МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»
на тему: «Web-додаток обміну повідомленнями»

_____

Здобувача(ки) групи ІТ-92-0/2 Дудченка Ярослава Андрійовича
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ярослав ДУДЧЕНКО
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник старший викладач кафедри ІТ, к.т.н. Ольга БОЙКО _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

_____ Світлана ВАЩЕНКО

«__» _____ 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Дудченко Ярослав Андрійович

1 Тема роботи *Web-додаток обміну повідомленнями*

керівник роботи *Бойко Ольга Василівна, к.т.н.* ,

затверджені наказом по університету №0588-VI від «29» травня 2023 р. _____

2 Строк подання студентом роботи «7» червня 2023 р.

3 Вхідні дані до роботи технічне завдання на розробку web-додатку обміну повідомленнями

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, проектування web-додатку, програмна реалізація web-додатку, розробка технічного завдання, планування робіт

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

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Оформлення планування робіт	24.02.2023 – 03.03.2023	
2	Оформлення технічного завдання	04.03.2023 – 08.03.2023	
3	Аналіз предметної області	09.03.2023 – 23.03.2023	
4	Проектування web-додатку	24.03.2023 – 31.03.2023	
5	Розробка web-додатку	01.04.2023 – 15.05.2023	
6	Тестування web-додатку	16.05.2023 – 23.05.2023	
7	Завантаження web-додатку на хостинг	24.05.2023 – 31.05.2023	
8	Оформлення пояснювальної записки	01.06.2023 – 10.06.2023	

Студент _____ Ярослав ДУДЧЕНКО
(підпис)

Керівник роботи _____ к.т.н., Ольга БОЙКО
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Web-додаток обміну повідомленнями».

Пояснювальна записка складається зі вступу, трьох розділів, висновків, списку використаних джерел із 23 найменувань, додатків. Загальний обсяг роботи − 119 сторінок, у тому числі 46 сторінок основного тексту, 3 сторінки списку використаних джерел, 65 сторінок додатків.

Кваліфікаційну роботу бакалавра присвячена розробці web-додатку обміну повідомленнями, який дозволить користувачам легко спілкуватись та обмінюватись інформацією. Цей додаток буде забезпечувати зручну платформу для взаємодії та співпраці між користувачами, що допоможе вирішувати важливі завдання та економити час і ресурси.

Перший розділ містить огляд попередніх досліджень та аналогів, що стосуються тематики роботи, а також визначає мету та задачі проекту.

Другий розділ включає структурно-функціональне моделювання, побудову UML діаграм, проектування бази даних. На цьому етапі було спроектовані такі види діаграм: контекстна діаграма IDEF0 та її декомпозиція, діаграма варіантів використання, три діаграми діяльності та одна діаграма послідовності.

У третьому розділі надана детальна архітектура web-додатку, де описані основні компоненти та їхні взаємозв'язки. Також показані основні етапи розробки web-додатку та технології розробки.

Результатом нашої роботи є готовий та працездатний web-додаток. Цей додаток надає користувачам зручну та ефективну платформу для спілкування та обміну повідомленнями. Він допомагає вирішувати важливі завдання та оптимізувати використання часових та матеріальних ресурсів.

Ключові слова: web-додаток, React, NodeJS, PostgreSQL, Express, Sass.

ЗМІСТ

ВСТУП

У сучасному швидкоплинному світі комунікація відіграє життєво важливу роль у підтримці зв'язку між людьми, а web-програми для обміну повідомленнями стали невід'ємною частиною сучасної комунікації. З широким розповсюдженням смартфонів та пристроїв з доступом до Інтернету, додатки для обміну повідомленнями стали зручним і популярним способом комунікації.

Також web-додатки обміну повідомленнями є актуальними для приватного та корпоративного спілкування і мають значну цінність незалежно від часу. Особливо зростає значення таких додатків у сучасному світі, де ми все більше спілкуємося в онлайн-середовищі.

Актуальність використання web-додатків обміну повідомленнями обумовлена:

1. Можливістю забезпечення приватного спілкування: люди шукають безпечний і зручний спосіб спілкування зі своїми друзями, сім'єю та колегами. Web-додаток обміну повідомленнями забезпечує можливість надсилати повідомлення, фотографії, відео та документи, дозволяючи людям підтримувати зв'язок у будь-який час і з будь-якого місця.

2. Можливістю забезпечення корпоративного спілкування: у сучасних бізнес-середовищах корпоративні команди все більше працюють в розподілених команд, віддалених офісах та фрілансерах. Web-додаток обміну повідомленнями стає основним інструментом для спілкування внутрішньої команди, обміну файлами, спільної роботи над проектами та координації завдань.

3. Миттєвим доступом до інформації: web-додатки обміну повідомленнями забезпечують миттєвий доступ до повідомлень і даних, що робить їх незамінними для швидкого обміну інформацією. Вони дозволяють заощадити час, що зазвичай би витрачався на відправку електронної пошти або організацію зустрічей.

4. Мобільністю: зростаюча кількість людей використовує мобільні пристрої для спілкування і роботи. Web-додаток обміну повідомленнями може бути доступним

на різних платформах, включаючи веб-браузери та мобільні додатки, що дозволяє користувачам спілкуватися з будь-якого пристрою.

Основною метою цієї дипломної роботи є розробка зручного у використанні web-додатку для обміну повідомленнями. У розробці планується використання сучасних технологій web-розробки, таких як React, SCSS, і існуючого протоколу обміну повідомленнями Socket.IO.

Задачі дипломної роботи:

1. Проаналізувати існуючі web-додатки для обміну повідомленнями та визначити найбільш ефективні та зручні функції.
2. Розробити дизайн та архітектуру web-додатку.
3. Розробити функціонал web-додатку для реєстрації та авторизації користувачів.
4. Розробити функціонал web-додатку для обміну повідомленнями між користувачами в реальному часі.
5. Розробити функціонал web-додатку для створення групових чатів та обміну повідомленнями в них.
6. Протестувати розроблений web-додаток, виявити та виправити можливі помилки та недоліки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Останніми роками проводяться різні дослідження, що стосуються web-додатків обміну повідомленнями.

Одним із найцікавіших досліджень є аналіз технології мережевого сокету [1]. У дослідженні було досліджено використання технології WebSocket для створення web-додатків обміну повідомленнями. Результати показали, що використання WebSocket дає змогу значно збільшити продуктивність і швидкодію застосунку, що особливо важливо для обміну повідомленнями в режимі реального часу.

Ще у одному дослідженні було розглянуто проблему безпеки під час створення web-додатків обміну повідомленнями [2]. Автори дослідження представили низку методів і технологій, які дають змогу забезпечити безпеку застосунку та захистити його від можливих атак.

Також варто згадати дослідження стандарту передачі потокового аудіо [3]. У дослідженні було розглянуто використання технології WebRTC для створення web-додатків обміну повідомленнями. Результати показали, що WebRTC дає змогу створювати високопродуктивні та надійні додатки обміну повідомленнями, які можна використовувати для різних цілей, включно з голосовим і відеозв'язком.

Таким чином, дослідження, проведені останніми роками, підтверджують значущість і актуальність теми створення web-додатків обміну повідомленнями. Вони також представляють низку методів і технологій, які можуть використовуватися для створення продуктивніших, надійніших і безпечніших додатків.

Крім того, в останні роки було опубліковано безліч інших досліджень і публікацій, пов'язаних із темою створення web-додатків обміну повідомленнями. Деякі з них фокусуються на архітектурі додатків, інші - на методах оптимізації продуктивності, а треті - на безпеці та захисті даних. Вивчення цих публікацій може

допомогти у виборі оптимального підходу до створення web-додатків обміну повідомленнями та підвищити якість розроблюваного додатка.

1.2 Аналіз програмних продуктів-аналогів

Web-програми для обміну повідомленнями стали важливим інструментом для спілкування в цифрову епоху. Вони пропонують користувачам можливість обмінюватися повідомленнями та мультимедійним контентом в режимі реального часу через Інтернет. З поширенням цих додатків користувачі стали більш вибагливими до продуктів, які вони обирають для використання. Тому важливо проаналізувати різні програми для обміну повідомленнями, доступні на ринку, та порівняти їх за функціями, користувацьким інтерфейсом, безпекою та іншими параметрами [4].

Для аналізу та порівняння обрано такі критерії, як користувацький інтерфейс, функції, безпека, зручність використання, корпоративність та ціна користування для приватних розмов та корпоративних.

Першим досліджуємо месенджер WhatsApp – один з найпопулярніших додатків для обміну повідомленнями у світі. Для цього додатку можна виділити наступні характеристики:

1. Інтерфейс користувача: WhatsApp має простий і зручний інтерфейс, який дозволяє користувачам легко орієнтуватися і користуватися додатком. Додаток має чистий дизайн з мінімалістичним підходом, що робить його простим у використанні як для початківців, так і для досвідчених користувачів.

2. Функції: WhatsApp дозволяє користувачам обмінюватися повідомленнями, голосовими нотатками, зображеннями та відео. Додаток також забезпечує групові повідомлення, голосові та відеодзвінки, а також обмін файлами. Крім того, WhatsApp пропонує наскрізне шифрування, що забезпечує безпечне спілкування між користувачами.

3. Безпека: WhatsApp забезпечує наскрізне шифрування, що означає, що тільки відправник і одержувач повідомлення можуть прочитати його вміст. Додаток також використовує двофакторну аутентифікацію для захисту облікових записів користувачів.

4. Зручність використання: WhatsApp простий у використанні, а його простий дизайн дозволяє користувачам легко орієнтуватися в додатку. Додаток також надає користувачам можливість створювати резервні копії історії чату, що є корисною функцією для тих, хто хоче зберегти свої розмови.

5. Корпоративність: WhatsApp має окрему бізнес-версію під назвою "WhatsApp Business", яка пропонує функції, пристосовані для малих і середніх підприємств. Вона включає інструменти для управління взаємодією з клієнтами, створення бізнес-профілів і використання автоматизованих повідомлень.

6. Ціноутворення: WhatsApp можна використовувати безкоштовно для особистих розмов. Однак для корпоративного використання WhatsApp Business API пропонує платну модель, що базується на обсягах повідомлень та додаткових функціях. Ціна залежить від регіону та конкретних вимог бізнесу.

Рисунок 1.1 – Інтерфейс додатку WhatsApp

Наступним для аналізу можна виділити Facebook Messenger – це додаток для обміну повідомленнями, розроблений компанією Facebook. Характеристики цього додатку наступні:

1. Користувацький інтерфейс: Facebook Messenger має елегантний інтерфейс з широким спектром функцій, включаючи чат-боти, ігри та інтеграцію з іншими додатками. Інтерфейс програми добре організований і простий у використанні.

2. Можливості: Facebook Messenger дозволяє користувачам надсилати повідомлення, здійснювати голосові та відеодзвінки, ділитися зображеннями та відео. Додаток також забезпечує групові повідомлення, обмін файлами та можливість надсилати гроші. Крім того, Facebook Messenger пропонує наскрізне шифрування, що забезпечує безпечне спілкування між користувачами.

3. Безпека: Facebook Messenger забезпечує наскрізне шифрування, що означає, що тільки відправник і одержувач повідомлення можуть прочитати його зміст. Додаток також використовує двофакторну аутентифікацію для захисту облікових записів користувачів.

4. Зручність використання: Facebook Messenger простий у використанні, а його лаконічний інтерфейс надає користувачам широкий спектр функцій. Інтеграція з іншими програмами також робить його корисним інструментом для тих, хто хоче оптимізувати свій робочий процес.

5. Корпоративність: Facebook Messenger надає платформу під назвою "Messenger для бізнесу", яка дозволяє компаніям взаємодіяти з клієнтами за допомогою обміну повідомленнями. Вона пропонує такі функції, як автоматичні відповіді, чат-боти та інтеграцію зі сторінками Facebook для підтримки клієнтів.

6. Ціноутворення: Facebook Messenger безкоштовний для особистого використання. Ціни на корпоративне використання можуть включати витрати, пов'язані з рекламою та просуванням бізнес-сторінок у Facebook.

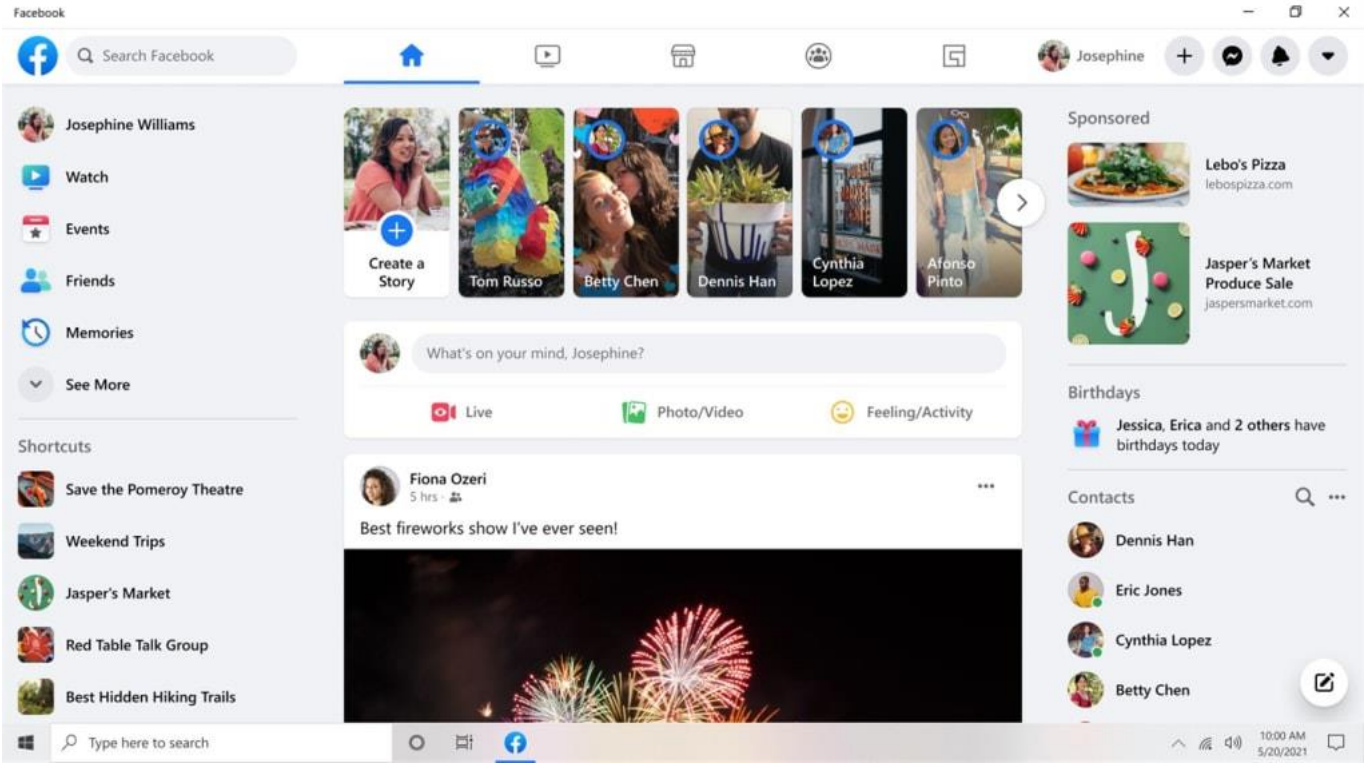


Рисунок 1.2 – Інтерфейс додатку Facebook

Далі для аналізу можна виділити Telegram – це хмарний додаток для обміну повідомленнями, який надає користувачам наскрізне шифрування та самознищення повідомлень. Характеристики наступні:

1. Користувацький інтерфейс: Telegram має простий та інтуїтивно зрозумілий користувацький інтерфейс, що робить його легким у використанні як для початківців, так і для досвідчених користувачів. Інтерфейс програми добре організований і надає користувачам широкий спектр функцій.

2. Можливості: Telegram дозволяє користувачам надсилати повідомлення, здійснювати голосові та відеодзвінки, ділитися зображеннями та відео. У додатку також є секретні чати, канали та групи. Крім того, Telegram пропонує наскрізне шифрування та самознищення повідомлень, що забезпечує безпечне спілкування між користувачами.

