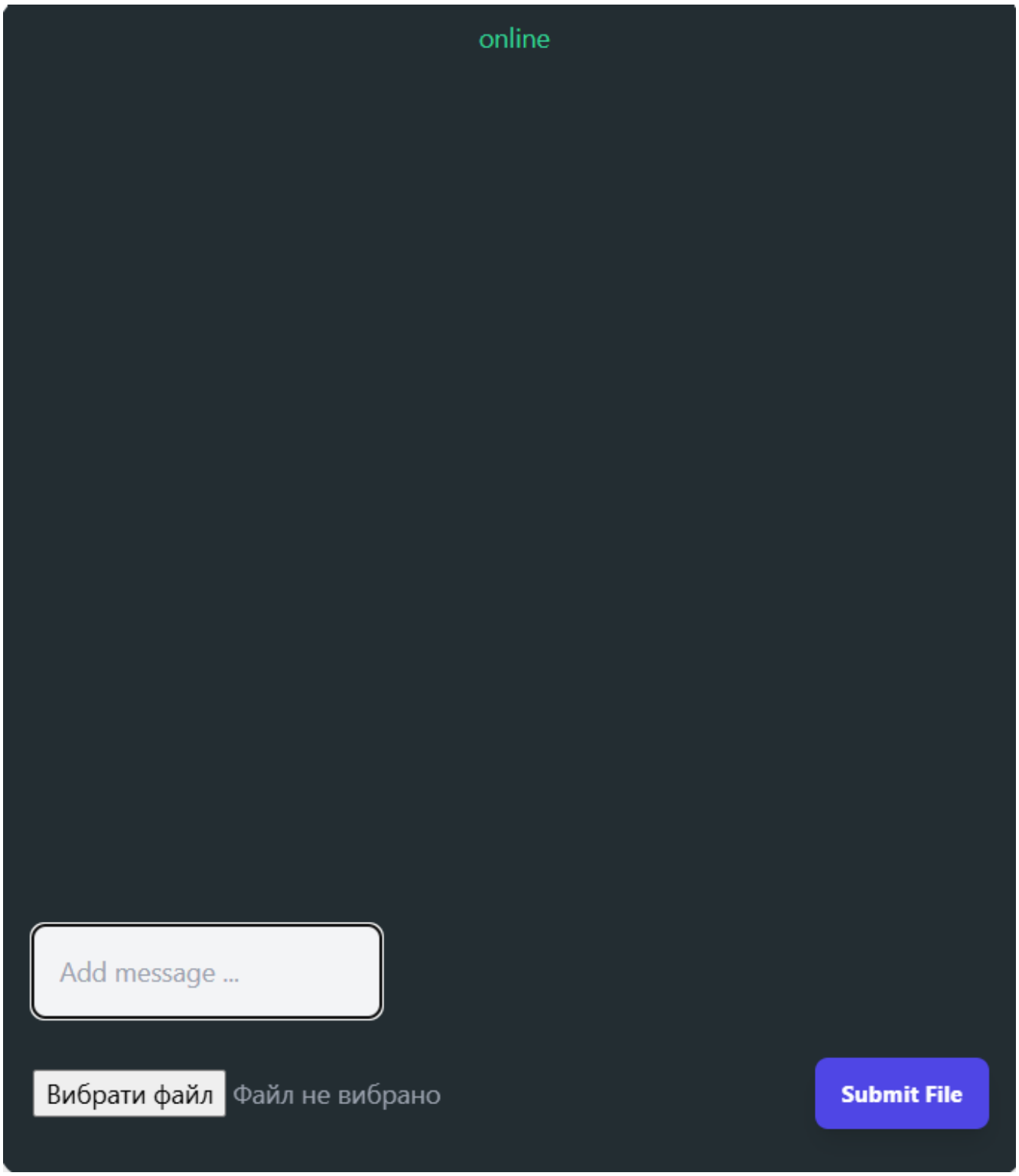

Рисунок 4.3 - Інтерфейс головного чату

4.1.2. Обмін повідомленнями та файлами у головному чаті

Інтерфейс головного чату спроєктований з акцентом на простоту та функціональність, забезпечуючи інтуїтивно зрозуміле середовище для комунікації.

У верхній частині екрана відображається статус "online", що свідчить про активне підключення користувача до мережі та готовність до спілкування. Основна частина екрана відведена під поле для відображення історії повідомлень, де користувачі можуть переглядати всю попередню переписку з моменту приєднання до чату (Рисунок 4.4 - Результат обміну повідомленнями):

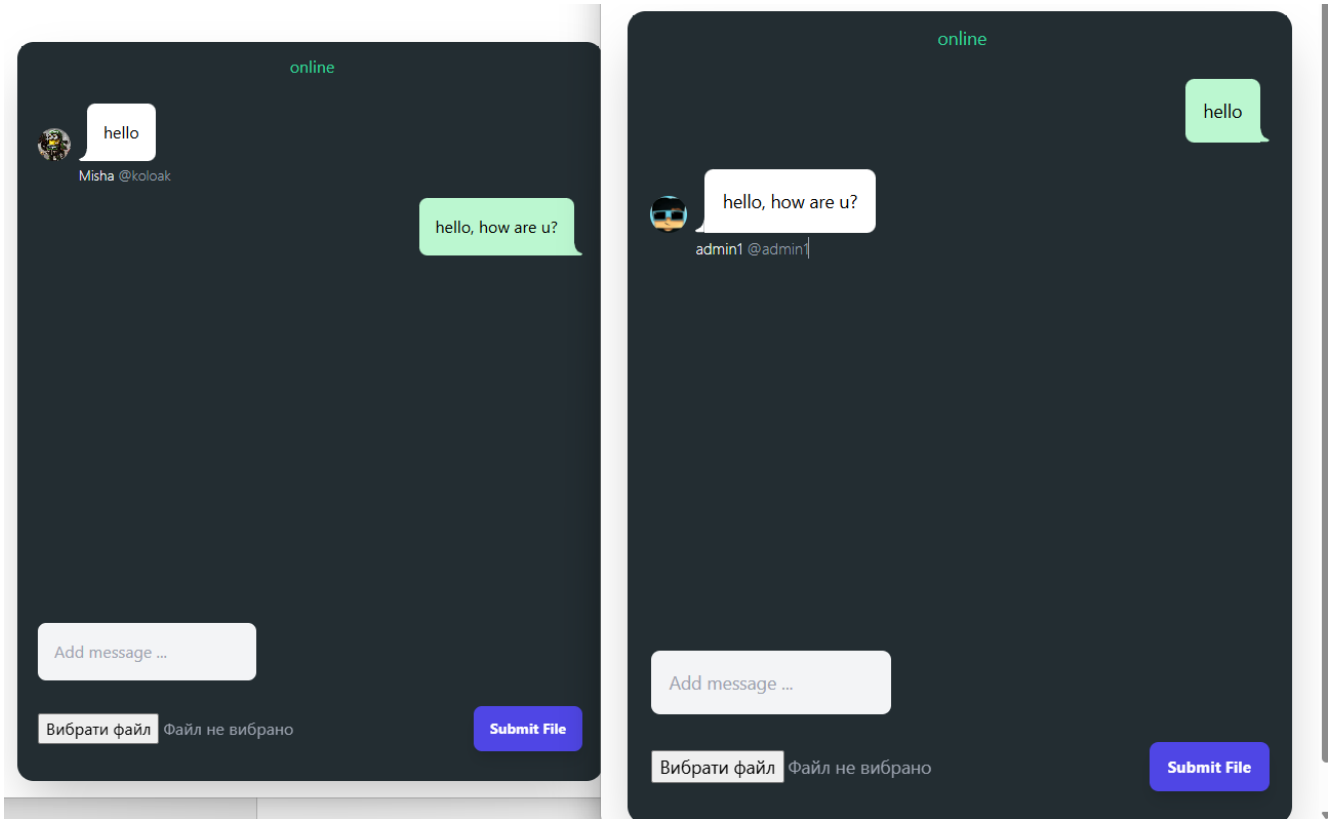

Рисунок 4.4 - Результат обміну повідомленнями

Для надсилання повідомлень передбачено поле вводу з підказкою "Add message...", де користувач вводить текст і надсилає його натисканням Enter або кнопкою, після чого повідомлення одразу з’являється в чаті. Також доступна функція прикріплення файлів через кнопку "Виберіть файл" з індикатором "Файл не выбран", що дозволяє обмінюватися документами та зображеннями. Після вибору файлу його назва відображається поруч, а відправка виконується кнопкою "Submit File" (Рисунок 4.5 - Результат обміну файлами):