3. Безпека: Telegram забезпечує наскрізне шифрування та самознищення повідомлень, що означає, що лише відправник та отримувач повідомлення можуть

прочитати його вміст. Додаток також використовує двофакторну аутентифікацію для захисту облікових записів користувачів.

4. Зручність використання: Telegram простий у використанні, а його простий дизайн дозволяє користувачам легко орієнтуватися в додатку. Секретні чати та повідомлення, що самознищуються, надають користувачам додатковий рівень безпеки.

5. Корпоративність: Telegram пропонує такі функції, як групи та канали, які можна використовувати для бізнес-цілей. Він надає інструменти для створення публічних або приватних каналів для взаємодії з клієнтами або членами команди.

6. Ціноутворення: Telegram є безкоштовним у використанні, і пряма плата за приватні або корпоративні розмови не стягується. Однак користувачі можуть додатково купити підписку Telegram-Premium, яка надає додаткові стікери та збільшене хмарне сховище. Бізнес може понести витрати, якщо вирішить використовувати сторонніх ботів або сервіси, інтегровані з Telegram.

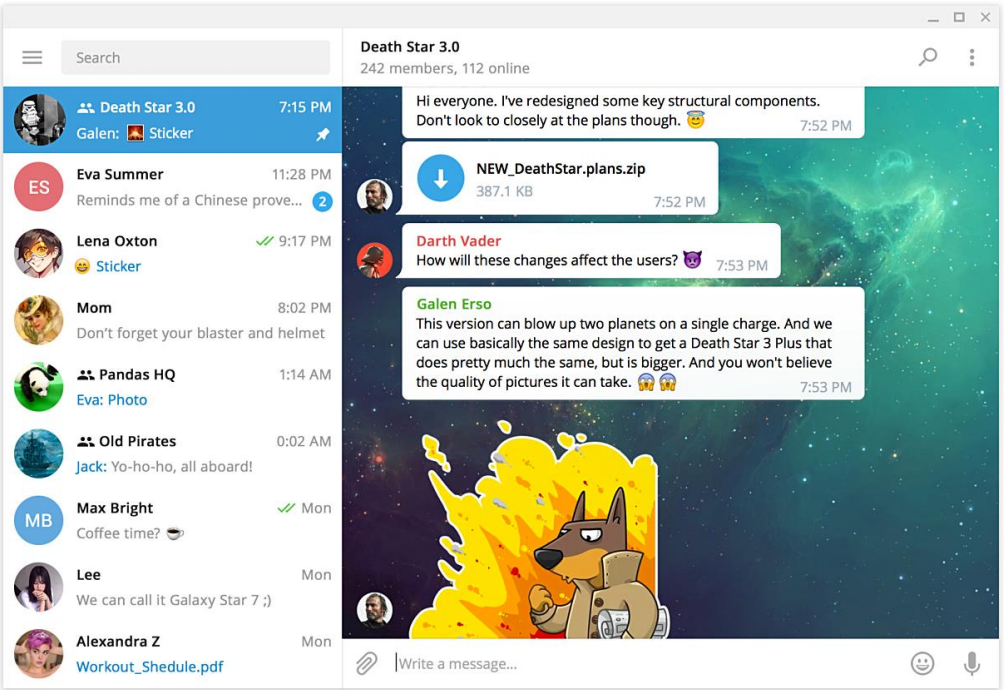


Рисунок 1.3 – Інтерфейс додатку Telegram

Також для аналізу можна виділити Discord – кросплатформна пропрієтарна система миттєвого обміну повідомленнями. Виділимо такі характеристики:

1. Інтерфейс користувача: Discord має сучасний та інтуїтивно зрозумілий користувацький інтерфейс, орієнтований на ігрові спільноти. Він надає інтерфейс на основі чату з різними каналами та макетами серверів, які можна індивідуально налаштовувати.

2. Функції: Discord пропонує текстові та голосові повідомлення, голосові та відеодзвінки, спільний доступ до екрану та обмін файлами. Він також надає функції, спеціально розроблені для ігрових спільнот, такі як голосові канали, ролі та інтеграція з ігровими платформами.

3. Безпека: Discord використовує стандартне шифрування для захисту комунікацій користувачів. Однак важливо зазначити, що хоча Discord і пропонує заходи конфіденційності та безпеки, він розроблений в першу чергу як соціальна платформа і може не забезпечувати такий самий рівень шифрування та безпеки, як деякі інші програми для обміну повідомленнями.

4. Зручність використання: Discord відомий своєю простотою використання, особливо в ігрових спільнотах. Інтерфейс простий і пропонує різні варіанти налаштування для створення унікальних спільнот.

5. Корпоративність: Хоча Discord в першу чергу популярний серед ігрових спільнот і випадкових користувачів, він також набув поширення в деяких бізнес-середовищах, особливо для малих і середніх команд. Однак він може не пропонувати той самий рівень функцій та інтеграцій, орієнтованих на підприємства, як Slack.

6. Ціноутворення: Discord є безкоштовним у використанні, з додатковими планами підписки під назвою "Discord Nitro", які пропонують додаткові переваги, такі як власні емодзі та розширені ліміти на завантаження файлів. Існує також бізнес-орієнтований план під назвою "Discord Nitro Classic", який пропонує переваги для великих спільнот і команд.

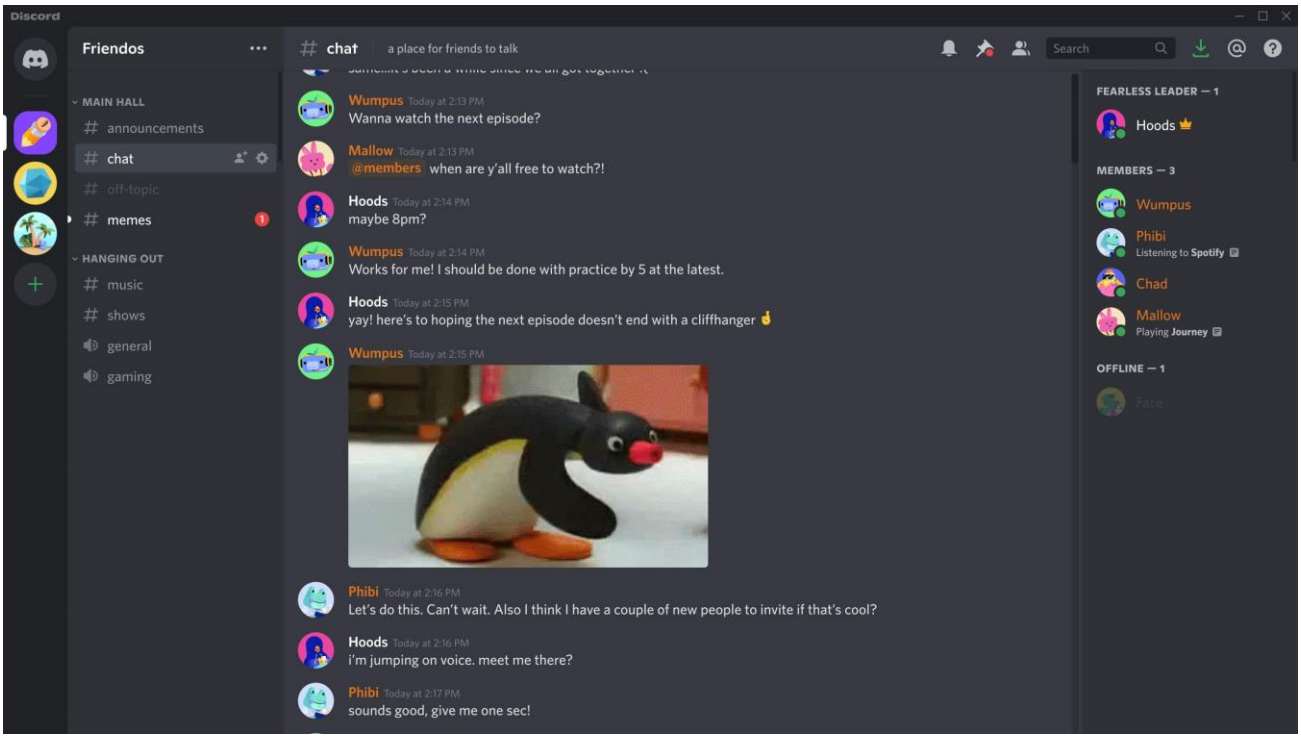


Рисунок 1.4 – Інтерфейс додатку Discord

І останнім для порівняння візьмемо Slack – корпоративний додаток для миттєвого обміну повідомленнями. Характеристики наступні:

1. Інтерфейс користувача: Slack має чистий і сучасний інтерфейс, орієнтований на командну роботу. Він надає канали, потоки та налаштовуваний макет робочого простору, що дозволяє користувачам легко організовувати розмови.

2. Можливості: Slack пропонує текстові повідомлення, голосові та відеодзвінки, обмін файлами, інтеграцію з різними програмами та сервісами, а також інструменти для спільної роботи, такі як спільні канали, опитування та нагадування. Він також надає розширену функцію пошуку для знаходження повідомлень і файлів.

3. Безпека: Slack використовує різні заходи безпеки, включаючи шифрування даних під час передачі та в стані спокою. Він пропонує функції безпеки корпоративного рівня, такі як двофакторна аутентифікація та управління корпоративними ключами, щоб забезпечити захист конфіденційної інформації.

4. Зручність використання: Slack розроблений для покращення командної співпраці та продуктивності. Його інтерфейс є зручним для користувача і надає такі

функції, як сповіщення, згадки та налаштовувані сповіщення, щоб тримати користувачів в курсі подій та організовано працювати.

5. Корпоративність: Slack широко використовується в корпоративному середовищі, починаючи від невеликих команд і закінчуючи великими організаціями. Він пропонує широкі можливості інтеграції зі сторонніми додатками та сервісами, що робить його універсальним інструментом для оптимізації робочих процесів та централізації комунікації.

6. Ціноутворення: Slack пропонує кілька тарифних планів залежно від розміру та потреб організації. Він надає безкоштовний план з обмеженими функціями та сховищем, а також платні плани, які пропонують додаткові функції, сховище та розширені засоби адміністрування для корпоративного використання.

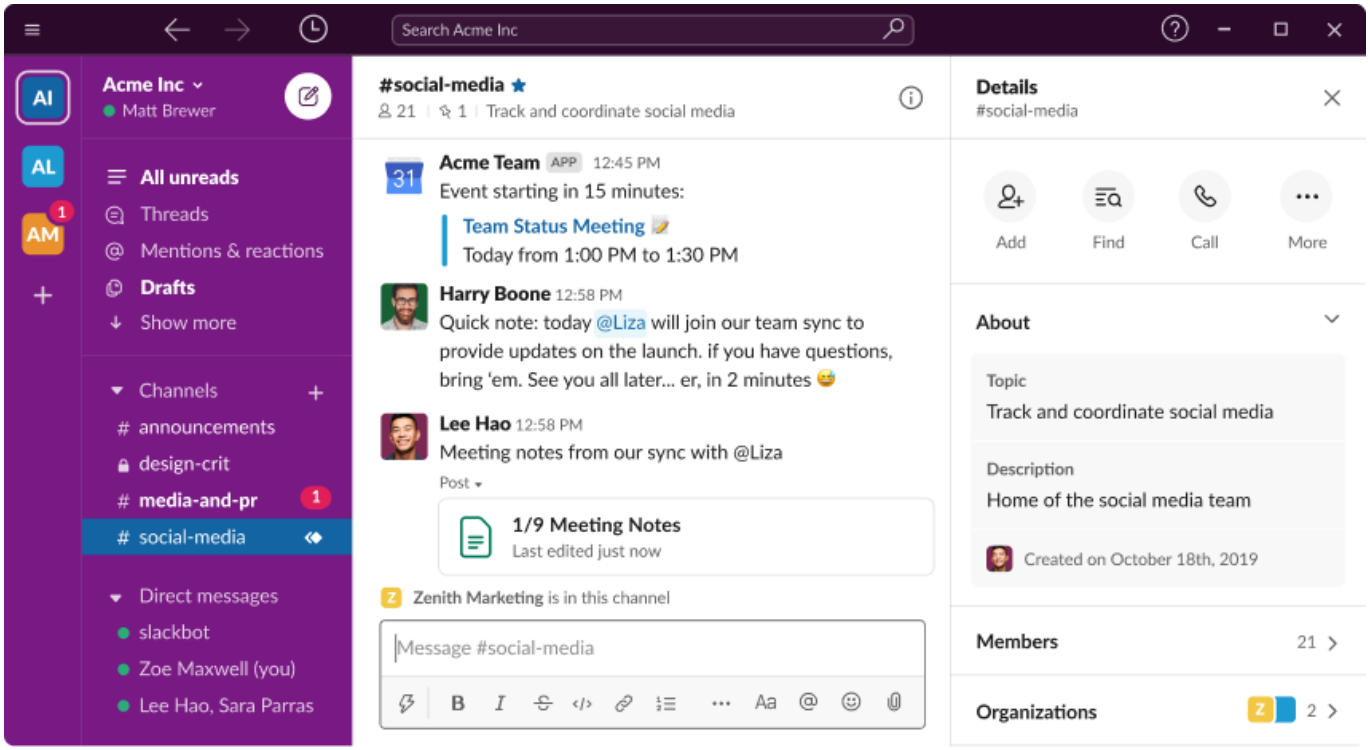


Рисунок 1.5 – Інтерфейс додатку Slack

Проаналізувавши характеристики даних додатків можна сформувати порівняльну таблицю.

Таблиця 1.1 – Порівняльний аналіз аналогів web-додатків обміну повідомленнями

Критерії	WhatsApp	Facebook Messenger	Telegram	Discord	Slack
Сучасний інтерфейс	+	+	+	+	+
Аудіо та відеодзвінки	+	+	+	+	+
Групові повідомлення	+	+	+	+	+
Обмін файлами	+	+	+	+	+
Секретні чати	-	-	+	-	-
Самознищуючі повідомлення	-	-	+	-	-
Корпоративність	+	+	+	+	+
Вартість приватних розмов	Безкоштовно	Безкоштовно	Безкоштовно/3$ в місяць або 32$ в рік за Premium підписку	Безкоштовно/9.99$ в місяць або 99.99$ в рік за підписку Discord Nitro	Безкоштовно

Продовження таблиці 1.1

Вартість корпоративних розмов	Від 0.0145$ за повідомлення	Безкоштовно, лише витрати на рекламу	Безкоштовно, лише витрати на сторонні сервіси та боти	Безкоштовно/4.99$ в місяць або 49.99$ в рік за підписку Discord Nitro Classic	Від 7.25$ за місяць

Отже, кожна веб-програма для обміну повідомленнями має свої сильні та слабкі сторони. WhatsApp і Facebook Messenger пропонують широкий спектр функцій, що робить їх ідеальними для користувачів, яким потрібен комплексний додаток для обміну повідомленнями. Telegram зосереджений на конфіденційності та безпеці, що робить його популярним серед користувачів, для яких ці функції є пріоритетними. Зрештою, вибір програми для використання залежить від індивідуальних потреб та уподобань користувача.

1.3 Постановка задачі

Мета цієї дипломної роботи є розробка та реалізація безпечного та зручного у використанні веб-додатку для обміну повідомленнями. Додаток буде розроблено з використанням сучасних технологій веб-розробки, таких як React, SCSS, і буде використовувати існуючий протокол обміну повідомленнями Socket.IO.

Завдання дипломної роботи:

1. Проаналізувати існуючі веб-додатки для обміну повідомленнями та визначити найбільш ефективні та зручні функції.
2. Розробити дизайн та спроектувати архітектуру веб-додатку.
3. Реалізувати функціонал веб-додатку для безпечної реєстрації та авторизації користувачів.

4. Реалізувати функціонал веб-додатку для обміну повідомленнями між користувачами в реальному часі.
5. Реалізувати функціонал веб-додатку для створення групових чатів та обміну повідомленнями в них.
6. Протестувати розроблений веб-додаток, виявити та виправити можливі помилки та недоліки.