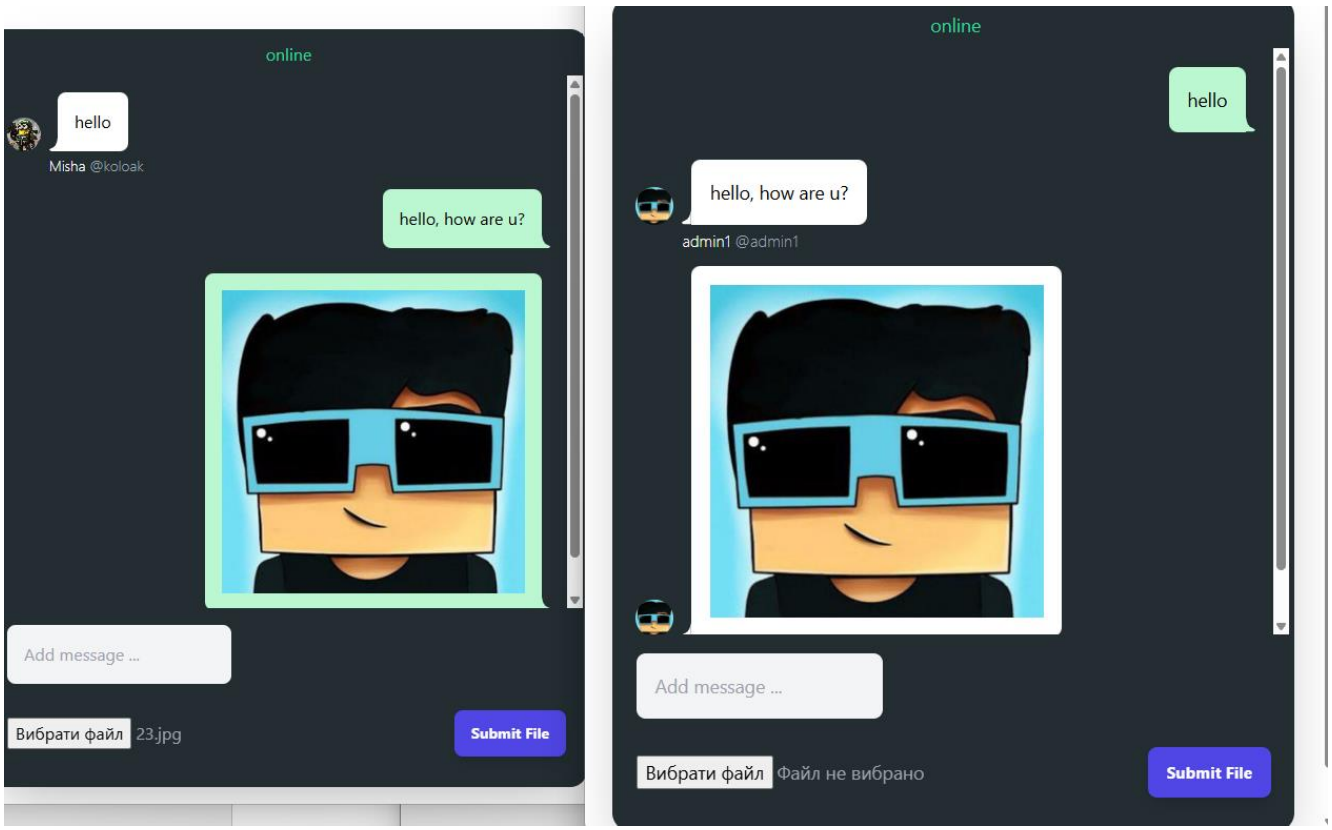

Рисунок 4.5 - Результат обміну файлами

4.1.3. Управління обліковим записом через меню користувача

Для зручного доступу до основних функцій управління обліковим записом у верхній частині інтерфейсу застосунку передбачено розширене меню користувача. Воно активується при натисканні на ім'я поточного користувача і розкриває список доступних вкладок для вибору (Рисунок 4.6 - Меню користувача):

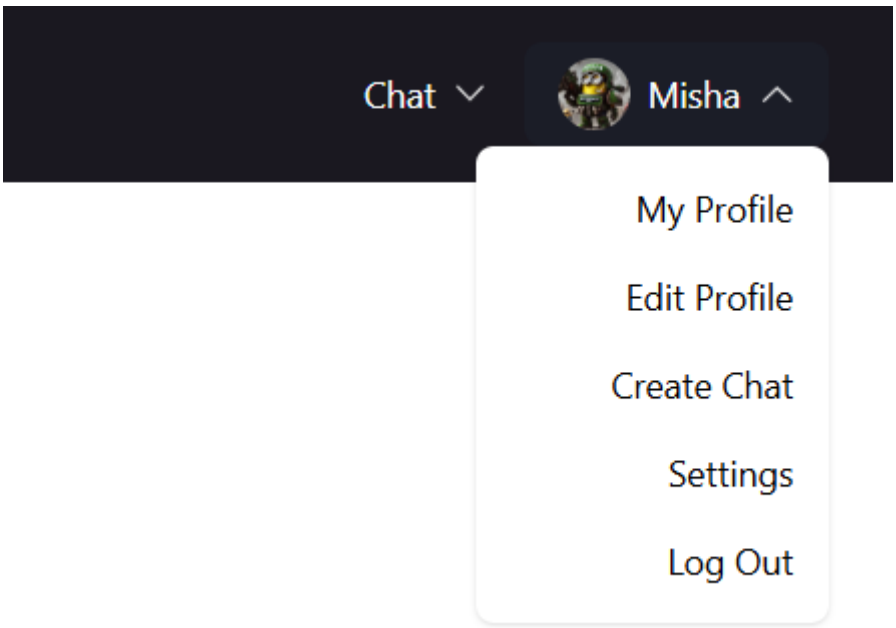

Рисунок 4.6 - Меню користувача

Це меню є навігаційним центром для швидкого переходу до ключових розділів, що стосуються управління профілем та налаштувань. Вкладка "My Profile" (Мій Профіль) дозволяє користувачеві миттєво переглянути свій особистий профіль, де відображаються ім'я користувача, фото профілю та інша інформація. Ця сторінка призначена виключно для перегляду інформації, без можливості її безпосереднього редагування (Рисунок 4.7 - Сторінка "Мій Профіль"):

Рисунок 4.7 - Сторінка "Мій Профіль"

Перехід на вкладку "Edit Profile" (Редагувати Профіль) відкриває можливості для внесення змін в особисті дані користувача. На цій сторінці користувач може змінити фотографію профілю, оновити ім'я користувача, а також додати або відредагувати додаткову інформацію. Збереження всіх внесених змін здійснюється натисканням кнопки "Submit", тоді як кнопка "Cancel" дозволяє скасувати зміни та повернутися до попереднього стану (Рисунок 4.8 - Сторінка "Редагувати Профіль"):

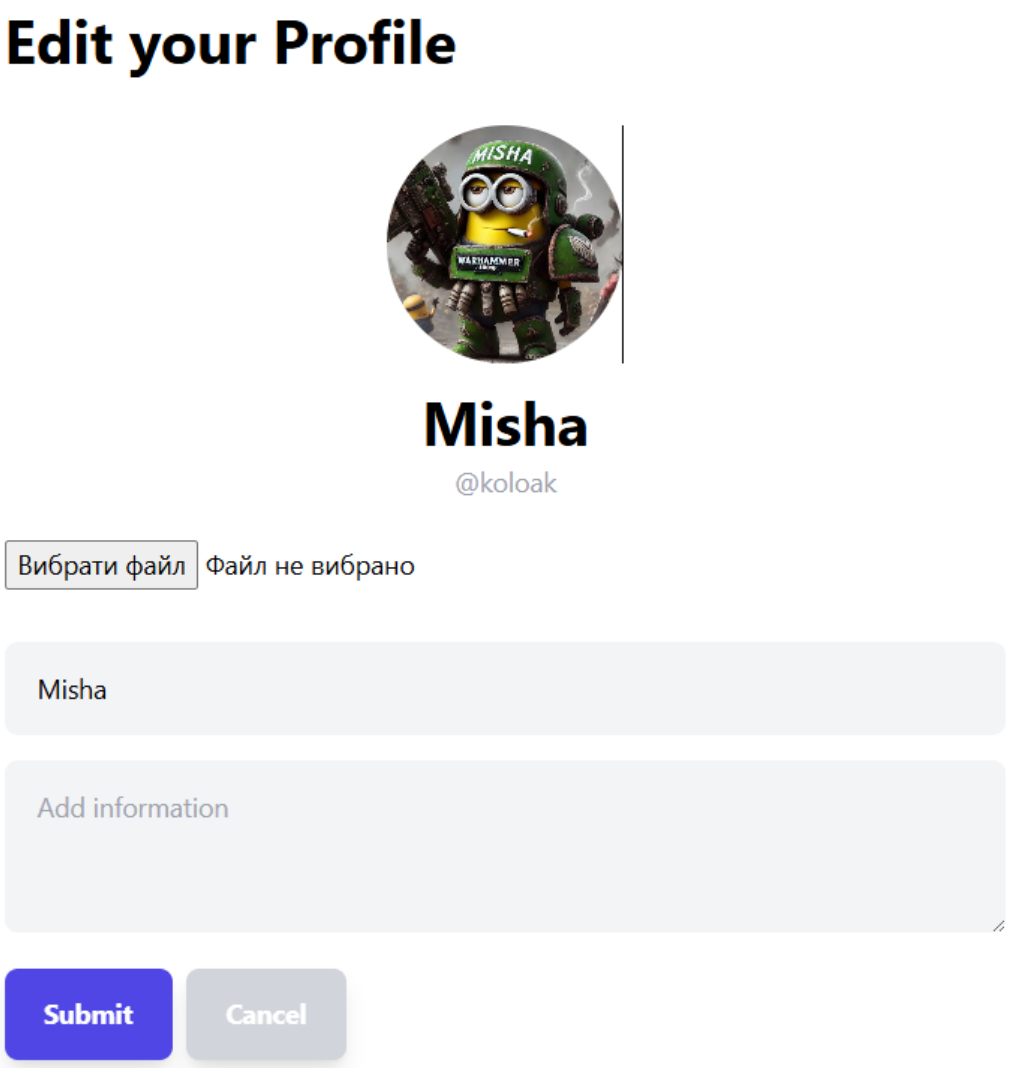

Рисунок 4.8 - Сторінка "Редагувати Профіль"

Вкладка "Settings" (Налаштування) надає доступ до конфіденційних налаштувань облікового запису, що дозволяють керувати ключовими параметрами безпеки та ідентифікації. Тут користувач може переглянути та, за потреби, змінити свою електронну адресу та ім'я користувача. Найбільш відповідальною функцією є

можливість видалити обліковий запис, про незворотність якої система чітко попереджає (Рисунок 4.9 - Сторінка "Налаштування облікового запису"):

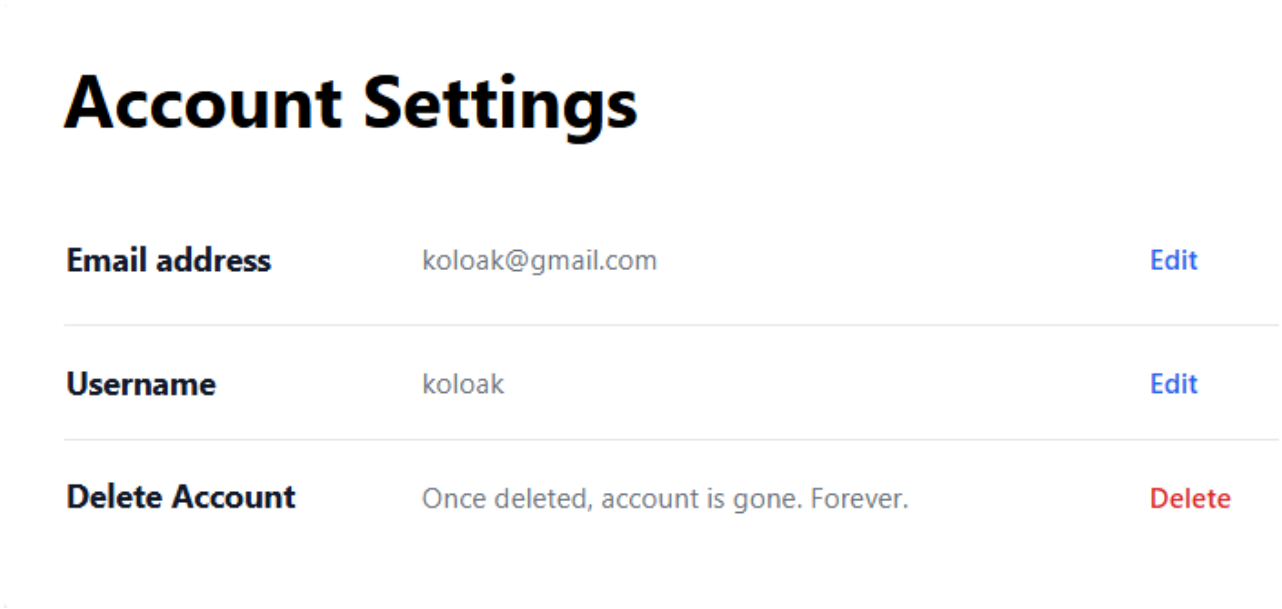

Рисунок 4.9 - Сторінка "Налаштування облікового запису"

Функція "Log Out" (Вийти) забезпечує безпечне завершення сесії користувача в застосунку. При її виборі з'являється діалогове вікно з питанням-підтвердженням, що запобігає випадковому виходу з системи (Рисунок 4.10 - Підтвердження виходу із системи):

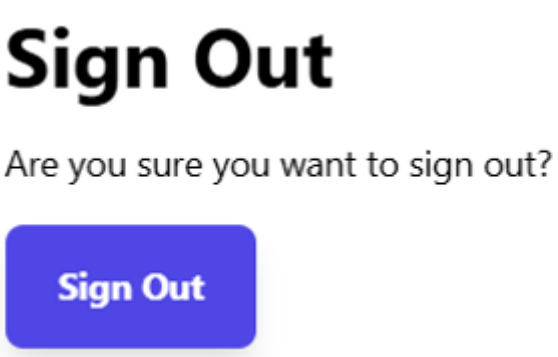

Рисунок 4.10 - Підтвердження виходу із системи

4.1.4. Створення та управління груповими чатами

Процес створення нового групового чату ініціюється користувачем через випадаюче меню профіля, розташоване у верхній частині інтерфейсу.

З цього меню користувач може обрати існуючий "Public Chat" або раніше створений груповий чат, такий як "Rt-chat". Для ініціації нового групового чату, користувачеві слід обрати опцію "Create Chat". Після цього з'являється спеціальне вікно для його конфігурації, де користувачеві пропонується ввести назву для нового групового чату. Після введення назви, натискання кнопки "Create Group Chat" завершує процес створення (Рисунок 4.11 - Приєднання користувачів до чату за URL-посиланням):

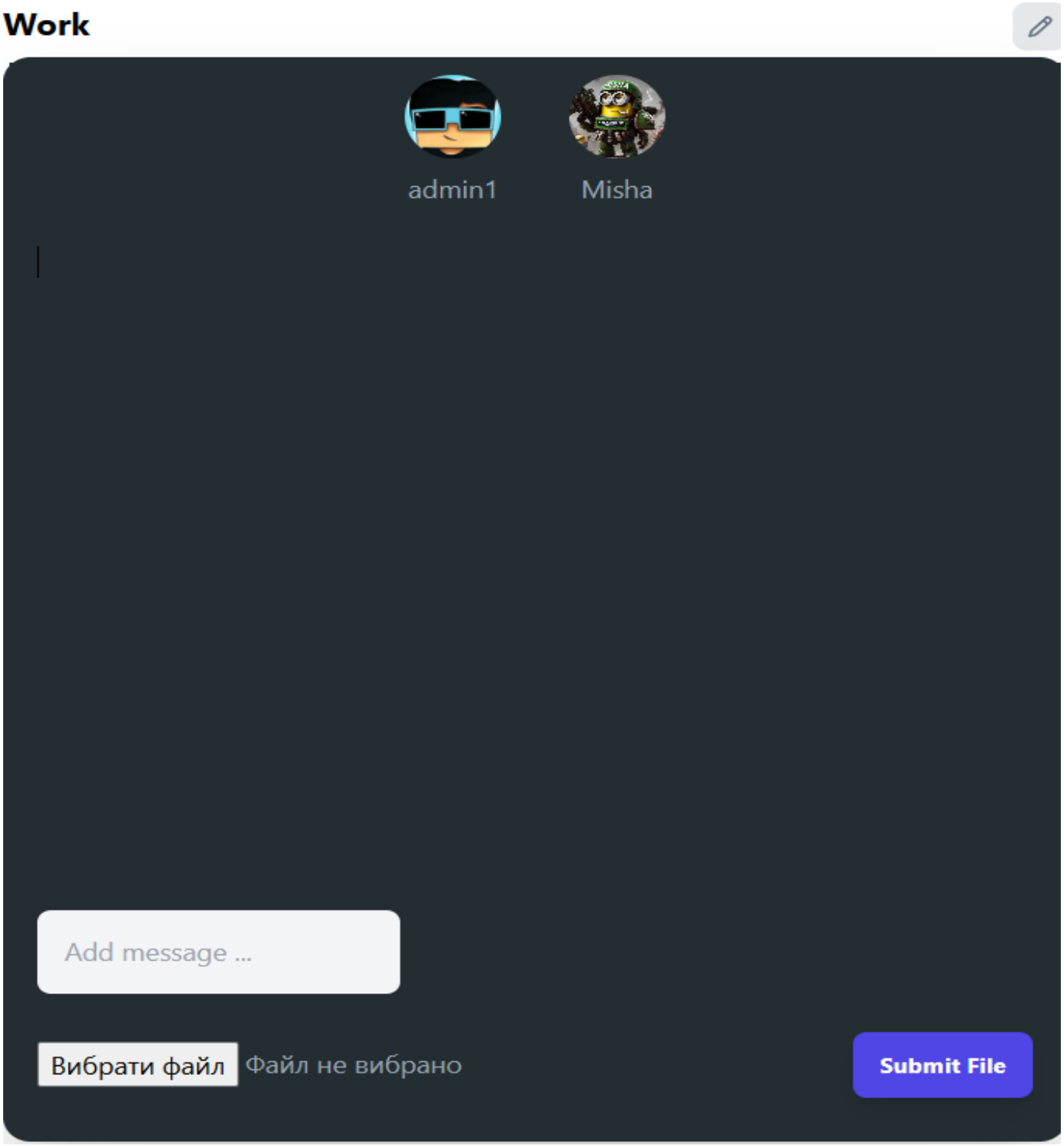

Рисунок 4.11 - Приєднання користувачів до чату за URL-посиланням

Особливістю групових чатів є можливість приєднання до них за допомогою унікального URL-посилання з токеном, яке генерується при створенні чату. Користувач-адміністратор або творець чату може скопіювати це посилання та надати його іншим користувачам. Коли інший користувач використовує це

посилання, він автоматично приєднується до чату, а його профіль та ім'я з'являються у вікні чату поруч з іншими учасниками.

Після створення, новий чат, наприклад "Work", з'являється в загальному списку доступних чатів, що дозволяє легко переключатися між ними. Інтерфейс новоствореного чату відповідає стандартному для обміну повідомленнями, включаючи поле для введення тексту, опцію для прикріплення файлів та кнопку для їх відправки. У верхній частині чату відображається його назва та піктограма олівця, що вказує на можливість редагування параметрів чату (Рисунок 4.12 - Список групових чатів):

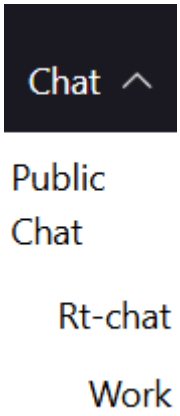

Рисунок 4.12 - Список групових чатів

Для ефективного управління груповими чатами, зокрема контролю над учасниками та загальними налаштуваннями, передбачена адміністративна панель чату. Доступ до неї здійснюється через піктограму олівця поруч із назвою чату. Панель "Edit Chat" надає адміністраторам можливість змінювати назву чату, управляти учасниками, видаляючи їх з чату за потреби, а також повністю видалити груповий чат (Рисунок 4.13 - Адміністративна панель чату "Edit Chat"):