Результатом дипломної роботи буде веб-додаток для обміну повідомленнями, який буде зручним та ефективним засобом спілкування користувачів через Інтернет. Додаток буде містити основні функції, що забезпечать його коректну роботу.

2 ПРОЕКТУВАННЯ WEB-ДОДАТКУ ОБМІНУ ПОВІДОМЛЕННЯМИ

2.1 Структурно-функціональне моделювання

Web-додатки обміну повідомленнями стали невід'ємною частиною сучасної комунікації, полегшуючи спілкування в режимі реального часу та обмін інформацією між різними платформами. Розробка таких додатків вимагає глибокого розуміння їхніх структурних і функціональних аспектів для забезпечення ефективної та зручної для користувача системи. Моделювання web-додатку обміну повідомленнями реалізується з використанням двох широко розповсюджених методів моделювання: IDEF0 та діаграми декомпозиції.

IDEF0 – це метод, який використовує графічне представлення для зображення систем і процесів організації як набору взаємопов'язаних функцій. Його метою є аналіз функцій в організації незалежно від об'єктів, відповідальних за їх виконання [5]. У стандарті IDEF0 вхідні дані використовуються для представлення таких об'єктів, як інформаційні та матеріальні потоки, які зазнають трансформації під час бізнес-процесу. Аспект управління відображає об'єкти, що беруть участь у процесі, включаючи матеріальні та інформаційні потоки, необхідні для його виконання [6]. Використання механізмів IDEF0 дозволяє наочно проілюструвати інструменти та ресурси, що використовуються для реалізації бізнес-процесу, такі як технології, персонал та інформаційні системи. Результат бізнес-процесу, описаного за допомогою стандарту IDEF0, за змістом близький до процесу, описаного на діаграмі потоків даних (DFD). Тим не менш, методологія IDEF0 демонструє невелике відхилення від традиційної схеми DFD для опису бізнес-процесів. Ключова відмінність полягає у включенні в мову додаткових аналітичних елементів. Цей стандарт опису бізнес-процесів пропонує відображати не тільки входи і виходи, як у форматі DFD, але й три типи входів. Перший тип зберігає назву "вхід", тоді як два інших називаються "елементи управління" і "механізми" [7].

Функціональне моделювання web-додатку обміну повідомленнями в нотації IDEF0 представлено на рисунку 2.1. Вхідними даними є інформація про користувача, текстова інформація та графічні зображення [8]. У моделі виділяються різні механізми, такі як web-додаток, технічні засоби та користувач. Функція, що визначається вимогами до функціонування системи, відповідає за управління цими механізмами. На виході отримуємо надіслане повідомлення.

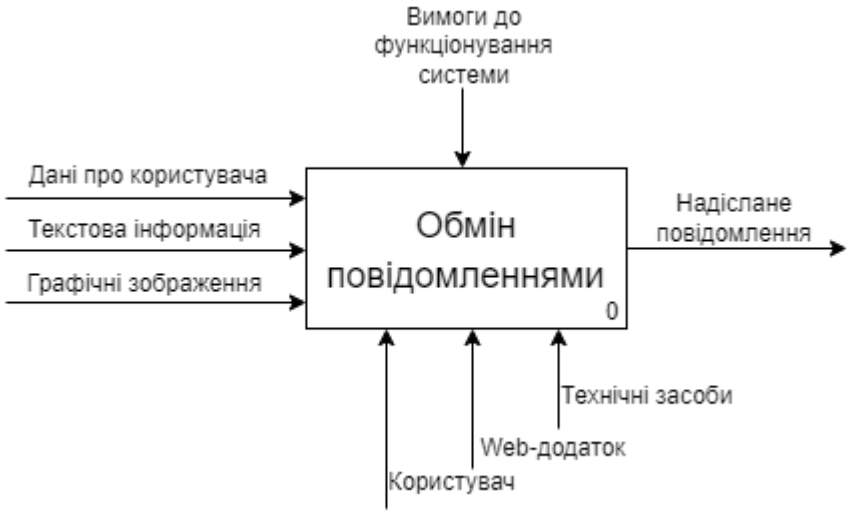


Рисунок 2.1 – Контекстна діаграма обміну повідомленнями в нотації IDEF0

Наведений web-додаток був розбитий на окремі підпроцеси для полегшення розуміння його структури. До цих підпроцесів входять авторизація, керування обліковим записом, відправлення повідомлення, отримання повідомлення, відображення повідомлення. Щоб зробити інформацію більш зручною для сприйняття, була створена таблиця, яка містить дані кожного з цих підпроцесів (табл. 2.1).

Таблиця 2.1 – Дані для складання діаграми декомпозиції

Підпроцес	Вхід	Управління	Механізми	Вихід
Авторизація	Дані про користувача	Вимоги до функціонування	Технічні засоби, web-додаток	Вхід у web-додаток

Продовження таблиці 2.1

Керування обліковим записом	Дані про користувача, графічні зображення	Вимоги до функціонування	Технічні засоби, web-додаток	Оновлені дані користувача
Вибір діалогу	Оновлені дані користувача	Вимоги до функціонування	Технічні засоби, web-додаток, користувач	Діалог
Друк повідомлення	Текстова інформація, графічні зображення	Вимоги до функціонування	Технічні засоби, web-додаток	Дані повідомлення
Відправлення повідомлення	Дані повідомлення	Вимоги до функціонування	Технічні засоби, web-додаток	Надіслане повідомлення

Діаграма декомпозиції першого рівня представлена на рисунку 2.2.

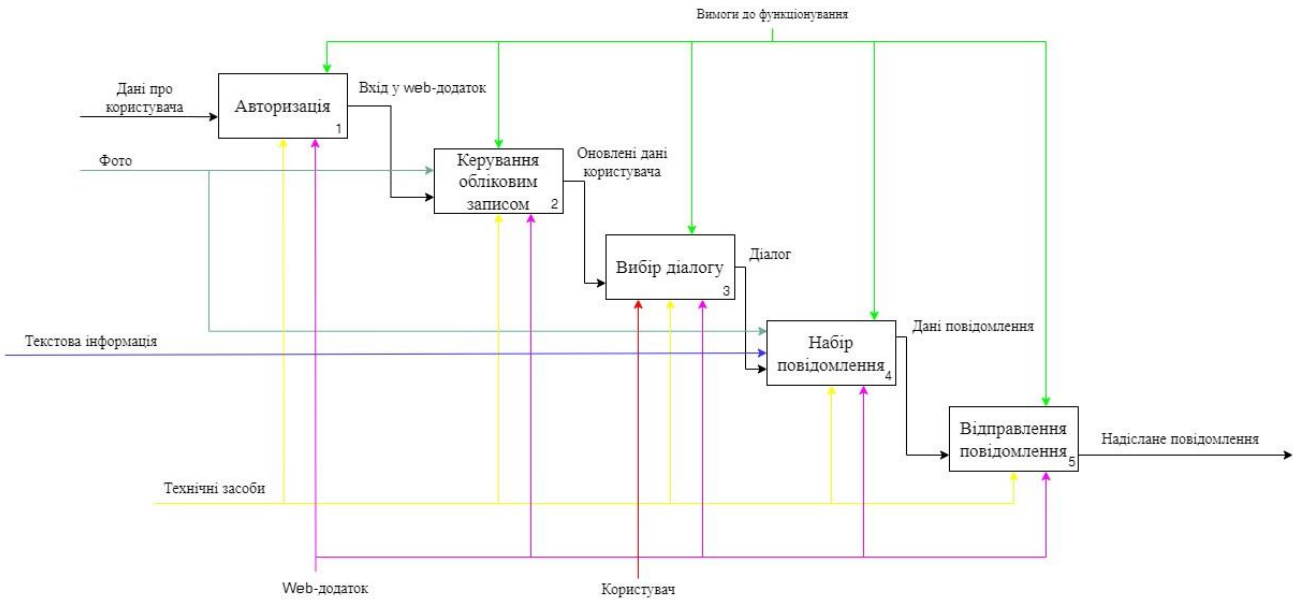


Рисунок 2.2 – Діаграма декомпозиції першого рівня

2.2 Моделювання варіантів використання web-додатку

Щоб зрозуміти функціональність і поведінку web-додатку для обміну повідомленнями, необхідно провести детальний аналіз варіантів використання. Цей аналіз включає в себе визначення залучених акторів, визначення їхніх цілей та опис взаємодії між акторами та системою. Аналіз варіантів використання дає цілісне уявлення про функції програми і допомагає охопити різні сценарії, в яких буде використовуватися система.

Основними акторами web-додатку обміну повідомленнями є користувач, база даних та програмний API інтерфейс, які наведено у таблиці 2.2.

Таблиця 2.2 – Опис акторів

Актор	Опис
Користувач	Особа, яка отримує доступ до програми обміну повідомленнями через веб-браузер. Користувач може виконувати різні дії, такі як надсилання повідомлень, створення груп, керування контактами та налаштування свого профілю.
База даних	База даних, створена засобами PostgreSQL, яка зберігає дані користувачів та повідомлень

На основі аналізу були визначені наступні варіанти використання веб-додатку для обміну повідомленнями (табл. 2.3).

Таблиця 2.3 – Опис варіантів використання

Назва	Опис
Реєстрація	Надає можливість користувачу створювати обліковий запис у web-додатку, надаючи необхідні дані, такі як ім'я користувача, пароль та адресу електронної пошти.
Авторизація	Надає можливість користувачу увійти до web-додатку, використовуючи свої зареєстровані облікові дані. Цей варіант використання забезпечує автентифікацію та надає доступ до облікового запису користувача.
Відправка повідомлень	Надає користувачу можливість надсилати повідомлення окремим особам або групам. Цей варіант використання включає створення повідомлення, вибір одержувачів і надсилання повідомлення. Він також включає такі функції, як прикріплення файлів, смайликів та форматування тексту.
Керування контактами	Надає користувачу можливість керувати своїм списком контактів, що включає додавання нових контактів, видалення існуючих. Цей варіант використання полегшує ефективну комунікацію завдяки впорядкуванню контактів.
Створення груп	Надає користувачу можливість створювати групи для групового обміну повідомленнями шляхом додавання інших користувачів до існуючого діалогу.
Перегляд інформації на головній сторінці	Надає користувачу можливість бачити список існуючих контактів та діалогів.
Налаштування профілю	Надає користувачу можливість налаштувати свій профіль, додавши фотографію профілю та оновивши особисту інформацію. Цей варіант використання дозволяє користувачам персоналізувати свій профіль

На рисунку 2.3 представлено діаграму варіантів використання web-додатку обміну повідомленнями в нотації UML.

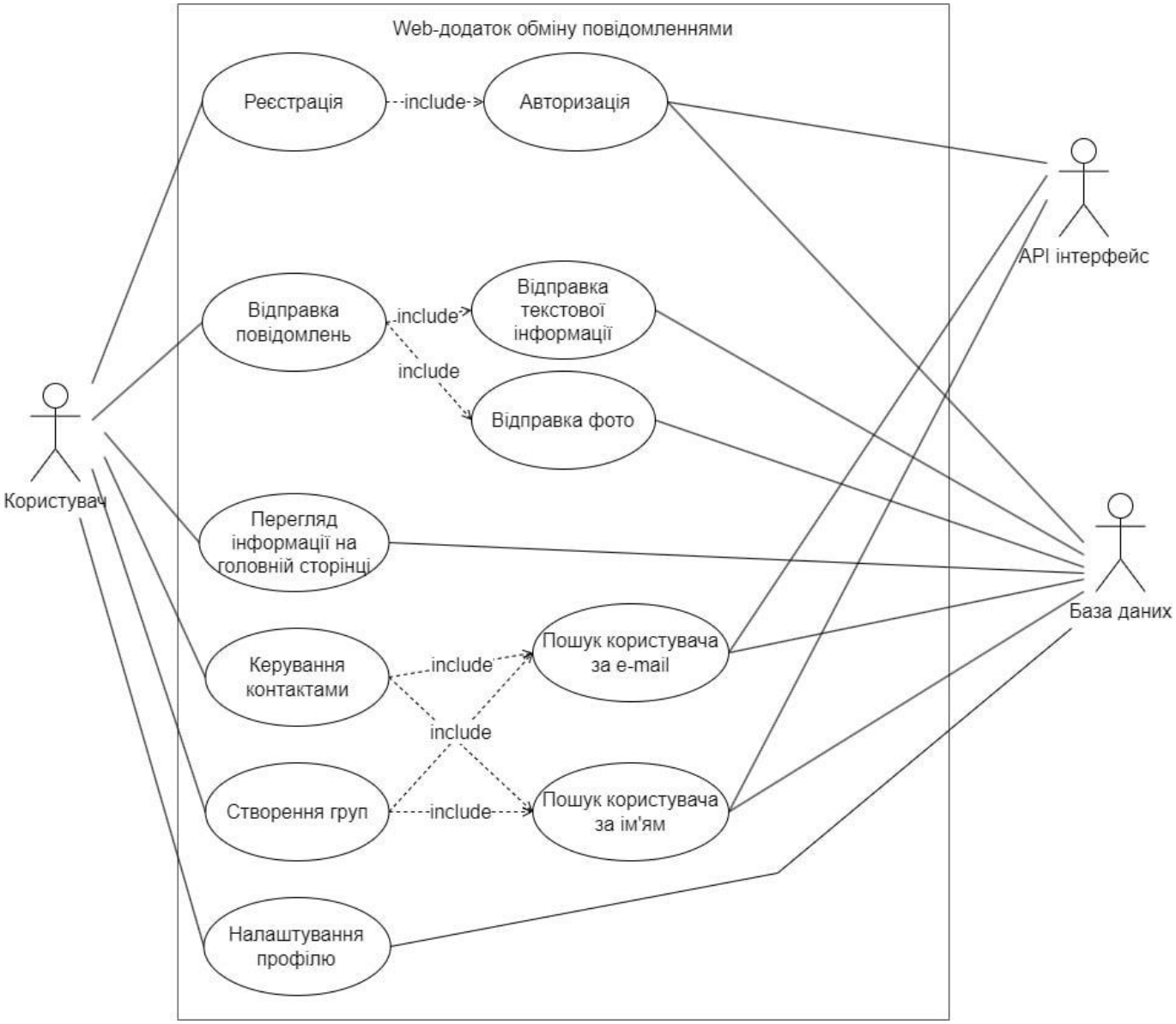


Рисунок 2.3 – Діаграма варіантів використання web-додатку

2.3 Моделювання діаграм діяльності та послідовності

У нотації UML діаграма діяльності - це поведінкова діаграма, яка представляє потік дій або процесів у системі або конкретному модулі [9]. Вона візуалізує послідовні та паралельні дії, точки прийняття рішень та потоки управління, що беруть участь у певному процесі. Діаграми діяльності використовують різні символи, включаючи вузли, ребра, дії, рішення і плаваючі площини, щоб зобразити кроки і

взаємозв'язки між різними видами діяльності. Вони використовуються для робочих процесів програмного забезпечення та поведінки системи.

Діаграми діяльності модулів web-додатку обміну повідомленнями показано на рисунках 2.4 – 2.6.