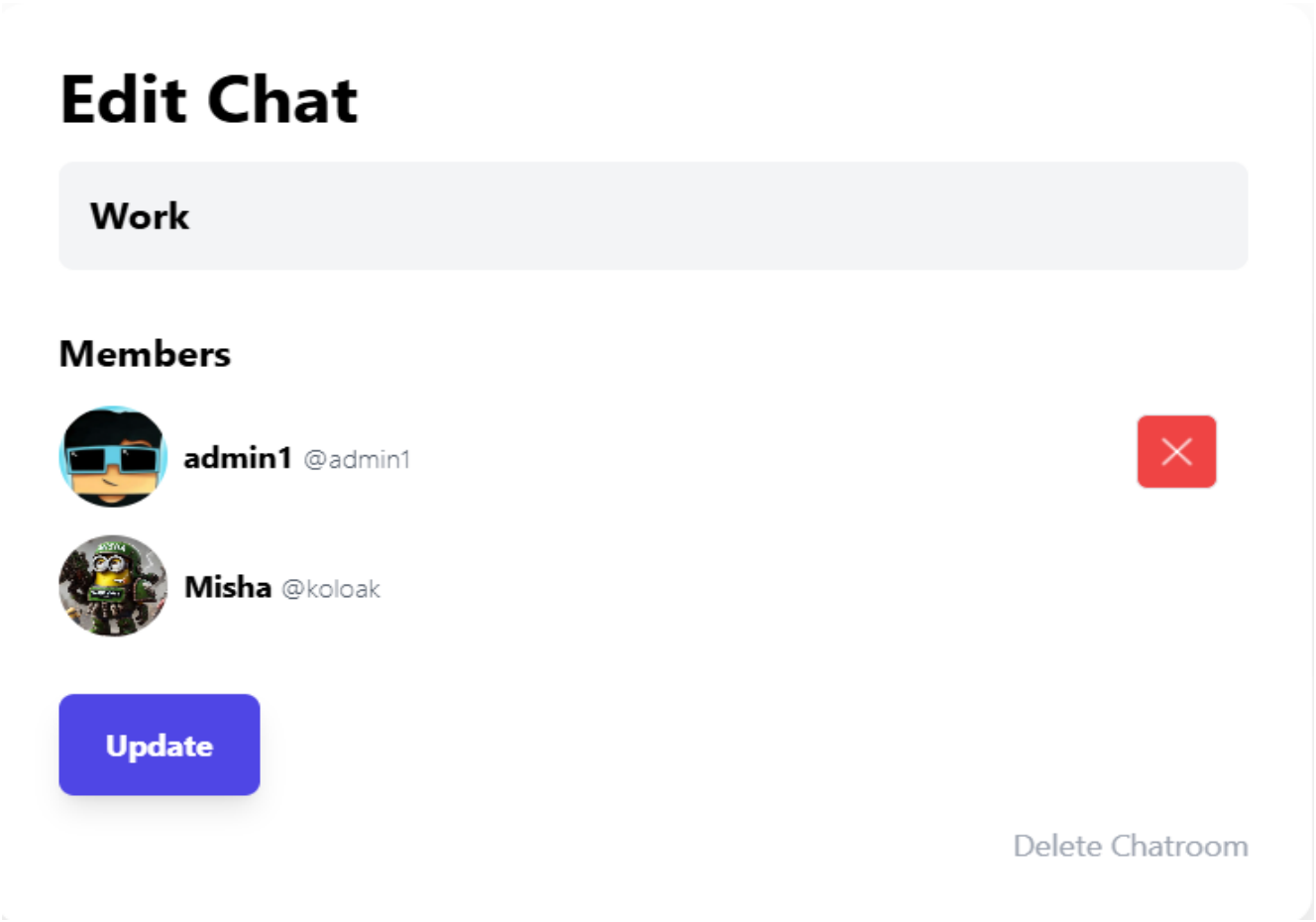

Рисунок 4.13 - Адміністративна панель чату "Edit Chat"

4.1.5. Обмін приватними повідомленнями

В рамках взаємодії в груповому чаті, користувачам також надається можливість ініціювати приватні повідомлення з будь-яким іншим учасником, присутнім у цьому чаті. Для цього достатньо обрати потрібного користувача у списку учасників чату. Після вибору користувача, система автоматично відкриває окреме вікно приватного чату між двома співрозмовниками (Рисунок 4.14 - Приватний чат між користувачами та кнопка розпочати приватний чат):

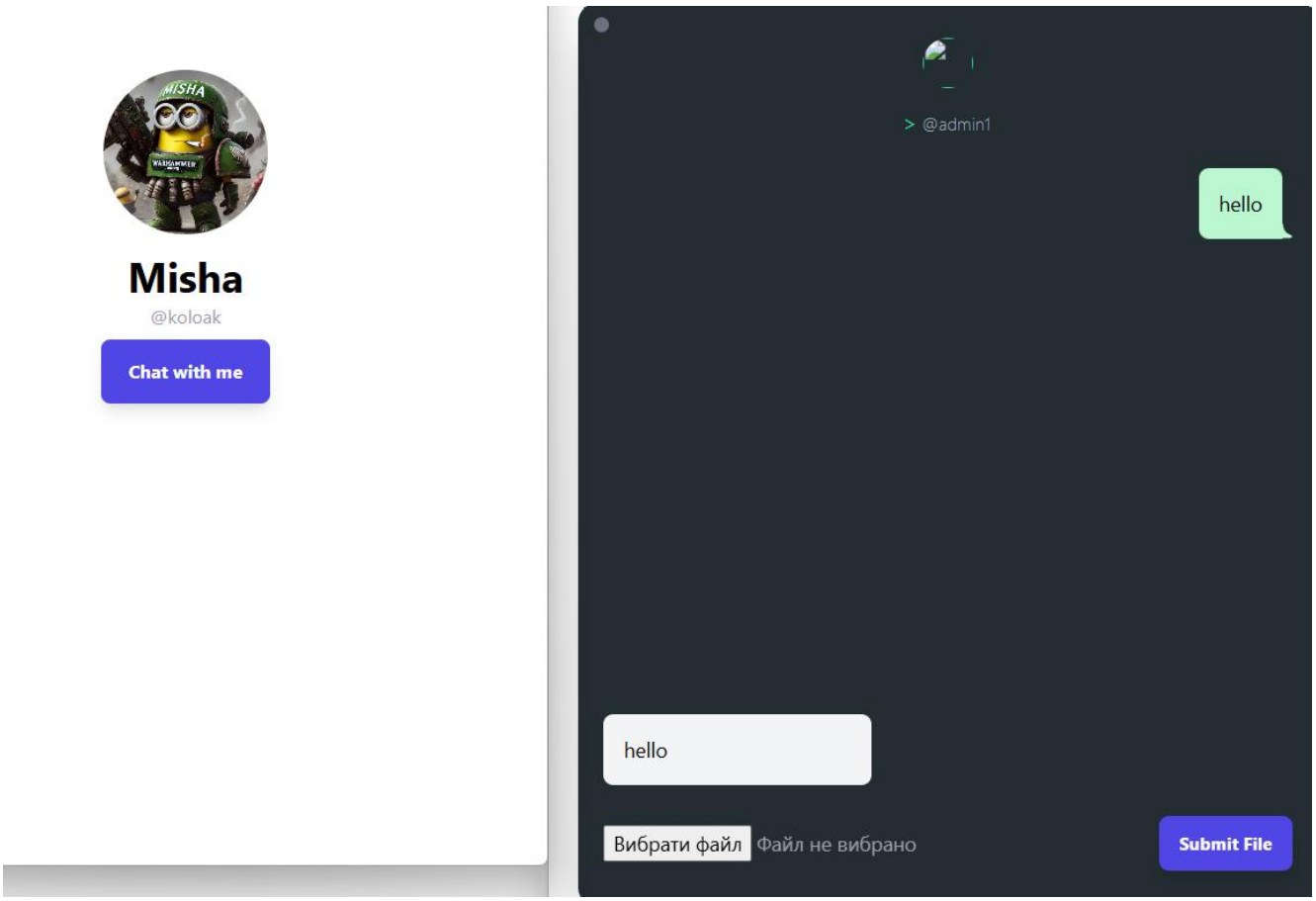

Рисунок 4.14 - Приватний чат між користувачами та кнопка розпочати приватний чат

4.2. Тестування адміністративного функціоналу

Для забезпечення стабільної роботи, безпеки та ефективного функціонування програми, розроблено спеціальний розділ адміністративного управління. Він надає розширені можливості для контролю та моніторингу ключових аспектів системи, включаючи чати та користувачів. Доступ до цих функцій мають лише користувачі з відповідними адміністративними правами.

4.2.1. Перегляд та управління групами чатів

Адміністративна панель дозволяє системним адміністраторам переглядати та керувати всіма створеними груповими чатами. На екрані представлений повний список усіх групових чатів, які існують у системі, включаючи чати з унікальними

ідентифікаторами та загальнодоступні чати. Адміністратор має можливість вибрати один або кілька чатів, встановивши відповідні позначки, та застосувати до них певні дії через випадаюче меню "Action" (Рисунок 4.15 - Управління груповими чатами):

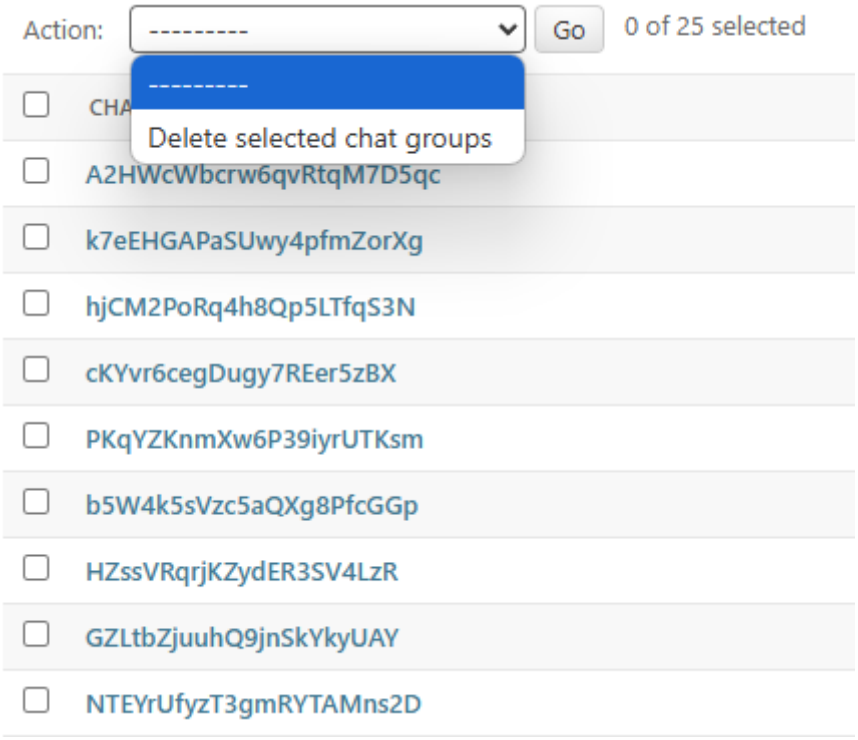

Рисунок 4.15 - Управління груповими чатами

4.2.2. Моніторинг та видалення групових повідомлень

Адміністративна панель надає можливість детального моніторингу всіх повідомлень, що надсилаються в групових чатах, а також функції їх видалення. На цьому екрані відображається список усіх групових повідомлень, включаючи ім'я відправника та текст повідомлення. Адміністратор може переглядати історію повідомлень та, за потреби, обирати небажані повідомлення для видалення. Випадаюче меню "Action" дозволяє вибрати дію "Delete selected group messages" (Рисунок 4.16 - Моніторинг та видалення групових повідомлень):

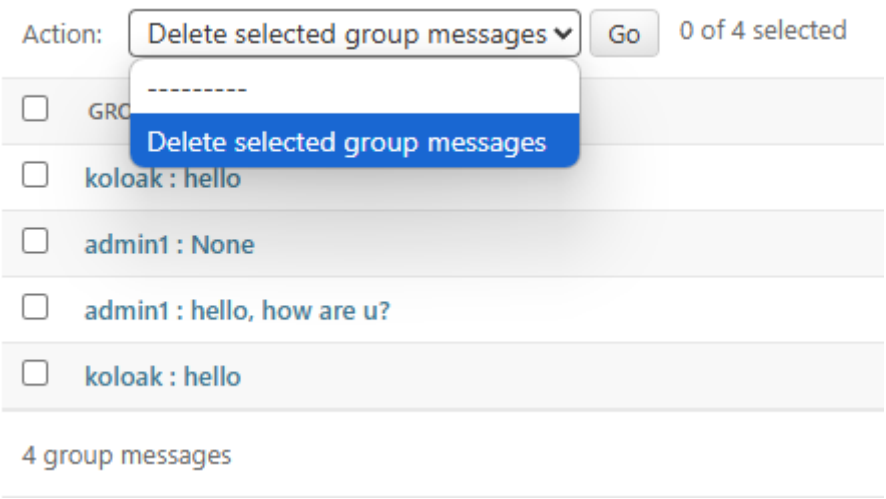

Рисунок 4.16 - Моніторинг та видалення групових повідомлень

4.2.3. Управління користувачами системи

Одним з ключових аспектів адміністративного управління є повний контроль над користувачами застосунку. Ця вкладка адміністративної панелі відображає детальний список усіх зареєстрованих користувачів системи. Для кожного користувача вказано його ім'я користувача (USERNAME) та адресу електронної пошти (EMAIL ADDRESS). Адміністратор має можливість переглядати всіх користувачів, а також виконувати операції з їх управління. Випадаюче меню "Action" дозволяє адміністратору вибрати дію "Delete selected users" (Рисунок 4.17 - Управління користувачами системи):

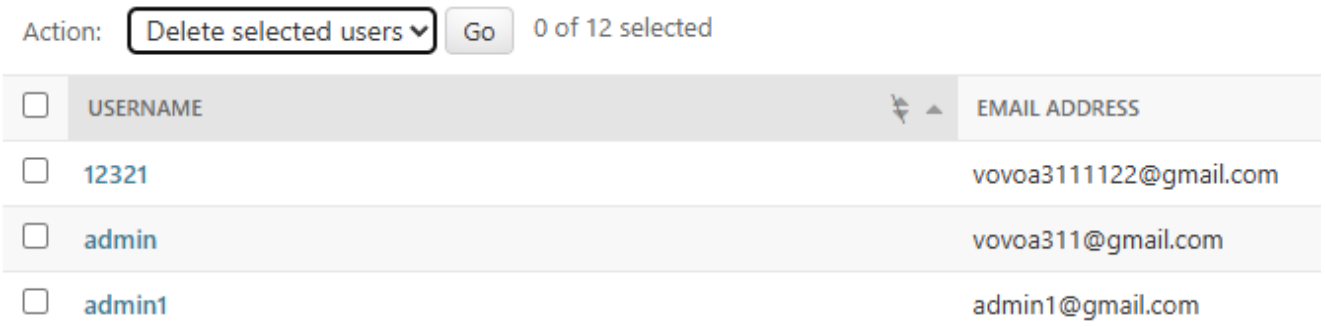

Рисунок 4.17 - Управління користувачами системи

4.2.4. Моніторинг онлайн-статусу користувачів у чатах

Для постійного моніторингу активності користувачів в реальному часі, адміністративна панель надає функцію відстеження їхнього онлайн-статусу в чатах. На цьому екрані представлені два ключові списки: "Users online" (Користувачі онлайн) та "Members" (Учасники). Список "Users online" відображає всіх користувачів, які наразі перебувають в онлайн-режимі в системі, дозволяючи адміністратору бачити, хто з користувачів активний у даний момент. Список "Members" відображає всіх зареєстрованих учасників системи, або ж певних чатів (залежно від контексту). Обидва списки містять імена користувачів. Ця функція є незамінною для адміністраторів, що дозволяє їм оперативно оцінювати завантаженість системи, відстежувати активність користувачів та вирішувати потенційні проблеми з підключенням (Рисунок 4.18 - Моніторинг онлайн-статусу користувачів у чатах):

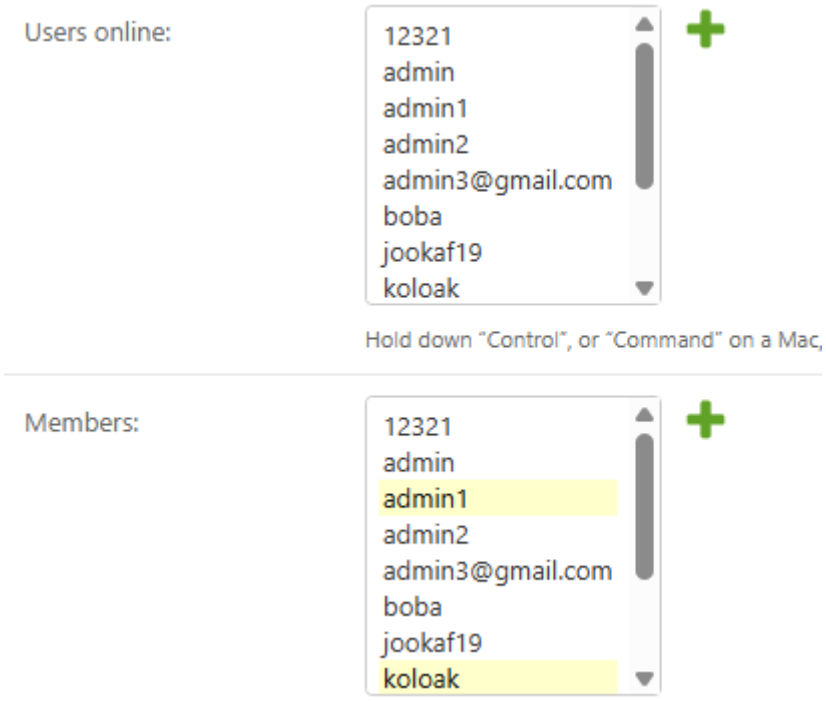

Рисунок 4.18 - Моніторинг онлайн-статусу користувачів у чатах

ВИСНОВКИ

У процесі виконання дипломної роботи було успішно розроблено та реалізовано програмне забезпечення для моніторингу локальної мережі з функціями месенджера та адміністративною панеллю. Комплексний аналіз сучасних комунікаційних платформ підтвердив актуальність створення автономного рішення для ізольованих мережевих середовищ, що стало основою для формування архітектури та функціональних вимог до системи.

Розроблений веб-застосунок базується на надійному технологічному стеку, що включає Django для серверної логіки, Django Channels та WebSocket для реалізації функціоналу чатів у реальному часі, а також Tailwind CSS, HTMX та Alpine.js для створення адаптивного та зручного користувацького інтерфейсу. Система забезпечує повний цикл аутентифікації користувачів, обмін текстовими повідомленнями та файлами у групових і приватних чатах, а також розширені можливості адміністративного керування.

Проведене тестування підтвердило стабільність та ефективність роботи месенджера в локальній мережі, включаючи швидку доставку повідомлень та коректну обробку файлів. Функціонал адміністративної панелі дозволяє здійснювати ефективний контроль над комунікаціями та обліковими записами користувачів. Виявлені обмеження продуктивності при високих навантаженнях слугують основою для подальших вдосконалень.

Перспективи розвитку проєкту передбачають інтеграцію наскрізного шифрування для підвищення рівня безпеки даних, масштабування системи за рахунок використання Redis та PostgreSQL для підтримки значної кількості користувачів, а також розробку мобільного додатку для розширення доступності. Реалізація цих напрямів дозволить створити багатофункціональне та високопродуктивне рішення, що відповідатиме зростаючим вимогам до безпечної та ефективної комунікації в локальних мережах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. У Гончаренко О. М. Основи програмування на Python та розробка веб-застосунків. Київ: Ліра-К, 2020. 412 с.
2. Поляков А. В., Мельник В. В. Безпека інформаційних систем і мереж. Харків: ХНУРЕ, 2018. 350 с.
3. Офіційна документація Django: Панель адміністратора. URL: https://docs.djangoproject.com/en/stable/ref/contrib/admin/.
4. Клочко О. О. Захист інформації в комп'ютерних системах та мережах. Київ: Юрінком Інтер, 2017. 450 с.
5. Іванов П. І., Сидоренко А. В. Архітектура розподілених систем реального часу. Київ: Наукова думка, 2021. 380 с.
6. Ковальчук В. М. Проблеми та перспективи використання месенджерів у корпоративних мережах. Вісник Національного університету "Львівська політехніка". Серія: Комп'ютерні науки. – 2019. – № 923. – С. 120-125.
7. Шевченко О. В., Петренко І. С. Розробка веб-додатків з використанням Django: підручник. Київ: Видавничий дім "Курс", 2022. 340 с.
8. Коваленко В. І. Мережеві технології та протоколи: навчальний посібник. Львів: Новий Світ-2000, 2021. 280 с.
9. Сайт Django Channels. URL: https://channels.readthedocs.io/en/stable/.
10. Документація Tailwind CSS. URL: https://tailwindcss.com/docs.
11. Офіційний сайт HTMX. URL: https://htmx.org/.
12. Alpine.js Documentation. URL: https://alpinejs.dev/.
13. SQLite Official Website. URL: https://www.sqlite.org/.