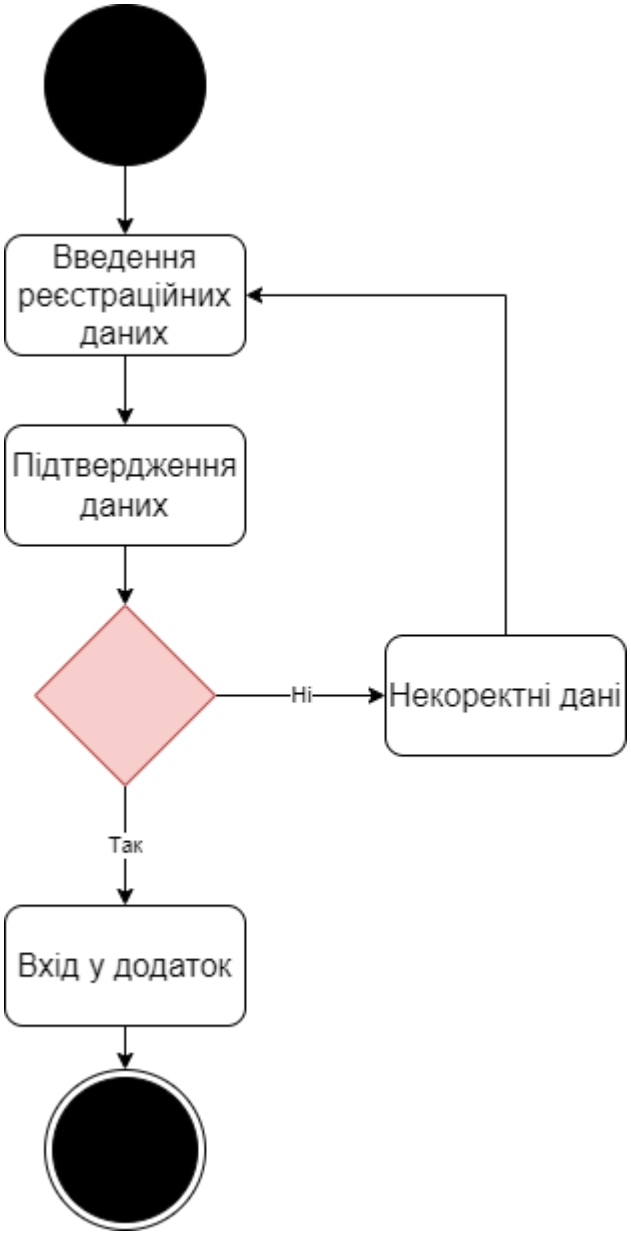


Рисунок 2.4 – Діаграма діяльності модулю реєстрації/авторизації

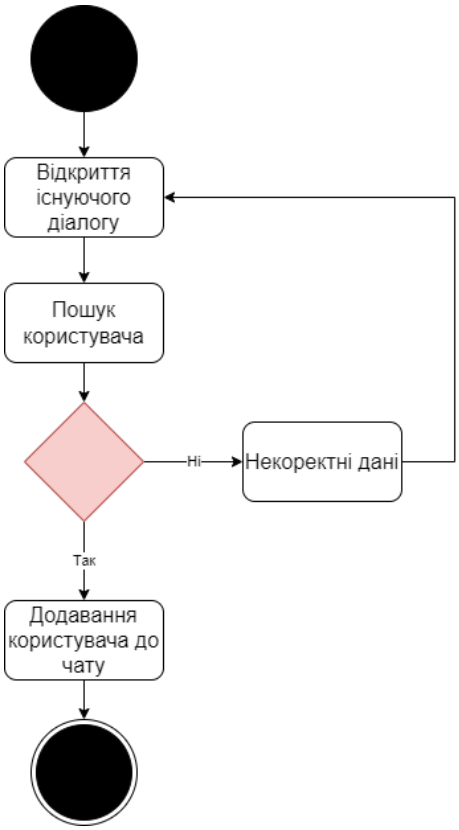


Рисунок 2.5 – Діаграма діяльності модулю додавання користувача до чату

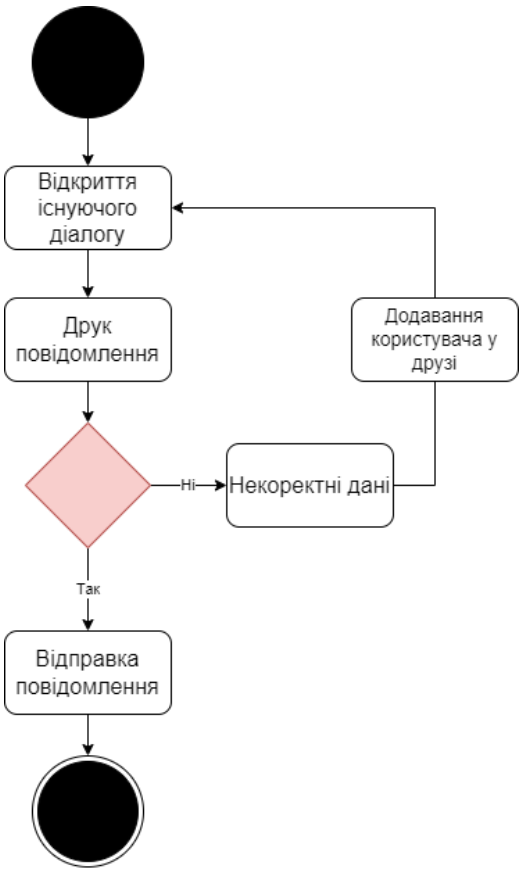


Рисунок 2.6 – Діаграма діяльності модулю відправки повідомлень

Діаграма послідовності - це тип діаграми UML (Unified Modeling Language - уніфікована мова моделювання), яка візуалізує взаємодію та порядок повідомлень, якими обмінюються різні об'єкти або учасники системи. Вона показує потік управління та комунікації між цими об'єктами протягом певного періоду часу [10].

На діаграмі послідовності вертикальні лінії представляють лінії життя або учасників, які можуть бути об'єктами, акторами або компонентами системи. Стрілки або повідомлення використовуються для зображення комунікації або передачі повідомлень між цими учасниками. Послідовність повідомлень відображається на горизонтальній часовій шкалі із зазначенням порядку, в якому вони відбуваються.

Діаграми послідовності зазвичай використовуються для зображення поведінки системи або певного сценарію, демонструючи взаємодію між різними компонентами і те, як вони співпрацюють для досягнення певної функціональності. Вони забезпечують візуальне представлення динамічних аспектів системи і допомагають зрозуміти послідовність подій і потік даних або управління між об'єктами. На рисунку 2.7 показано діаграму послідовності дій користувача.

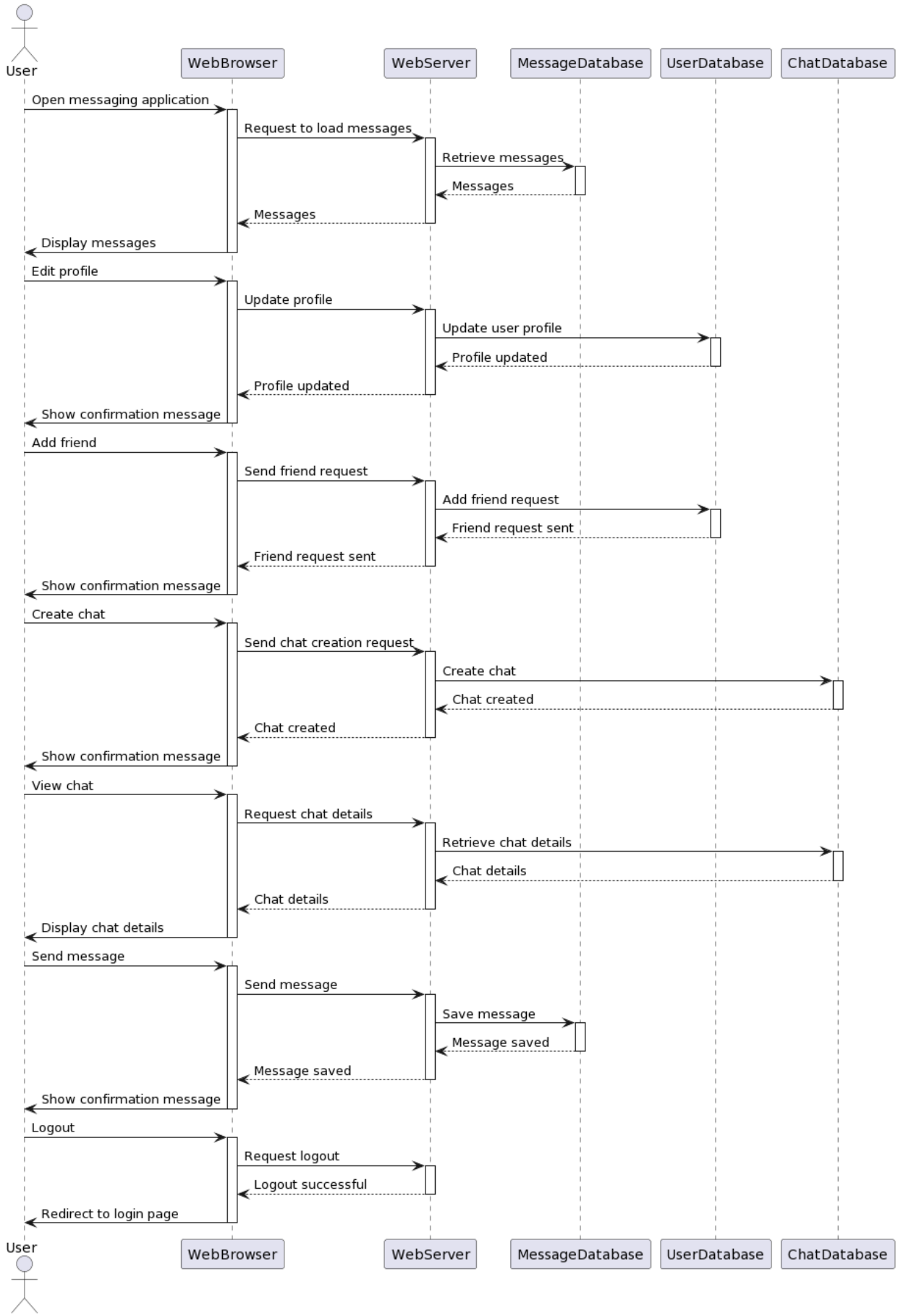


Рисунок 2.7 - Діаграма послідовності дій користувача

2.4 Проектування бази даних web-додатку

Розробка бази даних (БД) є важливою частиною процесу створення програмного забезпечення. Це означає проектування і налагодження структури даних, яка забезпечує ефективне зберігання і керування інформацією [11].

Основні кроки у розробці бази даних:

1. Визначення вимог: Розуміння потреб користувачів і вимог до даних, що будуть зберігатись в БД. Це включає аналіз функціональних і нефункціональних вимог, визначення сутностей та взаємозв'язків між ними.

2. Проектування схеми даних: Розробка схеми бази даних, яка визначає структуру даних, таблиці, поля, ключі і зв'язки між ними. Використовуються концептуальна модель (ER-діаграми), логічна модель (сутностно-реляційна модель) і фізична модель (визначення таблиць, типів даних і інших параметрів БД).

3. Вибір системи керування базами даних (СКБД): Визначення, яка СКБД буде використовуватись для реалізації БД. Враховуються вимоги до продуктивності, масштабованості, доступності, безпеки та інших факторів.

4. Реалізація БД: Створення таблиць, визначення полів, ключів, обмежень цілісності та інших аспектів БД. Використовуються SQL-запити або графічні інструменти для створення БД.

5. Налагодження та тестування: Перевірка правильності реалізації БД, виконання тестів на додавання, зміну і видалення даних, перевірка продуктивності та інші аспекти. Виявлені помилки виправляються і БД оптимізується.

6. Заповнення БД початковими даними: Введення початкових даних до таблиць, щоб система була готовою до використання.

7. Розгортання та супровід: Розміщення БД на сервері, забезпечення доступу до неї для користувачів і додатків. Після розгортання ведеться моніторинг, забезпечення резервного копіювання, оптимізація продуктивності та підтримка БД з часом [12].

В результаті проектування було створено базу даних, яка зображена на рисунку 2.8 з такими таблицями:

1. Chats – таблиця, яка містить дані про створені чати користувачів.
2. ChatUsers – таблиця, яка містить дані про користувачів у створених чатах.
3. Users – таблиця, яка містить дані про зареєстрованих користувачів.
4. Messages – таблиця, яка містить дані про надіслані повідомлення, зображення та емодзі.

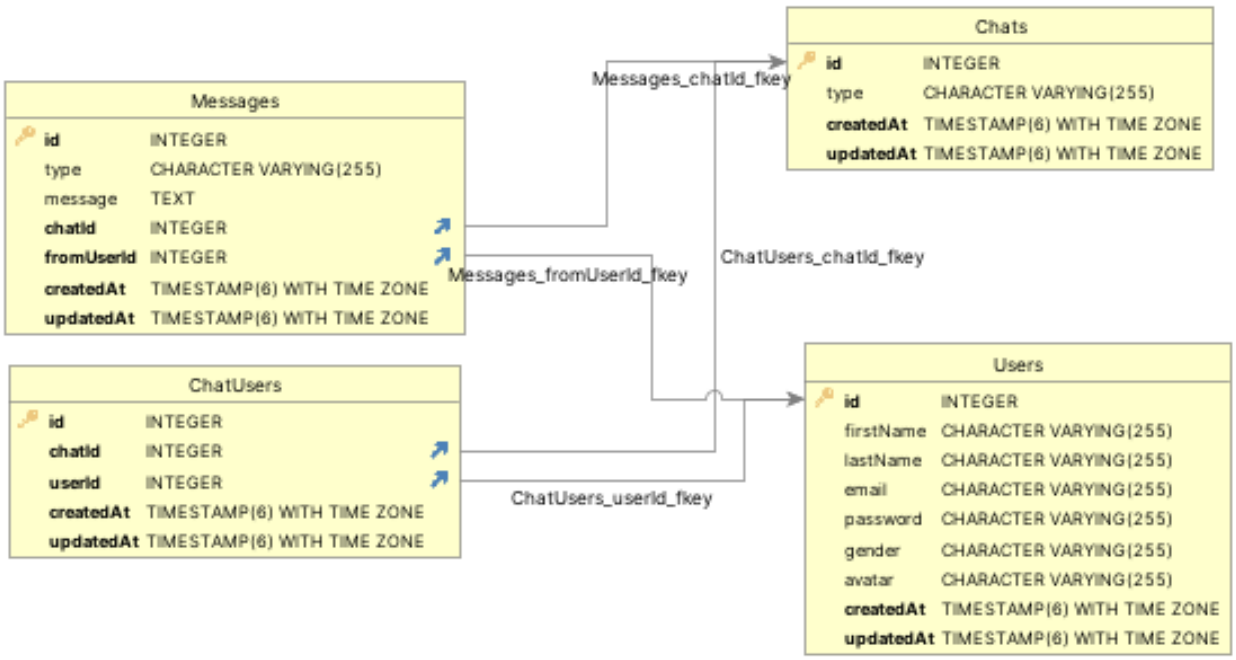


Рисунок 2.8 – Фізична модель бази даних

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ДОДАТКУ ОБМІНУ ПОВІДОМЛЕННЯМИ

3.1 Архітектура web-додатку

Архітектура веб-додатку включає в себе організацію компонентів, їх взаємодію та розподілення функцій для створення працездатної системи. Основні компоненти архітектури веб-додатку включають клієнтську сторону (Front-end), серверну сторону (Back-end) та базу даних.

Під час планування проєкту було обрано клієнт-серверну архітектуру додатку. Клієнт-серверна архітектура – це найпоширеніша модель, в якій клієнтська сторона (web-браузер) взаємодіє з серверною стороною (web-сервер). Клієнтська сторона відповідає за представлення інтерфейсу користувача і взаємодію з ним, а серверна сторона обробляє запити клієнта, забезпечує бізнес-логіку, доступ до бази даних та інші функції.

Основні компоненти клієнт-серверної архітектури:

1. Клієнт – це програмне забезпечення або пристрій, який використовується користувачем для взаємодії з веб-додатком. Це може бути web-браузер, мобільний додаток або API-клієнт. Клієнт відправляє запити до сервера і отримує відповіді з необхідними ресурсами.

2. Сервер – це програмне забезпечення або комп'ютер, який надає ресурси та послуги клієнтам. Сервер обробляє запити, виконує бізнес-логіку, доступ до бази даних, генерує відповіді і передає їх клієнту через мережу.

3. Комунікаційний протокол: для взаємодії між клієнтом і сервером використовується певний комунікаційний протокол, такий як HTTP (Hypertext Transfer Protocol). Запити клієнта та відповіді сервера передаються за допомогою цього протоколу.

4. Ресурси: сервер надає різні ресурси, такі як HTML-сторінки, CSS-стилі, JavaScript-код, зображення, відео, дані з бази даних та інші. Клієнтська сторона отримує ці ресурси і використовує їх для відображення і взаємодії з користувачем.

Клієнт-серверна архітектура дозволяє розділити обов'язки між клієнтом і сервером, забезпечуючи гнучкість, масштабованість та ефективну взаємодію між компонентами. Клієнтська сторона відповідає за інтерфейс користувача і взаємодію з ним, тоді як серверна сторона виконує бізнес-логіку та забезпечує доступ до ресурсів і даних.