ДОДАТОК А
ЛІСТИНГ ПРОГРАМИ МЕСЕНДЖЕРА ДЛЯ ЛОКАЛЬНОЇ МЕРЕЖІ

Програма месенджера для локальної мережі написана мовою Python з використанням фреймворку Django та бібліотеки Django Channels для забезпечення асинхронного обміну повідомленнями через протокол WebSocket.

Файл asgi.py

```
import os


from django.core.asgi import get_asgi_application

from channels.routing import ProtocolTypeRouter, URLRouter

from channels.security.websocket import AllowedHostsOriginValidator

from channels.auth import AuthMiddlewareStack


os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'a_core.settings')


django_asgi_app = get_asgi_application()


from a_rtchat import routing


application = ProtocolTypeRouter({
    "http": django_asgi_app,
    "websocket": AllowedHostsOriginValidator(
        AuthMiddlewareStack(URLRouter(routing.websocket_urlpatterns))
    ),
})
```

Файл chat.html

```
{% extends 'layouts/blank.html' %}


{% block content %}


<wrapper class="block max-w-2xl mx-auto my-10 px-6">
    {% if chat_group.groupchat_name %}
    <div class="flex justify-between">
    <h2>{{ chat_group.groupchat_name }}</h2>
    {% if user == chat_group.admin %}
    <a href="{% url 'edit-chatroom' chat_group.group_name %}">
        <div class="p-2 bg-gray-200 hover:bg-blue-600 rounded-lg group">
            <svg class="fill-gray-500 group-hover:fill-white" width="16" height="16">
                <path d="M11.013 1.427a1.75 1.75 0 0 1 2.474 0l1.086 1.086a1.75 1.75 0 0 1 0 2.474l-8.61 8.61c-.21.21-.47.364-.756.445l-3.251.93a.75.75 0 0 1-.927-.928l.929-3.25c.081-.286.235-.547.445-.758l8.61-8.61Zm.176 4.823L9.75 4.81l-6.286 6.287a.253.253 0 0 0-.064.108l-.558 1.953 1.953-.558a.253.253 0 0 0 .108-.064Zm1.238-3.763a.25.25 0 0 0-.354 0L10.811 3.75l1.439 1.44 1.263-1.263a.25.25 0 0 0 0-.354Z"></path>
            </svg>
        </div>
    </a>
    {% endif %}
    </div>
    {% endif %}
    <div id="chat_window" class="h-[45rem] flex flex-col bg-[#232d32] rounded-2xl shadow-2xl relative p-1">
        <div class="flex justify-center text-emerald-400 bg-[#232d32] p-2 sticky top-0 z-10">
            {% if other_user %}
            <div id="online-icon" class="gray-dot absolute top-2 left-2"></div>
            <a href ="{% url 'profile' other_user.username %}">
                <div class="flex items-center gap-2 p-4 sticky top-0 z-10">
```

```
                <img class="w-10 h-10 rounded-full object-cover" src=="{{ other_user.profile.avatar }}"
/>

                </div>

                   <span class="font-bold text-white">{{ other.user.profile.name }}</span>>

                   <span class="text-sm font-light text-gray-400">@{{ other_user.username }}</span>

                <div>

              </div>


          </a>

          {% elif chat_group.groupchat_name %}

          <ul class="flex gap-4">

             {% for member in chat_group.members.all %}

             <li>

                <a href="{% url 'profile' member.username %}" class="flex flex-col text-gray-400 items-
center justify-center w-20 gap-2">

                   <img src="{{ member.profile.avatar}}" class="w-14 h-14 rounded-full object-cover" />

                   {{ member.profile.name|slice:":10" }}

                </a>

             </li>

             {% endfor %}

          </ul>

          {% else %}

          <div id="online-icon"></div>

          <span id="online-count" class="pr-1"></span>online

          {% endif %}

       </div>

       <div id='chat_container' class="overflow-y-auto grow">

          <ul id='chat_messages' class="flex flex-col justify-end gap-2 p-4">

             {% for message in chat_messages reversed %}
```

```
{% include 'a_rtchat/chat_message.html' %}
{% if message.author == user %}
<li class="flex justify-end mb-4">
    <div class="bg-green-200 rounded-l-lg rounded-tr-lg p-4 max-w-[75%]">
        <span>{{ message.body }}</span>
    </div>
    <div class="flex items-end">
        <svg height="13" width="8" >
            <path fill="#bbf7d0"
d="M6.3,10.4C1.5,8.7,0.9,5.5,0,0.2L0,13l5.2,0C7,13,9.6,11.5,6.3,10.4z"/>
        </svg>
    </div>
</li>
{% else %}
<li>
    <div class="flex justify-start">
        <div class="flex items-end mr-2" >
            <a href="{% url 'profile' message.author.username %}">
                <img class="w-8 h-8 rounded-full object-cover" src="{{
message.author.profile.avatar }}">
            </a>
        </div>
        <div class="flex items-end" >
            <svg height="13" width="8" >
                <path fill="white" d="M2.8,13L8,13L8,0.2C7.1,5.5,6.5,8.7,1.7,10.4C-
1.6,11.5,1,13,2.8,13z"></path>
            </svg>
        </div>
        <div class="bg-white p-4 max-w-[75%] rounded-r-lg rounded-tl-lg">
```

```
                <span>{{ message.body }}</span>
            </div>
        </div>
        <div class="text-sm font-light py-1 ml-10">
            <span class="text-white">{{ message.author.profile.name }}</span>
            <span class="text-gray-400">@{{ message.author.username }}</span>
        </div>
    </li>
    {% endif %}
    {% endfor %}
  </ul>
</div>
<div class="sticky bottom-0 z-10 p-2 bg[#232d32]">
  <div class="flex flex-col gap-4 items-center rounded-xl px-2 py-2">
    <form id="chat_message_form"  class="flex items-center w-full"
      hx-ext="ws"
      ws-connect="/ws/chatroom/{{ chatroom_name }}"
      ws-send
       ="on htmx:wsAfterSend reset() me">
      {% csrf_token %}
      {{ form }}
    </form>
    <form id="chat_file_form" enctype="multipart/form-data" class="flex items-center w-full"
      hx-post="{% url 'chat-file-upload' chat_group.group_name %}"
      hx-target="#chat_messages"
      hx-swap="beforeend"
       ="on htmx:beforeSend reset() me" >
      {% csrf_token %}
```

```
                <input type="file" name="file" id="id_file" class="!bg-transparent text-gray-400">
                <button type="submit" class="whitespace-nowrap !text-sm !py-3 !px-4 h-fit">Submit File</button>
            </form>
        </div>
    </div>
</div>
</wrapper>

{% endblock %}

{%block javascript %}

<script>
    function scrollToBottom() {
        const container = document.getElementById('chat_container');
        container.scrollTop = container.scrollHeight;
    }
    scrollToBottom()
</script>

{% endblock %}
```

Файл consumers.py

```
from channels.generic.websocket import WebsocketConsumer
from django.shortcuts import get_object_or_404
from django.template.loader import render_to_string
from asgiref.sync import async_to_sync
```

```
import json
from .models import *

class ChatroomConsumer(WebsocketConsumer):
    def connect(self):
        self.user = self.scope['user']
        self.chatroom_name = self.scope['url_route']['kwargs']['chatroom_name']
        self.chatroom = get_object_or_404(ChatGroup, group_name=self.chatroom_name)

        async_to_sync(self.channel_layer.group_add)(
            self.chatroom_name, self.channel_name
        )

        #add
        if self.user not in self.chatroom.users_online.all():
            self.chatroom.users_online.add(self.user)
            self.update_online_count()

        self.accept()

    def disconnect(self, close_code):
        async_to_sync(self.channel_layer.group_discard)(
            self.chatroom_name, self.channel_name
        )

        if self.user in self.chatroom.users_online.all():
            self.chatroom.users_online.remove(self.user)
            self.update_online_count()
```

```
def receive(self, text_data):
    text_data_json = json.loads(text_data)
    body = text_data_json['body']

    message = GroupMessage.objects.create(
        body = body,
        author = self.user,
        group = self.chatroom
    )
    event = {
        'type': 'message_handler',
        'message_id': message.id,
    }
    async_to_sync(self.channel_layer.group_send)(
        self.chatroom_name, event
    )

def message_handler(self, event):
    message_id = event['message_id']
    message = GroupMessage.objects.get(id=message_id)
    context ={
        'message': message,
        'user': self.user,
    }
    html = render_to_string("a_rtchat/partials/chat_message_p.html", context=context)
    self.send(text_data=html)
```

```
    def update_online_count(self):
        online_count = self.chatroom.users_online.count() - 1
        event = {
            'type': 'online_count_handler',
            'online_count': online_count
        }
        async_to_sync(self.channel_layer.group_send)(self.chatroom_name, event)


    def online_count_handler(self, event):
        online_count = event['online_count']
        context = {
            'online_count' : online_count,
            'chat_group' : self.chatroom


        }
        html = render_to_string("a_rtchat/partials/online_count.html", context)
        self.send(text_data=html)
```

Файл header.html

```
{% load static %}
<header class="flex items-center justify-between bg-[#1a1820] h-20 px-8 text-white sticky top-0 z-40">
    <div class="flex items-center">
        <a class="flex items-center gap-2" href="/">
            <img class="h-6" src="{% static 'images/logo.svg' %}" alt="Logo"/>
            <span class="text-lg font-bold">-Sphar</span>
        </a>
        <div id="online-user-count"></div>
    </div>
    <nav class="block bg-[#1a1820] relative">
```

```
<ul class="navitems flex items-center justify-center h-full">
    {% if request.user.is_authenticated %}
    <li x-data="{ dropdownChatOpen: false }" class="relative">
      <a @click="dropdownChatOpen = !dropdownChatOpen" @click.away="dropdownChatOpen = false"
        <button class="flex items-center gap-2">
          Chat
            <img x-bind:class="dropdownChatOpen && 'rotate-180 duration-300'" class="w-4" src="https://img.icons8.com/small/32/ffffff/expand-arrow.png" alt="Dropdown"> <!-- Рядок 16: Додано button і src -->
        </button>
      </a>
      <div x-show="dropdownChatOpen" x-cloak class="absolute right-0 bg-white text-black"
        x-transition:enter="duration-300 ease-out"
        x-transition:enter-start="opacity-0 -translate-y-5 scale-90"
          x-transition:enter-end="opacity-100 translate-y-0 scale-100"> <!-- Рядок 18: Додано синтаксис класу та атрибути -->
        <ul class="hoverlist [&>li>a]:justify-end">
          <li><a href="{% url 'home' %}">Public Chat</a></li>
          {% for chatroom in user.chat_groups.all %}
          {% if chatroom.groupchat_name %}
          <li>
            <a class="leading-5" href="{% url 'chatroom' chatroom.group_name %}">
              {{ chatroom.groupchat_name }}
            </a>
          </li>
          {% endif %}
          {% endfor %}
          {% for chatroom in user.chat_groups.all %}
            {% if chatroom.is_private %}
```

```
                    {% for member in chatroom.members.all %}
                        {% if member != user %}
                                <li><a href="{% url 'chatroom' chatroom.group_name %}">{{ member.profile.name }}</a></li>
                        {% endif %}
                    {% endfor %}
                {% endif %}
            {% endfor %}
        </ul>
    </div>
</li>
<li x-data="{ dropdownOpen: false }" class="relative">
    <a @click="dropdownOpen = !dropdownOpen" @click.away="dropdownOpen = false" class="cursor-pointer select-none">
    <img class="h-8 w-8 rounded-full object-cover" src="{{ user.profile.avatar }}" alt="Avatar" />
    {{ user.profile.name }}
        <img x-bind:class="dropdownOpen && 'rotate-180 duration-300'" class="w-4" src="https://img.icons8.com/small/32/ffffff/expand-arrow.png" alt="Dropdown" />
    </a>
    <div x-show="dropdownOpen" x-cloak class="absolute right-0 bg-white text-black shadow rounded-lg w-40 p-2 z-20"
    x-transition:enter="duration-300 ease-out"
    x-transition:enter-start="opacity-0 -translate-y-5 scale-90"
    x-transition:enter-end="opacity-100 translate-y-0 scale-100">
        <ul class="hoverlist [&>li>a]:justify-end">
            <li><a href="{% url 'profile' %}">My Profile</a></li>
            <li><a href="{% url 'profile-edit' %}">Edit Profile</a></li>
            <li><a href="{% url 'new-groupchat' %}">Create Chat</a></li>
            <li><a href="{% url 'profile-settings' %}">Settings</a></li>
```

```
                <li><a href="{% url 'account_logout' %}">Log Out</a></li>
            </ul>
        </div>
    </li>
    {% else %}
    <li><a href="{% url 'account_login' %}">Login</a></li>
            <li><a href="{% url 'account_signup' %}?next={% url 'profile-onboarding' %}">Signup</a></li>
    {% endif %}
  </ul>
 </nav>
</header>
```

Файл views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.urls import reverse
from allauth.account.utils import send_email_confirmation
from django.contrib.auth.decorators import login_required
from django.contrib.auth import logout
from django.contrib.auth.models import User
from django.contrib.auth.views import redirect_to_login
from django.contrib import messages
from .forms import *


def profile_view(request, username=None):
    if username:
        profile = get_object_or_404(User, username=username).profile
    else:
        try:
```

```
            profile = request.user.profile
        except:
            return redirect_to_login(request.get_full_path())
    return render(request, 'a_users/profile.html', {'profile':profile})


@login_required
def profile_edit_view(request):
    form = ProfileForm(instance=request.user.profile)

    if request.method == 'POST':
        form = ProfileForm(request.POST, request.FILES, instance=request.user.profile)
        if form.is_valid():
            form.save()
            return redirect('profile')

    if request.path == reverse('profile-onboarding'):
        onboarding = True
    else:
        onboarding = False

    return render(request, 'a_users/profile_edit.html', { 'form':form, 'onboarding':onboarding })


@login_required
def profile_settings_view(request):
    return render(request, 'a_users/profile_settings.html')


@login_required
def profile_emailchange(request):
```

```
    if request.htmx:
        form = EmailForm(instance=request.user)
        return render(request, 'partials/email_form.html', {'form':form})

    if request.method == 'POST':
        form = EmailForm(request.POST, instance=request.user)

        if form.is_valid():

@login_required
def profile_usernamechange(request):
    if request.htmx:
        form = UsernameForm(instance=request.user)
        return render(request, 'partials/username_form.html', {'form':form})

    if request.method == 'POST':
        form = UsernameForm(request.POST, instance=request.user)

        if form.is_valid():
            form.save()
            messages.success(request, 'Username updated successfully.')
            return redirect('profile-settings')
        else:
            messages.warning(request, 'Username not valid or already in use')
            return redirect('profile-settings')

    return redirect('profile-settings')
```

```
@login_required
def profile_emailverify(request):
    send_email_confirmation(request, request.user)
    return redirect('profile-settings')


@login_required
def profile_delete_view(request):
    user = request.user
    if request.method == "POST":
        logout(request)
        user.delete()
        messages.success(request, 'Account deleted, what a pity')
        return redirect('home')


    return render(request, 'a_users/profile_delete.html')
```

Файл profile_settings.html:

```
{% extends 'layouts/box.html' %}


{% block content %}


<h1 class="mb-8">Account Settings</h1>


<table class="w-full text-sm text-left text-gray-500">
    <tbody>
        <tr>
            <th scope="row" class="pt-4 pb-1 text-base font-bold text-gray-900">
                Email address
```

```
    </th>
    <td id="email-address" class="pt-4 pb-1 pl-4">
        {% if user.email %}{{ user.email }}{% else %}No Email{% endif %}
    </td>
    <td class="pt-4 pb-1 pl-4">
        <a id="email-edit" class="cursor-pointer font-medium text-blue-600 hover:underline"
            hx-get="{% url 'profile-emailchange' %}"
            hx-target="#email-address"
            hx-swap="innerHTML" >
            Edit
        </a>
    </td>
</tr>

<tr class="border-b">
    <th scope="row" class="pb-4 font-medium text-gray-900">
    </th>
    <td class="pb-4 pl-4">
        {% if user.emailaddress_set.first.verified %}
        <span class="text-green-500">Verified</span>{% else %}

        {% endif %}
    </td>
    <td class="pb-4 pl-4">

    </td>
</tr>
```

```
<tr class="border-b">
    <th scope="row" class="py-4 text-base font-bold text-gray-900">
        Username
    </th>
    <td id="username" class="py-4 pl-4">
        {{ user.username }}
    </td>
    <td class="py-4 pl-4">
        <a id="username-edit" class="cursor-pointer font-medium text-blue-600 hover:underline"
            hx-get="{% url 'profile-usernamechange' %}"
            hx-target="#username"
            hx-swap="innerHTML" >
            Edit
        </a>
    </td>
</tr>


<tr class="">
    <th scope="row" class="py-4 text-base font-bold text-gray-900">
        Delete Account
    </th>
    <td class="py-4 pl-4">
        Once deleted, account is gone. Forever.
    </td>
    <td class="py-4 pl-4">
        <a href="{% url 'profile-delete' %}" class="font-medium text-red-600 hover:underline">
            Delete
        </a>
```

```
            </td>
        </tr>
    </tbody>
</table>


{% endblock %}
```

Файл forms.py

```
from django.forms import ModelForm
from django import forms
from .models import *


class ChatmessageCreateForm(ModelForm):
    class Meta:
        model = GroupMessage
        fields = ['body']
        widgets = {
            'body' : forms.TextInput(attrs={'placeholder' : 'Add message ...', 'class': 'p-4 text-black', 'maxlength' : '300', 'autofocus': True}),
        }


class NewGroupForm(ModelForm):
    class Meta:
        model = ChatGroup
        fields = ['groupchat_name']
        widgets = {
            'groupchat_name' : forms.TextInput(attrs={
                'placeholder': 'Add name ...',
                'class': 'p-4 text-black',
```

```
            'maxlength' : '300',
            'autofocus': True,
        }),
    }

class ChatRoomEditForm(ModelForm):
    class Meta:
        model = ChatGroup
        fields = ['groupchat_name']
        widgets = {
            'groupchat_name' : forms.TextInput(attrs={
                'class': 'p-4 text-xl font-bold mb-4',
                'maxlength' : '300',
            }),
        }
```

Файл models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.conf import settings

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    image = models.ImageField(upload_to='avatars/', null=True, blank=True)
    displayname = models.CharField(max_length=20, null=True, blank=True)
    info = models.TextField(null=True, blank=True)

    def __str__(self):
        return str(self.user)
```

```
    @property
    def name(self):
        if self.displayname:
            return self.displayname
        return self.user.username


    @property
    def avatar(self):
        if self.image:
            return self.image.url
        return f'{settings.STATIC_URL}images/avatar.svg'
```