Клієнт-серверну архітектуру web-додатку обміну повідомленнями зображено на рисунку 3.1.

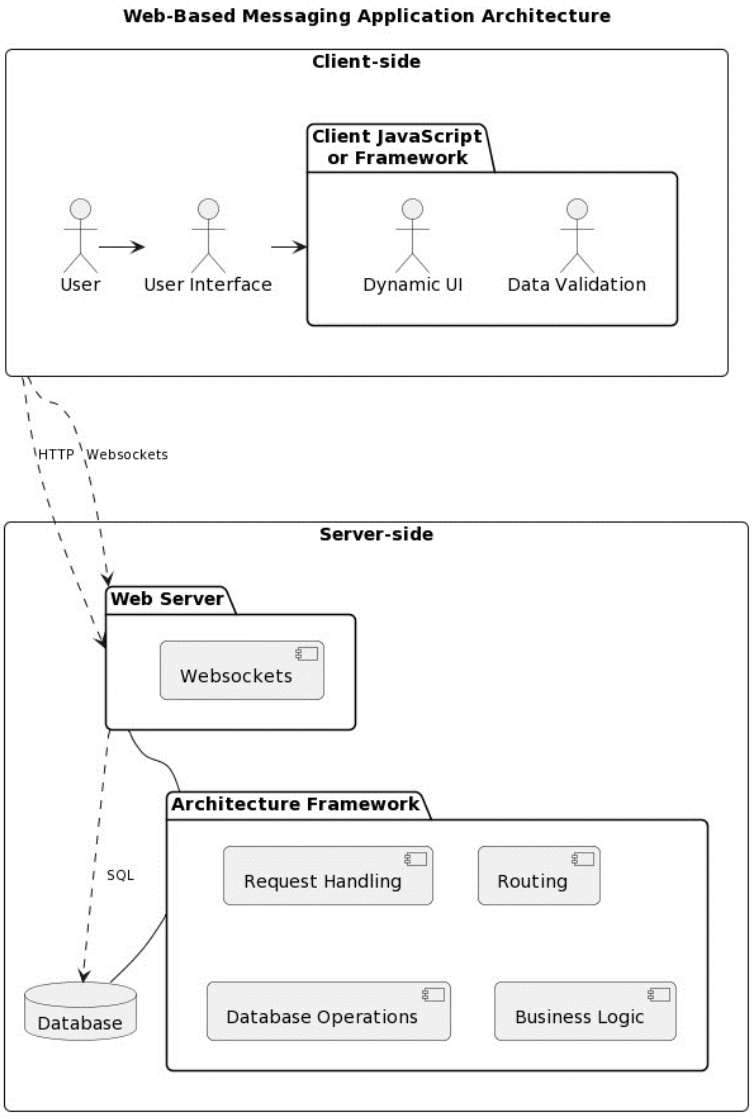


Рисунок 3.1 – Архітектура web-додатку обміну повідомленнями

3.2 Вибір засобів реалізації

Розробка веб-додатку спиратиметься на набір технологій та бібліотек, які в сукупності сприятимуть його успішній реалізації. Для цього проекту було обрано такі технології, як PostgreSQL, Express, React та Node.js, а також бібліотеки: Node-sass, Axios, Antd та Emoji-mart [12].

PostgreSQL слугуватиме системою управління базами даних, забезпечуючи надійну та міцну основу для зберігання та пошуку даних. Її розширені можливості та підтримка складних запитів роблять її придатною для ефективного управління даними додатку.

Express, мінімалістичний та гнучкий фреймворк веб-додатків для Node.js, буде використаний для побудови бекенду додатку. Його простота і зручність використання дозволяють швидко розробляти RESTful API та обробляти HTTP-запити, забезпечуючи ефективний зв'язок між фронтендом і базою даних.

React, популярна бібліотека JavaScript, забезпечить роботу фронтенду веб-додатку. Завдяки своїй компонентній архітектурі React дозволяє створювати багаторазові компоненти інтерфейсу користувача, що призводить до більш модульної та зручної для підтримки кодової бази. Віртуальний DOM React також сприяє ефективному рендерингу та оновленню користувацького інтерфейсу, підвищуючи загальну продуктивність програми.

Node.js, середовище виконання JavaScript, побудоване на JavaScript-движку V8 від Chrome, виступатиме в якості серверної платформи для запуску веб-додатку. Його модель вводу/виводу, керована подіями, що не блокує, робить його дуже масштабованим і придатним для обробки паралельних запитів. Node.js дозволить додатку мати швидкий і чуйний сервер, забезпечуючи безперебійну взаємодію між фронтендом і бекендом.

На додаток до основних технологій було обрано декілька бібліотек, які покращують процес розробки та надають додатковий функціонал. Node-sass дозволить використовувати Sass (Syntactically Awesome Style Sheets) для більш

ефективного та зручного стилю додатку. Axios слугуватиме HTTP-клієнтом для запитів до API, надаючи інтуїтивно зрозумілий інтерфейс та підтримку для роботи з перехоплювачами відповідей та скасуванням запитів. Antd, широко використовувана бібліотека інтерфейсу користувача, запропонує широкий спектр попередньо розроблених компонентів, які можуть бути легко інтегровані в додаток, заощаджуючи час і зусилля розробників. Нарешті, Emoji-mart надасть повний набір емодзі та інтуїтивно зрозумілий користувацький інтерфейс для вибору та відображення емодзі в додатку.

Завдяки ретельному відбору та поєднанню цих технологій і бібліотек, веб-додаток реалізовано з використанням потужного стеку, який має пріоритети ефективності, масштабованості, ремонтопридатності та бездоганного користувацького інтерфейсу.

3.3 Реалізація серверної частини

Розробка серверної частини web-додатку обміну повідомленнями розпочалася з розробки бази даних. БД PostgreSQL для веб-додатку було розроблено за допомогою sequelize, інструменту об'єктно-реляційного відображення (ORM) для Node.js. Sequelize полегшив взаємодію з базою даних, пропонуючи зручний інтерфейс для визначення моделей, виконання запитів та управління зв'язками даних.

Процес розробки бази даних розпочався з ретельного проектування схеми, яка мала на меті точно відобразити структуру даних додатку та задовольнити його вимоги. Sequelize дозволив створити моделі, які відображали відповідні таблиці бази даних. Кожна модель визначала атрибути, типи даних та зв'язки відповідної сутності. Функція міграції sequelize полегшила контроль версій схеми бази даних, забезпечуючи її еволюцію відповідно до мінливих потреб програми. Міграції дозволили безперешкодно модифікувати структуру бази даних без ручного оновлення схеми або ризиків втрати даних. Завдяки використанню можливостей

міграції sequelize, схема бази даних залишалася послідовною та узгодженою з мінливими вимогами додатку.

Sequelize надав повний набір методів і операторів запитів, що спростило такі операції з базою даних, як створення, пошук, оновлення та видалення записів. Ці методи запитів були розроблені, щоб бути виразними і гнучкими, дозволяючи створювати складні запити з простим і інтуїтивно зрозумілим синтаксисом. sequelize відповідав за генерацію SQL запитів, гарантуючи сумісність запитів і оптимальне виконання в різних версіях PostgreSQL [14].

Крім того, Sequelize пропонував вбудовану підтримку для управління зв'язками між моделями, включаючи зв'язки "один-до-одного", "один-до-багатьох" і "багато-до-багатьох". Ця функція спростила управління пов'язаними сутностями даних, забезпечуючи цілісність даних і надаючи зручні методи для запитів до пов'язаних записів. Можливості Sequelize зі створення асоціацій відіграли вирішальну роль у встановленні та підтримці зв'язків між різними сутностями в базі даних.

В процесі розробки для створення та міграції бази даних використовувались наступні команди:

1. sequelize-cli init: Ця команда ініціалізувала конфігурацію Sequelize, створивши необхідні файли та каталоги для керування міграцією бази даних, моделями та початковими даними.

2. sequelize-cli model:generate: Використовується для створення нового файлу моделі, який визначає атрибути та зв'язки для конкретної сутності.

3. sequelize-cli db:migrate: Виконує міграцію бази даних, застосовуючи всі очікувані зміни схеми і синхронізуючи базу даних з останніми визначеннями моделі.

4. sequelize-cli db:seed:all: Запускає спеціальний клас, заповнюючи базу даних початковими або тестовими даними.

Ці команди, надані інтерфейсом командного рядка Sequelize (CLI), є основою процесу розробки бази даних, що дозволяє ефективно створювати, змінювати і синхронізувати схему бази даних.

Таким чином, розробка бази даних PostgreSQL з використанням Sequelize полегшила створення та управління схемою бази даних, зв'язками даних та операціями запитів. Можливості міграції Sequelize забезпечили послідовну еволюцію схеми бази даних, а підтримка асоціацій спростила роботу з пов'язаними сутностями даних. Використання Sequelize спростило процес розробки бази даних, що дозволило створити ефективну та надійну основу для веб-додатку.

Наступним кроком було розроблено моделі для взаємодії з базою даних, що представляють структуру і взаємозв'язки даних, що зберігаються в БД. Вони визначають таблиці або колекції в базі даних і включають методи для запитів і маніпулювання даними. Моделі взаємодіють з базою даних і використовуються контролерами для обробки операцій з даними. Приклад моделі User зображено на рисунку 3.2.

```
User.init({
    firstName: DataTypes.STRING,
    lastName: DataTypes.STRING,
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    gender: DataTypes.STRING,
    avatar: {
        type: DataTypes.STRING,
        get() {
            const avatar = this.getDataValue('avatar')
            const url = `${config.appUrl}:${config.appPort}`

            if (!avatar) {
                return `${url}/${this.getDataValue('gender')}.svg`
            }

            const id = this.getDataValue('id')
            return `${url}/user/${id}/${avatar}`
```

Рисунок 3.2 – Приклад фрагменту моделі User

Контролери є важливим компонентом у розробці веб-додатків, оскільки вони відіграють вирішальну роль в управлінні взаємодією з користувачами, обробці запитів та управлінні основною логікою програми. Реалізацію можна розглянути на

прикладі контролера реєстрації та авторизації, фрагмент якого зображено на рисунку 3.3.

Рисунок 3.3 – Фрагмент коду створення контролера authController

Даний фрагмент коду відповідає за такі дії:

– якщо користувача не знайдено, повертається код статусу 404 з повідомленням "Користувача не знайдено!";

– якщо користувача знайдено, порівнюється наданий пароль зі збереженим паролем за допомогою методу compareSync з бібліотеки bcrypt;

– якщо паролі збігаються, генерується маркер аутентифікації за допомогою функції generateToken і додається до нього аватар користувача;

– і якщо все вірно, надсилається об'єкт користувача з токеном у відповідь.

Загалом, контролер authController обробляє функціонал входу та реєстрації користувачів, перевіряє паролі, генерує токени аутентифікації та надсилає відповіді на основі результатів операцій.

Для забезпечення взаємодії користувачів у реальному часі було використано Socket.io – це бібліотека JavaScript, яка забезпечує двосторонній зв'язок між веб-браузером і сервером у режимі реального часу. Вона забезпечує рівень абстракції над протоколом WebSockets, що дозволяє створювати ефективні та стійкі з'єднання між клієнтами та серверами. Фрагмент коду зображено на рисунку 3.4.

```
socket.on('add-user-to-group', ({ chat, newChatter }) => {

    if (users.has(newChatter.id)) {
        newChatter.status = 'online'
    }

    // old users
    chat.Users.forEach((user, index) => {
        if (users.has(user.id)) {
            chat.Users[index].status = 'online'
            users.get(user.id).sockets.forEach(socket => {
                try {
                    io.to(socket).emit('added-user-to-group', { chat, chatters: [newChatter] })
                } catch (e) { }
            })
        }
    })
```

Рисунок 3.4 – Фрагмент коду сокету

У даному фрагменті коду описано створення події «add-user-to-group». Ця подія спрацьовує, коли користувача додано до групового чату. Вона сповіщає наявних учасників чату та новоприєднаного користувача.

Отже, даний фрагмент коду демонструє реалізацію обробника подій на основі Socket.IO для web-додатку обміну повідомленнями. А загалом web-сокет обробляє різні події, керує з'єднаннями користувачів і полегшує спілкування між клієнтами в режимі реального часу.

Також для більш надійного зберігання паролів користувачів, було використано бібліотеку bcrypt, яка за допомогою хеш-функції зашифровує пароль. В результаті зашифрований пароль заноситься у базу даних у вигляді набору символів. Фрагменти коду зображені на рисунках 3.5-3.6.

```
$2b$10$7ASTsI1x6amUZtkJ4ZITyu0LSXVOzOR6WoJAQP8ve5CEeGFahjAE.
$2b$10$gM/ui8.fcWuFYUsx/bk1L.f.vW4R7COm0oQkFOCpoTfLm3CCfHCfe
$2b$10$ZwGWXLTokLk5p3ICoP.Ofe5khzZXIcGTYu1iun4HJvuw2nnZI4KmG
```

Рисунок 3.5 – Приклад зашифрованих паролів

```
        sequelize,
        modelName: 'User',
        hooks: {
            beforeCreate: hashPassword,
            beforeUpdate: hashPassword
        }
    });
    return User;
};

const hashPassword = async (user) => {
    if (user.changed('password')) {
        user.password = await bcrypt.hash(user.password, 10)
    }

    return user
}
```

Рисунок 3.6 – Фрагмент коду шифрування паролів

Останнім кроком реалізації серверної частини було налаштування коректної маршрутизації між сторінками web-додатку, код якої зображено на рисунку 3.7.

```
const router = require('express').Router()

router.get('/home', (req, res) => {
    return res.send('Home screen')
})

router.use('/', require('./auth'))
router.use('/users', require('./user'))
router.use('/chats', require('./chat'))

module.exports = router
```

Рисунок 3.5 – Код маршрутизації між сторінками

Даний код налаштовує основний маршрутизатор для web-додатку. Він визначає маршрут «/home», який при зверненні відповідає «Домашній екран». Він також налаштовує підмаршрути для аутентифікації, маршрути, пов'язані з користувачами, і маршрути, пов'язані з чатом.

3.4 Реалізація клієнтської частини

При розробці веб-додатків клієнтська частина відіграє вирішальну роль у забезпеченні інтерактивного та зручного інтерфейсу для взаємодії користувачів з додатком. Цей розділ описує процес розробки клієнтської частини, яка включає в себе інтерфейсні компоненти, користувацький інтерфейс і логіку на стороні клієнта.

Розробка клієнтської частини починається з проектування інтерфейсу користувача (UI) веб-додатку. Дизайн інтерфейсу включає створення візуально привабливих макетів, вибір відповідних кольорів, типографіки та розробку інтуїтивно зрозумілої навігації для безперешкодної роботи користувача. Цей етап включає створення каркасу, прототипування та ітеративні процеси проектування для вдосконалення інтерфейсу.

Першим кроком розробки клієнтської частини стало розробка сторінки авторизації користувача, що зображена на малюнку 3.6.

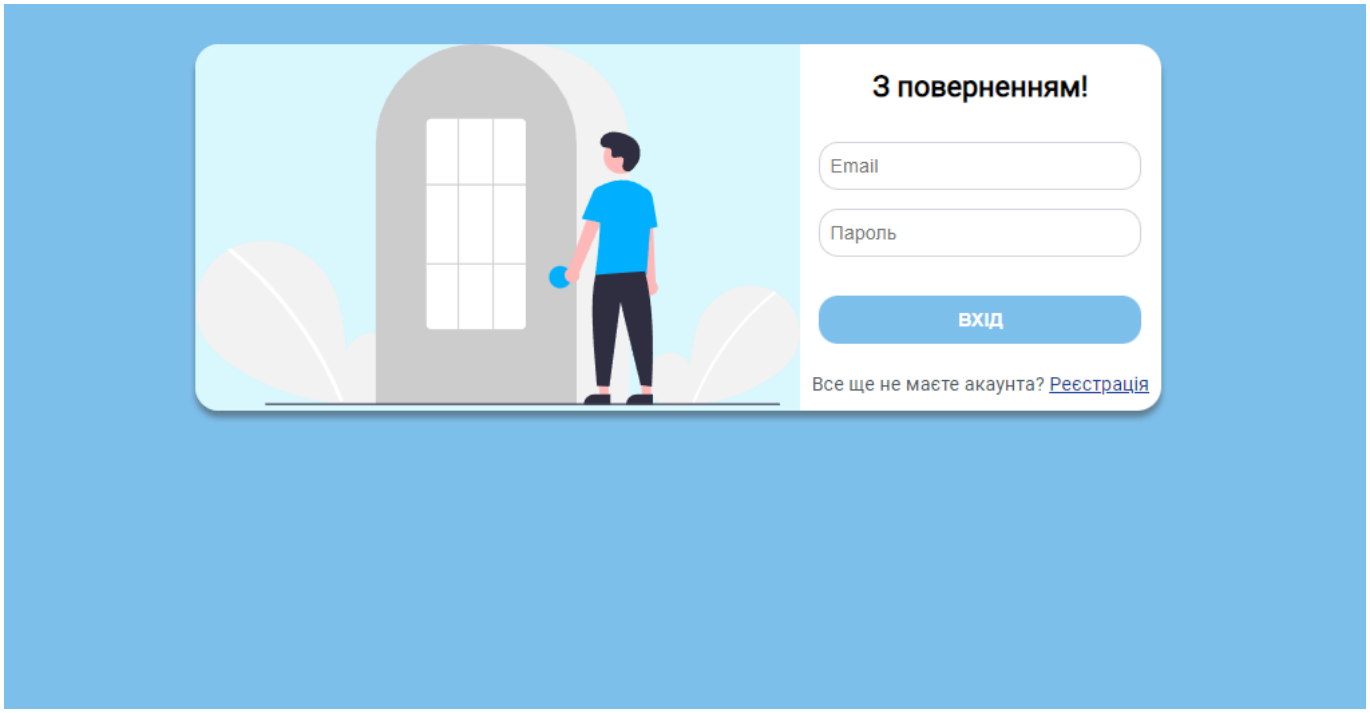


Рисунок 3.6 - Сторінка авторизації користувача

Дана сторінка містить поля email та пароль, які користувач повинен ввести для входу в систему. Якщо користувач не зареєстрований, то він має можливість перейти на сторінку реєстрації.

Авторизацію було розроблено за допомогою React, HTML та Sass, фрагменти коду яких представлені на рисунках 3.7-3.8 [15].

```
<div id='auth-container'>
    <div id='auth-card'>
        <div className='card-shadow'>
            <div id='image-section'>
                <img src={loginImage} alt='Login' />
            </div>

            <div id='form-section'>
                <h2>З поверненням!</h2>

                <form onSubmit={submitForm}>
                    <div className='input-field mb-1'>
                        <input
                            onChange={e => setEmail(e.target.value)}
                            value={email}
                            required='required'
                            type='text'
                            placeholder='Email' />
                    </div>
```

Рисунок 3.7 – Фрагмент HTML коду сторінки авторизації

Стилі для даної сторінки були розроблені за допомогою Sass (Syntactically Awesome Style Sheets) - це препроцесор CSS, який розширює можливості за допомогою таких функцій, як змінні, вкладеність, міксини та функції. Він забезпечує більш організований та ефективний спосіб написання таблиць стилів, роблячи код CSS більш зручним для підтримки та повторного використання [16].

За допомогою Sass можна визначати змінні для зберігання та повторного використання значень у таблиці стилів, що зменшує потребу в повторюваному коді. Вкладеність дозволяє створити більш інтуїтивно зрозумілу структуру, де дочірні селектори вкладені в батьківські, що покращує читабельність.

```scss
&>div {
    display: flex;
    flex-direction: column;
    min-width: 100%;
    margin-top: 10px;
    background-color: white;
    border-radius: 20px;

    @media (min-width: 800px) {
        flex-direction: row;
        min-width: 500px;
        margin-top: 2.5rem;
    }

    #image-section {
        background-color: $bg-light;
        border-top-left-radius: 20px;
        border-top-right-radius: 20px;
        border-bottom-left-radius: 0;
```

Рисунок 3.8 – Фрагмент SCSS стилів

Далі було розроблено сторінку реєстрації (див. рис. 3.9), на яку користувач може перейти, якщо не зареєстрований.

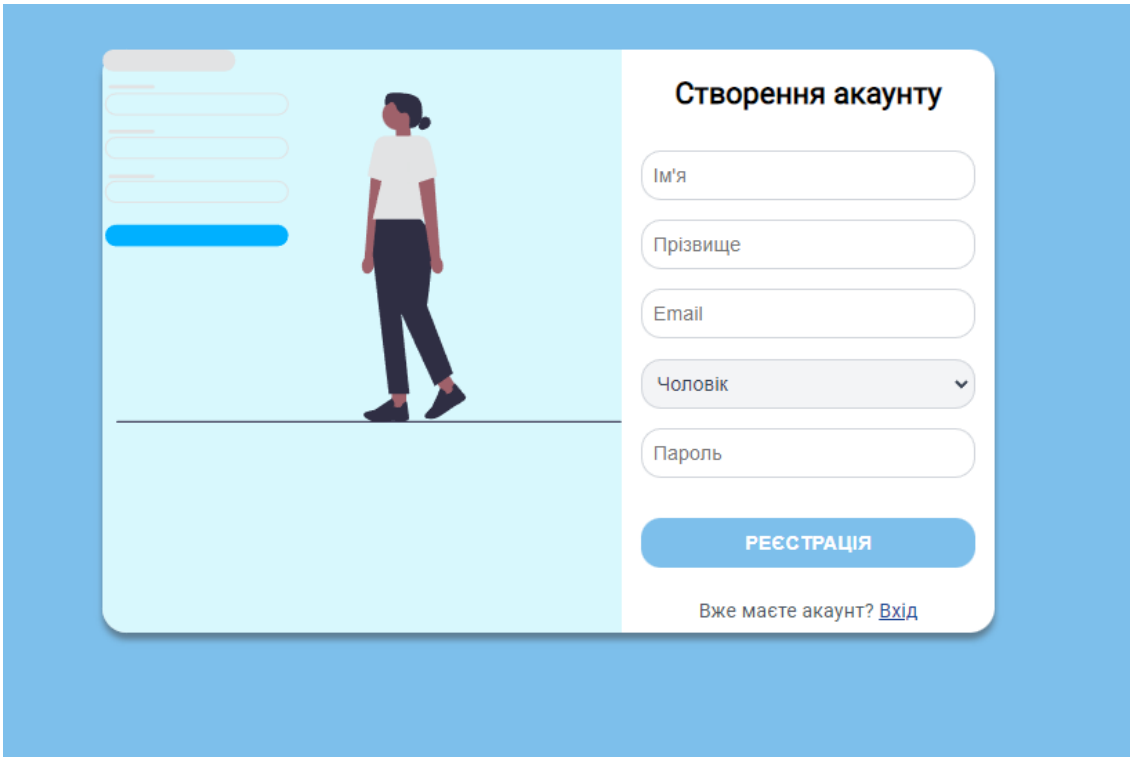


Рисунок 3.9 – Сторінка реєстрації користувача

Після реєстрації користувач автоматично переходить на головну сторінку web-додатку, зображення та фрагменти коду відображено на рисунку 3.10-3.12.

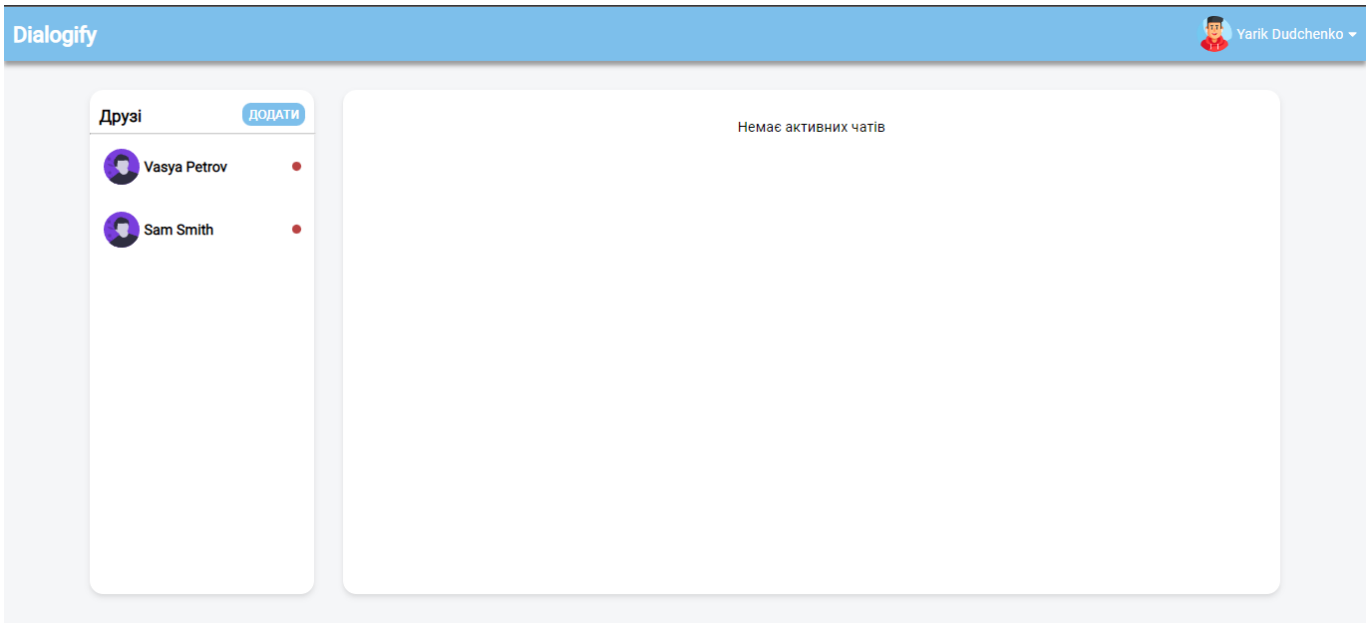


Рисунок 3.10 – Головна сторінка web-додатку

```
    return (
        <div id='chat-container'>
            <Navbar />
            <div id='chat-wrap'>
                <FriendList />
                <Messenger />
            </div>
        </div>
    );
}
```

Рисунок 3.11 – Фрагмент HTML-коду головної сторінки

```scss
#chat-container {
    display: flex;
    flex-direction: column;
    position: absolute;
    top: 0;
    bottom: 0;
    right: 0;
    left: 0;
    background-color: $bg-gray;
    height: 100%;
    margin: 0 !important;

    #chat-wrap {
        display: flex;
        flex-direction: column;
        flex-grow: 1;
        overflow: hidden;
```

Рисунок 3.12 – Фрагмент SCSS стилів головної сторінки

Головна сторінка має такі блоки:

1. Navbar – верхня частина сторінки, на якій відображено назву web-додатку зліва та аватар з прізвищем та ім’ям справа з випадаючим списком, у якому можна або редагувати профіль, або вийти із додатку, фрагменти якого зображені на рисунках 3.13-3.16.

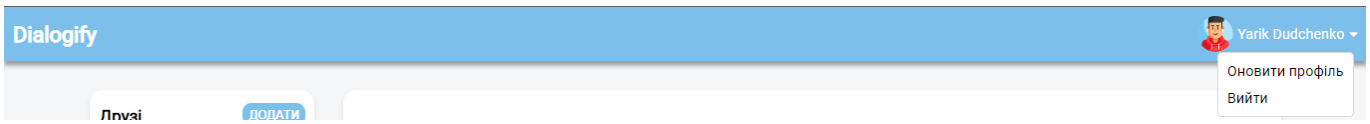


Рисунок 3.13 – Верхня частина сторінки

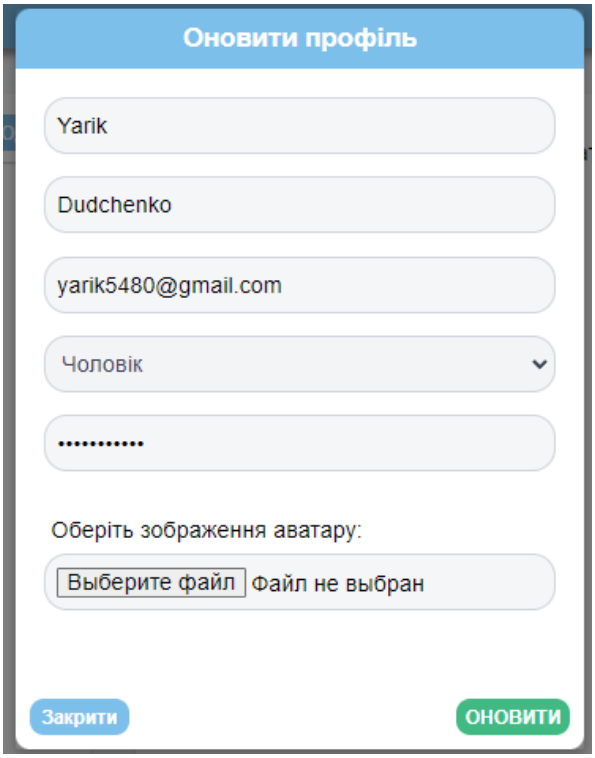


Рисунок 3.14 – Модальне вікно оновлення профілю

```
return (
    <div id='navbar' className='card-shadow'>
        <h2>Dialogify</h2>
        <div onClick={() => setShowProfileOptions(!showProfileOptions)} id='profile-menu'>
            <img width="40" height="40" src={user.avatar} alt='Avatar' />
            <p>{user.firstName} {user.lastName}</p>
            <FontAwesomeIcon icon='caret-down' className='fa-icon' />

            {
                showProfileOptions &&
                <div id='profile-options'>
                    <p onClick={() => setShowProfileModal(true)}>Оновити профіль</p>
                    <p onClick={() => dispatch(logout())}>Вийти</p>
                </div>
            }
```

Рисунок 3.15 – Фрагмент HTML-коду верхньої частини сторінки

```scss
#profile-menu {
    display: flex;
    justify-content: flex-end;
    align-items: center;
    height: 100%;
    position: relative;
    cursor: pointer;

    p {
        display: flex;
        align-self: center;
        color: white;
    }

    img {
        display: flex;
        color: white;
        align-self: center;
        margin-right: 5px;
```

Рисунок 3.16 – Фрагмент SCSS стилів верхньої частини сторінки

2. Friends – блок, який містить доданих друзів, у якому відображається ім’я та прізвище користувача та останнє надіслане або відправлене повідомлення, який зображено на рисунку 3.17.

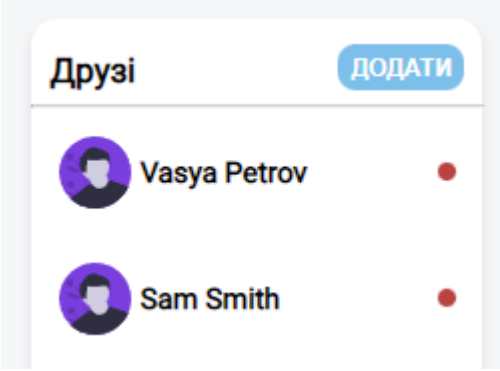


Рисунок 3.17 – Блок з друзями

Також цей блок має кнопку «Додати», після натискання якої відкривається модальне вікно, зображене на рисунку 3.18 в якому користувач має змогу знайти та додати друзів.