Файл chat_message.html:

```
{% if message.author == user %}
<li class="flex justify-end mb-4">
    <div class="bg-green-200 rounded-l-lg rounded-tr-lg p-4 max-w-[75%]">
        {% include 'a_rtchat/partials/message_content.html' %}
    </div>
    <div class="flex items-end">
        <svg height="13" width="8" >
          <path fill="#bbf7d0" d="M6.3,10.4C1.5,8.7,0.9,5.5,0,0.2L0,13l5.2,0C7,13,9.6,11.5,6.3,10.4z"/>
        </svg>
    </div>
</li>
{% else %}
<li>
    <div class="flex justify-start">
        <div class="flex items-end mr-2" >
```

```
            <a href="{% url 'profile' message.author.username %}">
                <img class="w-8 h-8 rounded-full object-cover" src="{{ message.author.profile.avatar }}">
            </a>
        </div>
        <div class="flex items-end" >
            <svg height="13" width="8" >
                                <path fill="white" d="M2.8,13L8,13L8,0.2C7.1,5.5,6.5,8.7,1.7,10.4C-1.6,11.5,1,13,2.8,13z"></path>
            </svg>
        </div>
        <div class="bg-white p-4 max-w-[75%] rounded-r-lg rounded-tl-lg">
            {% include 'a_rtchat/partials/message_content.html' %}
        </div>
    </div>
    <div class="text-sm font-light py-1 ml-10">
        <span class="text-white">{{ message.author.profile.name }}</span>
        <span class="text-gray-400">@{{ message.author.username }}</span>
    </div>
</li>
{% endif %}
```

Файл profile_edit.html

```
{% extends 'layouts/box.html' %}

{% block content %}

{% if onboarding %}
<h1 class="mb-4">Complete your Profile</h1>
{% else %}
```

```
<h1 class="mb-4">Edit your Profile</h1>

{% endif %}


<div class="text-center flex flex-col items-center">

    <img id="avatar" class="w-36 h-36 rounded-full object-cover my-4" src="{{ user.profile.avatar }}" />

  <div class="text-center max-w-md">

    <h1 id="displayname">{{ user.profile.displayname|default:"" }}</h1>

    <div class="text-gray-400 mb-2 -mt-3">@{{ user.username }}</div>

  </div>

</div>

<form method="POST" enctype="multipart/form-data">

  {% csrf_token %}

  {{ form.as_p }}

  <button type="submit" >Submit</button>

  {% if onboarding %}

  <a class="button button-gray ml-1" href="{% url 'home' %}">Skip</a>

  {% else %}

    <a class="button button-gray ml-1" href="{{ request.META.HTTP_REFERER }}">Cancel</a>

  {% endif %}

</form>


<script>


  const fileInput = document.querySelector('input[type="file"]');
```

```
    fileInput.addEventListener('change', (event) => {
    const file = event.target.files[0];
    const image = document.querySelector('#avatar');

    if (file && file.type.includes('image')) {
        const url = URL.createObjectURL(file);
        image.src = url;
    }
    });

    const display_nameInput = document.getElementById('id_displayname');
    const display_nameOutput = document.getElementById('displayname');

    display_nameInput.addEventListener('input', (event) => {
        display_nameOutput.innerText = event.target.value;
    });

</script>

{% endblock %}
```

Файл profile_delete.html

```
{% extends 'layouts/box.html' %}

{% block content %}
```

```
<h1>Delete Account</h1>
<p>Are you sure you want to delete your account?</p>

<form method='POST'>
    {% csrf_token %}
    <button type="submit" class="button-red mt-6">Yes, I want to delete my account</button>
    <a class="button button-gray ml-1" href="{{ request.META.HTTP_REFERER }}">Cancel</a>
</form>

{% endblock %}
```

Файл urls.py

```
from django.urls import path
from .views import *

urlpatterns = [
    path('', chat_view, name ="home"),
    path('chat/<username>', get_or_create_chatroom, name="start-chat"),
    path('chat/room/<chatroom_name>', chat_view, name="chatroom"),
    path('chat/new_groupchat/', create_groupchat, name="new-groupchat"),
    path('chat/edit/<chatroom_name>', chatroom_edit_view, name="edit-chatroom"),
    path('chat/delete/<chatroom_name>', chatroom_delete_view, name="chatroom-delete"),
    path('chat/fileupload/<chatroom_name>', chat_file_upload, name="chat-file-upload"),
]
```

Файл base.html

```
{% load static %}
{% load django_htmx %}
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="Django Template">
    <title>Project Title</title>
    <link rel="icon" type="image/x-icon" href="{% static 'favicon.ico' %}">
    <script src="https://cdn.jsdelivr.net/npm/alpinejs@3.x.x/dist/cdn.min.js" defer></script>
    <script src="https://unpkg.com/htmx.org@2.0.2"></script>
    <script src="https://unpkg.com/htmx.org/dist/ext/ws.js"></script>
    <script src="https://unpkg.com/hyperscript.org@0.9.12"></script>
    {% django_htmx_script %}
    <script src="https://cdn.tailwindcss.com"></script>
    <style type="text/tailwindcss">
        [x-cloak] {
            display: none !important;
        }
        h1 {
            @apply text-4xl font-bold mb-4
        }
        h2 {
            @apply text-xl font-bold mb-2
        }
        h3 {
            @apply text-lg font-bold
        }
        p {
            @apply mb-4
```

```
}
.button, button, [type='submit'], [type='button'] {
    @apply bg-indigo-600 text-white font-bold rounded-lg shadow-lg transition-all cursor-pointer
}
.button, button a, [type='submit'], [type='button'] {
    @apply px-6 py-4 inline-block
}
.button:hover, button:hover, [type='submit']:hover, [type='button']:hover {
    @apply bg-indigo-700
}
.button:active, button:active, [type='submit']:active, [type='button']:active {
    @apply scale-95
}
.button.alert, button.alert {
    @apply bg-red-700
}
.button.alert:hover, button.alert:hover {
    @apply bg-red-600
}
.button-red {
    @apply !bg-red-500 hover:!bg-red-600
}
.button-gray {
    @apply !bg-gray-300 hover:!bg-[#c3c9d0]
}
.navitems>li>a {
    @apply flex items-center gap-2 h-12 px-4 hover:bg-[rgba(31,41,55,0.3)] rounded-lg;
}
```

```
.hoverlist>* {
    @apply hover:bg-gray-100 rounded-md transition duration-150;
}
.hoverlist>*>a {
    @apply flex items-center p-2;
}
.highlight {
    @apply !bg-indigo-100;
}
.allauth content a {
    @apply underline underline-offset-2
}
.allauth content a:hover {
    @apply text-indigo-500
}
.allauth form[action="/accounts/signup/"] ul {
    @apply hidden
}
.allauth form[action="/accounts/signup/"] ul.errorlist {
    @apply block
}
.allauth .helptext {
    @apply block mt-4
}
label {
    @apply hidden
}
input[type=file] {
```

```
        @apply bg-white pl-0
    }
    .textarea, textarea, input {
        @apply w-full rounded-lg py-4 px-5 bg-gray-100
    }
    .errorlist li {
        @apply p-1 pl-4 border-l-red-500 border-l-4 border-solid mb-2 text-red-500
    }
    label[for="id_remember"] {
        @apply inline-block w-auto mr-2
    }
    input[name="remember"] {
        @apply w-auto
    }
    .alert-info { @apply bg-sky-500 }
    .alert-success { @apply bg-green-500 }
    .alert-warning { @apply bg-red-500 }
    .alert-danger { @apply bg-red-500 }
    .green-dot {
        @apply rounded-full bg-green-500 p-1.5
    }
    .gray-dot {
        @apply rounded-full bg-gray-500 p-1.5
    }
    .graylight-dot {
        @apply rounded-full bg-gray-300 p-1.5
    }
</style>
```

```
</head>

<body hx-headers='{"X-CSRFToken": "{{ csrf_token }}"}' class="{% block class %}{% endblock %}">


    {% include 'includes/messages.html' %}


    {% include 'includes/header.html' %}


    {% block layout %}

    {% endblock %}


    {% block javascript %}{% endblock %}


</body>

</html>
```

Файл settings.py

```
from pathlib import Path


BASE_DIR = Path(__file__).resolve().parent.parent


SECRET_KEY = 'django-insecure-rj#-z^kx3j+1ay397otg6j8m_8#v^$^$jys6&41vy^&6le)ezc'


DEBUG = True


ALLOWED_HOSTS = ['localhost', '127.0.0.1', '*']


CSRF_TRUSTED_ORIGINS = [ 'https://*' ]
```

```
INSTALLED_APPS = [
    'daphne',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_cleanup.apps.CleanupConfig',
    'django_htmx',
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'a_home',
    'a_users',
    'a_rtchat',
]

SITE_ID = 1

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
```

```
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'allauth.account.middleware.AccountMiddleware',
    'django_htmx.middleware.HtmxMiddleware',
]

AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
    'allauth.account.auth_backends.AuthenticationBackend',
]

ROOT_URLCONF = 'a_core.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [ BASE_DIR / 'templates' ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

```
ASGI_APPLICATION = 'a_core.asgi.application'


CHANNEL_LAYERS = {
    "default": {
        "BACKEND": "channels.layers.InMemoryChannelLayer",
    }
}



DATABASES = {
   'default': {
       'ENGINE': 'django.db.backends.sqlite3',
       'NAME': BASE_DIR / 'db.sqlite3',
   }
}


AUTH_PASSWORD_VALIDATORS = [
   {
       'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
   },
   {
       'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
   },
   {
       'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
   },
   {
```

```
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

STATIC_URL = 'static/'
STATICFILES_DIRS = [ BASE_DIR / 'static' ]

MEDIA_URL = 'media/'
MEDIA_ROOT = BASE_DIR / 'media'

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

LOGIN_REDIRECT_URL = '/'

EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
ACCOUNT_LOGIN_METHODS = {'email'}
```