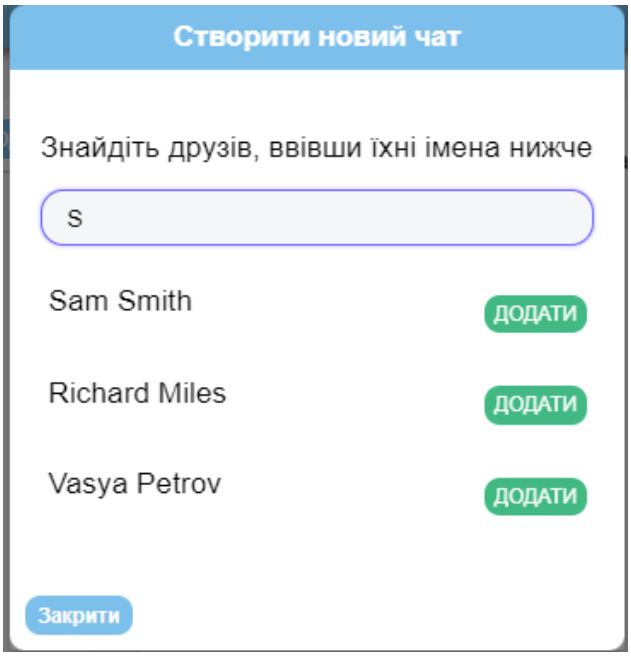


Рисунок 3.18 – Модальне вікно додавання друзів

3. Messenger – блок, який за замовчуванням відображає напис «Немає активних чатів» (див. рис. 3.19), але якщо користувач обере будь-якого друга, то користувач отримає можливість надсилати повідомлення (див. рис. 3.20).

Рисунок 3.19 – Неактивний блок чату

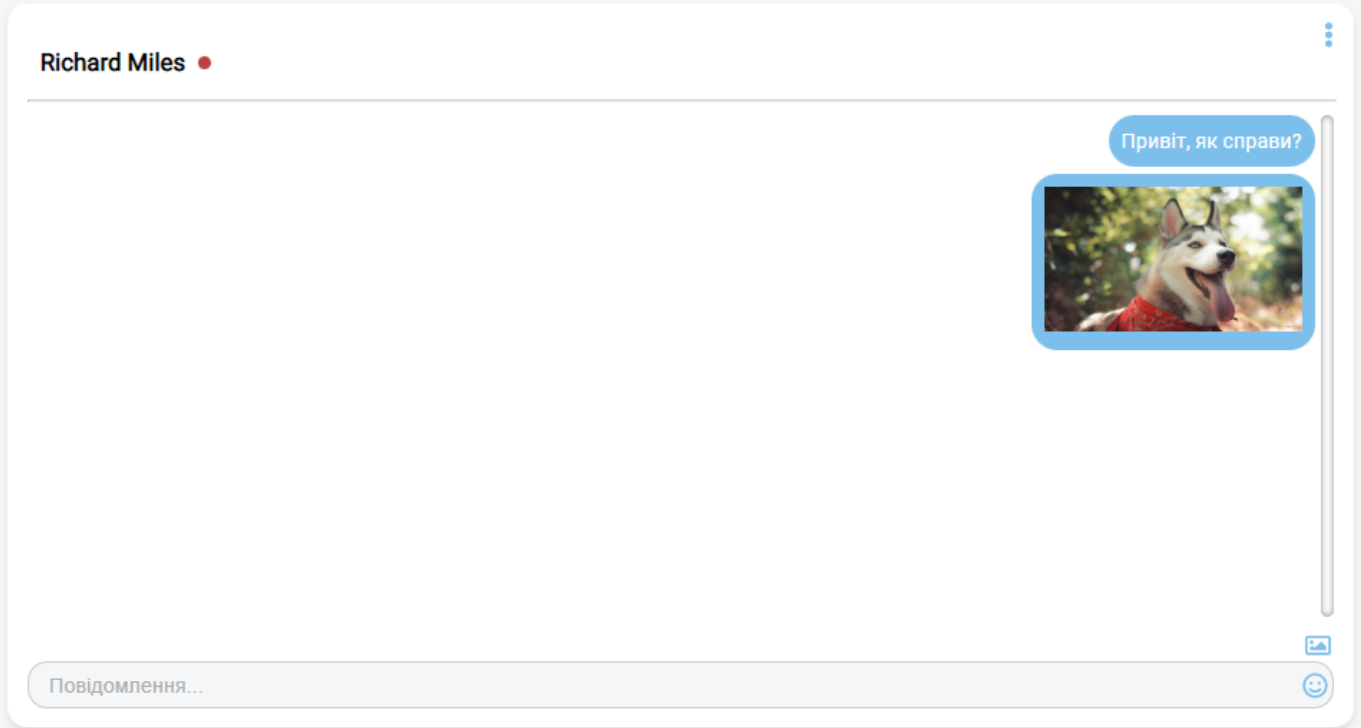


Рисунок 3.20 – Активний блок чату з користувачем

Також блок Messenger має поле для вводу повідомлення з можливістю надсилати зображення, зображення та фрагменти коду якого відображено на рисунках 3.21-3.22.

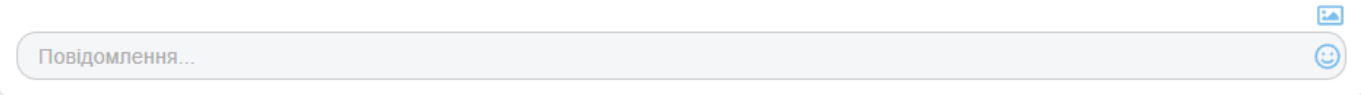


Рисунок 3.21 – Поле для вводу повідомлень

```
<div id='message-input'>
    <input
        ref={msgInput}
        value={message}
        type='text'
        placeholder='Повідомлення...'
        onChange={e => handleMessage(e)}
        onKeyDown={e => handleKeyDown(e, false)}
    />
    <FontAwesomeIcon
        onClick={() => setShowEmojiPicker(!showEmojiPicker)}
        icon={['far', 'smile']}
        className='fa-icon'
    />
</div>

<input id='chat-image' ref={fileUpload} type='file' onChange={e => setImage(e.target.files[0])} />
```

Рисунок 3.22 – Фрагмент HTML-коду для вводу повідомлень

Останнім кроком реалізації клієнтської частини стало реалізація обробки помилок, приклад та фрагменти коду яких зображено на рисунках 3.23-3.25. Якщо користувач вводить некоректну інформацію або користувач доданий вже до чату і т.д, то виводяться повідомлення про помилку або успішність.

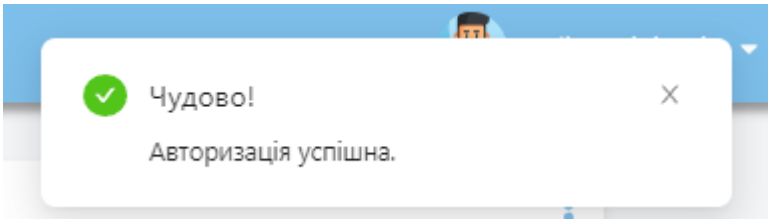


Рисунок 3.23 – Повідомлення про успішну реєстрацію

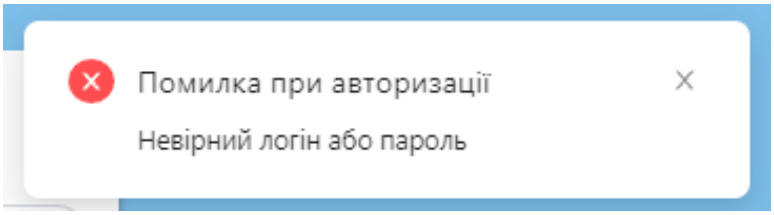


Рисунок 3.24 – Повідомлення при невірних даних при авторизації

```
register: (data) => {
    return API.post('/register', data)
        .then(({ data }) => {
            setHeadersAndStorage(data);
            openNotification({
                title: 'Чудово!',
                text: 'Реєстрація успішна.',
                type: 'success',
            });
            return data;
        })
        .catch(err => {
            openNotification({
                title: 'Помилка при реєстрації',
                text: 'Невірні дані для реєстрації',
                type: 'error',
              });
            throw err
```

Рисунок 3.25 – Фрагмент коду обробки помилок

ВИСНОВКИ

Web-додаток для обміну повідомленнями, розроблений у цій дипломній роботі, надає користувачам зручний засіб для обміну текстовими повідомленнями. Додаток був розроблений з орієнтацією на користувача, з акцентом на простоту використання та доступність.

Процес розробки включав широке дослідження та аналіз існуючих додатків для обміну повідомленнями, після чого було розроблено та впроваджено функціональну архітектуру, включаючи схему бази даних. Використання сучасних веб-технологій та кращих практик у розробці забезпечує швидку та ефективну роботу додатку.

Загалом, додаток відповідає всім функціональним і нефункціональним вимогам, викладеним у початкових специфікаціях. Він дозволяє користувачам реєструватися, входити в систему, надсилати і отримувати повідомлення, переглядати повідомлення, обмінюватися файлами зображень, а також бачити онлайн/офлайн-статус інших користувачів. Додаток також має чуйний та інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко орієнтуватися та користуватися різними функціями.

Подальша робота може включати розробку додаткових функцій, таких як видалення повідомлень, голосові та відеодзвінки, а також подальше покращення його безпеки та продуктивності. Додаток може бути розширений для підтримки групових чатів, стікерів, відправки файлів різних форматів тощо. Також можуть бути виконані поліпшення у відповідності до зворотного зв'язку користувачів та змін у вимогах до зручності використання та ефективності додатку.

В цілому, розробка веб-додатку для обміну повідомленнями є важливим етапом у поліпшенні комунікаційних можливостей користувачів та забезпеченні зручного та ефективного інструменту обміну інформацією. Додаток в майбутньому планується впровадити у Сумській філії Харківського університету внутрішніх справ, що підтверджується наданим актом у додатку В.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ogundeyi K. E., Yinka-Banjo C. WebSocket in real time application. *Nigerian journal of technology*. 2019. Vol. 38, no. 4. P. 1010. URL: https://doi.org/10.4314/njt.v38i4.26 (дата звернення: 08.05.2023).

2. Johnny Botha, Carien van 't Wout, Louise Leenen. A comparison of chat applications in terms of security and privacy. 2019. URL: https://www.researchgate.net/publication/334537058_A_Comparison_of_Chat_Applications_in_Terms_of_Security_and_Privacy/references (дата звернення: 08.05.2023).

3. Voice chat web app using webrtc / A. Tiwari et al. *IJRDO -journal of computer science engineering*. 2023. Vol. 8, no. 11. P. 23–28. URL: https://doi.org/10.53555/cse.v8i11.5444 (дата звернення: 08.05.2023).

4. Порівняння популярних корпоративних месенджерів | HURMA. *HURMA*. URL: https://hurma.work/blog/porivnyannya-populyarnih-korporativnih-mesendzheriv/ (дата звернення: 10.05.2023).

5. What is IDEF - Definition, Methods, and Benefits - Edraw. *[OFFICIAL] Edraw Software: Unlock Diagram Possibilities*. URL: https://www.edrawsoft.com/what-is-idef.html (дата звернення: 10.05.2023).

6. HDR D. N. F. Equivalent IDEF0 and SysML models. URL: https://www.linkedin.com/pulse/equivalent-idef0-sysml-models-dr-nicolas-figay (дата звернення: 10.05.2023).

7. What is a context diagram and how do you use it? | MiroBlog. *MiroBlog*. URL: https://miro.com/blog/context-diagram/#:~:text=A%20context%20diagram%20outlines%20how,to%20improve%20an%20existing%20system. (дата звернення: 11.05.2023).

8. What is a Context Diagram (and How Can You Create One)? - Venngage. *Venngage*. URL: https://venngage.com/blog/context-diagram/ (дата звернення: 12.05.2023).

9. UML - Overview. *Online Courses and eBooks Library*. URL: https://www.tutorialspoint.com/uml/uml_overview.htm (дата звернення: 13.05.2023).

10. What is Unified Modeling Language (UML)?. *Ideal Modeling & Diagramming Tool for Agile Team Collaboration*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (дата звернення: 12.05.2023).

11. What is PostgreSQL. *PostgreSQL Tutorial – Comprehensive Postgresql Tutorial*. URL: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/ (дата звернення: 18.05.2023).

12. What is PERN Stack? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/what-is-pern-stack/ (дата звернення: 18.05.2023).

13. Free Online Course: PERN Stack Course - Build a Yelp clone (Postgres, Express, React, Node.js) from freeCodeCamp | Class Central. *Class Central*. URL: https://www.classcentral.com/course/freecodecamp-pern-stack-course-build-a-yelp-clone-postgres-express-react-node-js-57824 (дата звернення: 18.05.2023).

14. Sequelize v6 | Sequelize. *Sequelize | Feature-rich ORM for modern TypeScript & JavaScript*. URL: https://sequelize.org/docs/v6/ (дата звернення: 09.05.2023).

15. Quick start – react. URL: https://react.dev/learn (дата звернення: 08.05.2023).

16. Sass: Sass Basics. *Sass: Syntactically Awesome Style Sheets*. URL: https://sass-lang.com/guide (дата звернення: 09.05.2023).

17. JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата звернення: 09.05.2023).

18. Node.js Підручник. Уроки для початківців. W3Schools українською. *HTML CSS JavaScript PHP. W3Schools українською. Уроки для початківців*. URL: https://w3schoolsua.github.io/nodejs/index.html#gsc.tab=0 (дата звернення: 09.05.2023).

19. Express routing. *Express - Node.js web application framework*. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 09.05.2023).

20. Wikiwand - Технічне завдання. *Wikiwand*. URL: https://www.wikiwand.com/uk/Технічне_завдання (дата звернення: 09.05.2023).

21. Gantt Chart: The Ultimate Guide (Definitions & Examples). *ProjectManager*. URL: https://www.projectmanager.com/guides/gantt-chart (дата звернення: 09.05.2023).

22. Огляд робочих структур проекту. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/uk-ua/dynamics365/project-operations/prod-pma/work-breakdown-structures (дата звернення: 09.05.2023).

23. Organization Breakdown Structure (OBS) - PSA - EN. *PSA - EN*. URL: https://uplandsoftware.com/psa/resources/glossary/organization-breakdown-structure-obs/ (дата звернення: 09.05.2023).

ДОДАТОК А
Технічне завдання

ТЕХНІЧНЕ ЗАВДАННЯ
на розробку інформаційної системи «Web-додаток обміну повідомленнями»

ПОГОДЖЕНО:
к.т.н. кафедри інформаційних технологій
_____ Бойко О.В.

Студент групи ІТ-92-0/2
_____ Дудченко Я.А.

Суми 2023

1. Призначення й мета створення web-додатку

1.1 Призначення web-додатку

Web-додаток обміну повідомленнями призначений для того, щоб дозволити користувачам обмінюватися текстовими повідомленнями через Інтернет в режимі реального часу.

1.2 Мета створення web-додатку

Головна мета проекту – створення якісного web-додатку для обміну текстовими повідомленнями через Інтернет з використанням сучасних технологій web-розробки та дизайну із забезпеченням зручної взаємодії між користувачами.

1.3 Цільова аудиторія

Цільовою аудиторією даного проекту є замовник та звичайні користувачі в мережі Інтернет.

2 Вимоги до web-додатку

2.1 Вимоги до web-додатку в цілому

2.1.1 Вимоги до структури й функціонування web-додатку

Web-додаток обміну повідомленнями повинен бути реалізований за допомогою web-інструментів та забезпечувати визначений набір функціональних можливостей.

Кінцевий продукт даного проекту має бути представлений web-додатком, який містить якісне інформаційне наповнення та графічні матеріали.

2.1.2 Вимоги до користувача

Для користування web-додатком не потрібні спеціальні навички. Всі функції доступні через інтуїтивно зрозумілий інтерфейс, який легко освоюється користувачами з базовим рівнем комп'ютерної грамотності. Додатковою вимогою є наявність комп'ютера з підключенням до Інтернету та web-браузера, що також є стандартними для більшості користувачів.

2.1.3 Вимоги до збереження інформації

Уся інформація щодо користувача надана у web-додатку повинна зберігатися у базі даних реалізованій засобами системи управління базами даних PostgreSQL.

2.1.4 Вимоги до розмежування доступу

Розроблюваний web-додаток має бути загальнодоступним у мережі Інтернет. Права доступу лише одні – користувач, який матиме змогу зареєструватися в додатку та авторизовуватися в подальшому для ведення діалогів з іншими учасниками.

2.2 Структура web-додатку

2.2.1 Загальна інформація про структуру web-додатку

До структури web-додатку входять усі його web-сторінки, які є загальнодоступними.

Перелік сторінок web-додатку наступний:

– «Основна сторінка» сторінка містить список діалогів з користувачами, з якої можна обирати необхідну бесіду та вести спілкування, також доступна функція пошуку користувача та перехід в налаштування;

– «Сторінка реєстрації» дозволяє користувачу зареєструватися в системі;

– Форма «Створення діалогу» містить поля для вводу ім’я користувача або e-mail для відправки першого повідомлення;

– «Сторінка авторизації» містить поля для вводу логіну та паролю для входу в додаток.

2.2.2 Управління контентом

Управління контентом web-додатку буде здійснюватися самим користувачем, який матиме змогу надсилати повідомлення; також буде надана можливість відправки зображень, які будуть зберігатися у базі даних.

2.2.3 Дизайн web-додатку

Дизайн web-додатку має бути виконаний у мінімалістичному та сучасному стилі. Корпоративними кольорами є білий, блакитний та палітра відтінків різних кольорів. Тому під час розробки web-додатку треба використовувати саме ці кольори.

Види і розміри шрифтів повинні бути комфортними для перегляду. Інформаційні блоки, графічні матеріали та інші елементи web-сторінок повинні мати зручне і логічне розташування. Шаблон майбутнього програмного продукту зображено на рисунку А.1

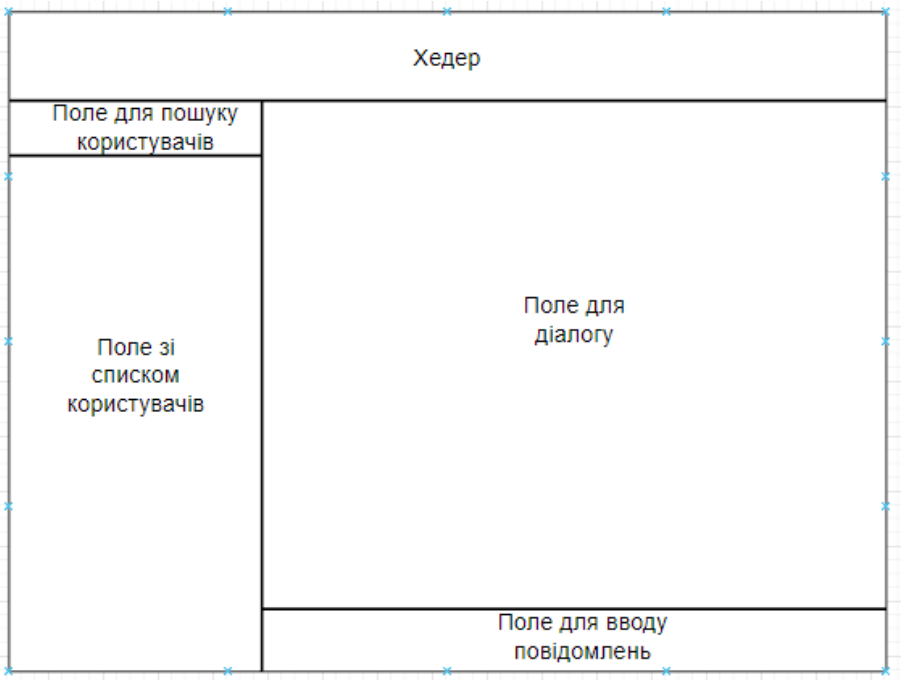


Рисунок А.1 – Схема головної сторінки

2.2.4 Система навігації (карта web-додатку)

Карта web-додатку зображена на рисунку А.2.

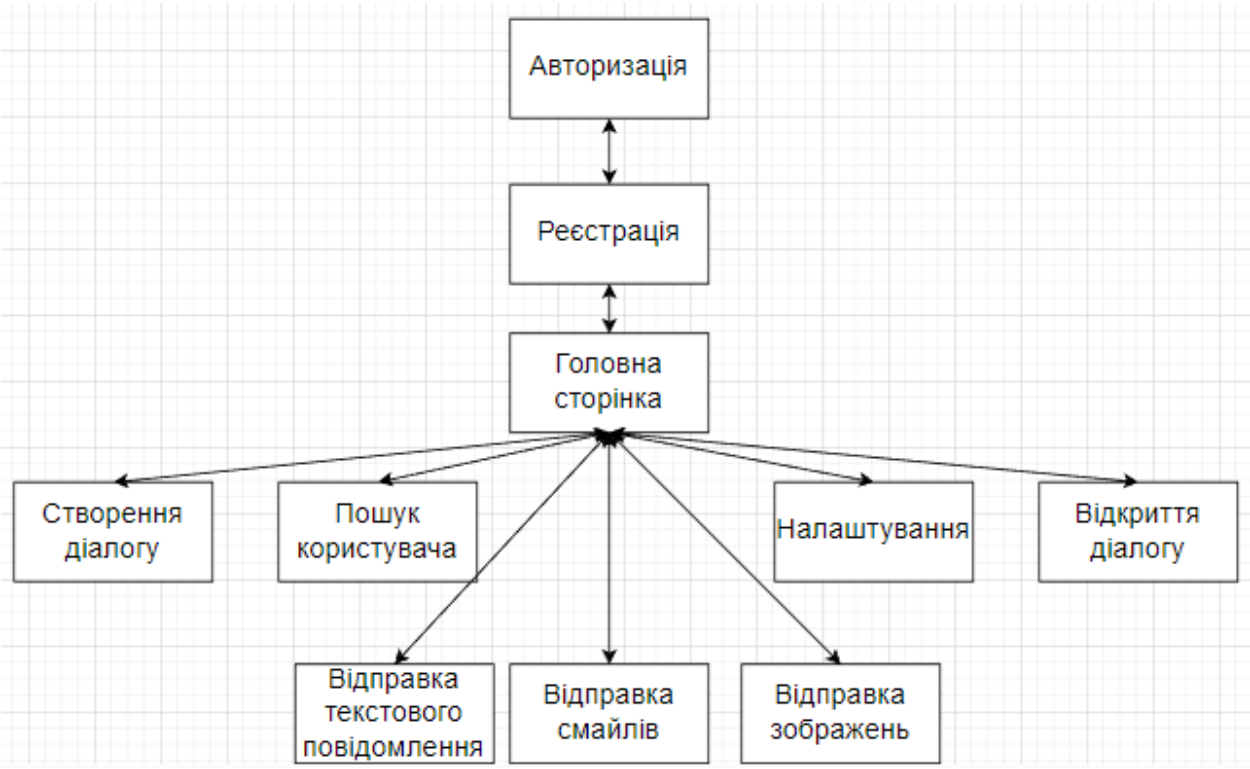


Рисунок А.2 – Система навігації

2.3 Вимоги до функціонування системи

2.3.1 Вимоги до лінгвістичного забезпечення

Весь текст у web-додатку має бути виконаний українською мовою.

2.3.2 Вимоги до програмного забезпечення

Для забезпечення стабільної роботи web-додатку web-браузер має бути Internet Explorer 7.0 і вище, або Firefox 3.5 і вище, або Opera 9.5 і вище, або Safari 3.2.1 і вище, або Chrome 2 і вище.

2.4 Вимоги до функціонування системи

2.4.1 Потреби користувача

Потреби користувача, визначені на основі рішення замовника, представлені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	Потреби користувача	Джерело
UN-01	Перегляд діалогів	Клієнт
UN-02	Створення діалогу	Клієнт
UN-03	Реєстрація та авторизація	Клієнт
UN-04	Відправка зображень	Клієнт
UN-05	Редагування профілю	Клієнт
UN-06	Відправка текстових повідомлень	Клієнт

2.4.2 Системні вимоги

Проаналізувавши потреби користувачів було визначено наступні вимоги:

– наявність реєстрації та авторизації клієнтів;
– пошук інформації про діалоги на web-додатку;
– можливість створення нового діалогу за допомогою форми;
– наявність поля для друку та надсилання повідомлення;
– можливість відправки зображень.

3 Склад і зміст робіт зі створення web-додатку обміну повідомленнями

Детальний опис етапів створення web-додатку наведено в таблиці А.2.

Таблиця А.2 – Етапи створення web-додатку

№	Назва роботи проекту	Тривалість
1	***Розробка web-додатку***	**60 днів**
1.1	**Дослідження предметної області**	**10 днів**
1.1.1	Аналіз предметної області	5 днів
1.1.2	Аналіз існуючих програмних продуктів для вирішення поставленої задачі	5 днів
1.2	**Постановка задачі**	**5 днів**
1.2.1	Формування переліку вимог до програми	3 дні
1.2.2	Вибір засобів реалізації	2 дні
1.3	**Моделювання програмного продукту**	**15 дні**
1.3.1	Моделювання програмного продукту в нотації IDEF0	4 днів
1.3.2	Розробка моделі аналізу	4 днів
1.3.3	Створення моделі проектування	5 днів
1.3.4	Розробка моделі реалізації	2 днів
1.4	Реалізація програмного продукту	21 днів
1.4.1	Верстка сторінок	6 днів
1.4.2	Робота з бек-енд	10 днів

Продовження таблиці А.2

1.4.3	Розробка і впровадження бази даних	5 днів
1.5	Тестування програмного продукту	5 днів
1.5.1	Beta-тестування	3 днів
1.5.2	Alpha-тестування	2 днів
1.6	Публікація на хостинг	1 день
1.7	Написання програмної документації	3 дні

4 Вимоги до складу й змісту робіт із введення web-додатку в експлуатацію

Web-додаток має бути затверджено та розміщено на web-хостингу.

ДОДАТОК Б
Планування робіт

1 Планування змісту структури робіт

Планування змісту та структури робіт - це важливий етап в будь-якому проекті. Він допомагає зрозуміти, які кроки необхідно зробити для досягнення поставленої мети і в якому порядку ці кроки повинні виконуватися. Основні складові планування змісту та структури робіт включають наступне:

1. Аналіз вимог проекту: визначення основних цілей проекту, а також вимог до нього, які мають бути виконані. Це може включати розуміння технічних вимог, бізнес-потреб та вимог замовника.
2. Визначення завдань та підзадач: визначення конкретних завдань та підзадач, необхідних для досягнення мети проекту. Це допомагає розібратися з роботою на більш детальному рівні та підготувати план виконання.
3. Розподіл робіт: розподіл завдань та підзадач між учасниками проекту. Це допомагає кожному члену команди знати, що саме він повинен робити, які вимоги до його завдання та до якого терміну його треба виконати.
4. Визначення термінів виконання: встановлення конкретних термінів виконання кожного завдання та підзадачі. Це допомагає відслідковувати прогрес виконання робіт та забезпечує, що проект буде завершено вчасно.
5. Оцінка ризиків: визначення потенційних ризиків, які можуть виникнути під час виконання проекту та розробка плану їхнього управління.

2 Деталізація мети проєкту методом SMART

Щоб проект був успішним та конкурентоспроможним треба на концептуальному етапі правильно визначити його мету за допомогою SMART-методу. Це дозволяє більш конкретно представити призначення розроблюваного продукту.

Для виконавця даного проекту формат постановки SMART-мети такий:

«*Розробити web-додаток обміну повідомленнями до кінця 4 курсу для спрощення процедури спілкування*».

Таблиця Б.1 – Деталізація мети проекту методом SMART

Specific	Створити функціональний додаток для скорочення часу користувача на обмін повідомленнями
Measurable	Веб-додаток матиме змогу зберігати чат, зображення та емодзі
Achievable	Мета досяжна, є затверджене технічне завдання
Relevant	Для підвищення рейтингу та конкурентоспроможності чату
Time-framed	Термін встановлений – 30 травня 2023 року

3 Планування змісту робіт

WBS (Work Breakdown Structure – Ієрархічна структура робіт) – це графічний вигляд елементів проєкту, які згруповані ієрархією у єдине ціле з продуктом проекту. Ієрархічна структура робіт (WBS) може виглядати як звичайний перелік робіт, які необхідно виконати у межах реалізації проєкту. Ієрархічна структура робіт є планом роботи, проте не роботою самою по собі. Це означає, що ієрархічна структура робіт не лише перераховує завдання проекту, а й поділяє їх на менші взаємозалежні компоненти. Такий підхід робить роботу проекту більш керованою, та дозволяє проектним менеджерам відслідковувати та коректувати обсяг роботи відповідно до внутрішніх та зовнішніх організаційних змін.

Ієрархічна структура робіт допомагає проектному менеджеру скласти графік робіт та гнучко коректувати його. Таким чином, ієрархічна структура Робіт визначає не лише обсяги робіт, їх вартість та необхідні дії по виконанню роботи, а й якісні критерії для оцінювання ефективності проекту після його впровадження. Отже, виступаючи як суттєва частина життєвого циклу проекту та процесу його планування, ієрархічна структура робіт допомагає визначити сукупність напрямів проекту за часом. На рисунку Б.1-Б.2 представлено WBS з розробки web-додатку обміну повідомленнями.

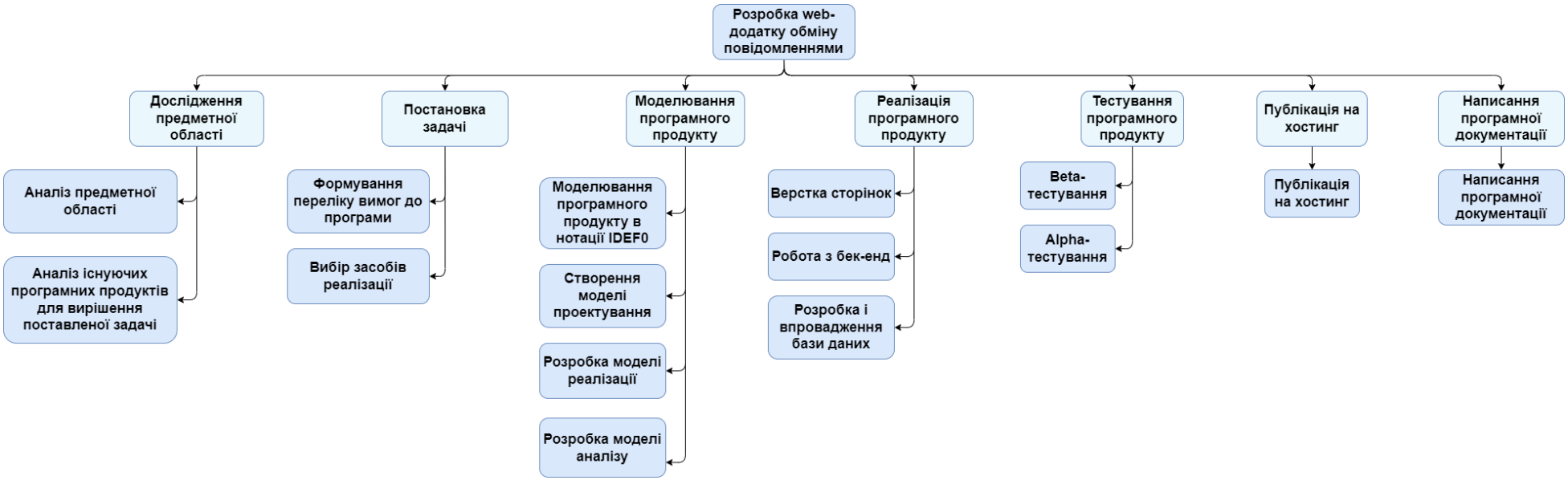


Рисунок Б.1 – WBS - структура робіт проєкту

4 Планування структури виконавців

Наступним етапом після декомпозиції процесів є розробка організаційної структури виконавців або OBS, яка визначається як графічна структура відображення учасників або відповідальних осіб, які беруть участь у реалізації проекту.

У ролі відповідальних осіб виступають співробітники, що відповідають за організацію і виконання елементарної роботи, що зазначена у WBS. Кожну елементарну роботи можна розглядати як окремий проект.

На рисунку Б.2 представлено організаційну структуру планування проекту. Список виконавців, що функціонують в проекті описано в таблиці Б.2.

Таблиця Б.2 – Виконавці проекту

Роль	**Ім’я**	**Проєктна роль**
Розробник	Дудченко Я.А.	Виконує front-end та back-end розробку
Проектувальник	Дудченко Я.А.	Виконує проектування бази даних та розробляє структуру web-додатку.
Тестувальник	Дудченко Я.А.	Відповідає за beta-тестування функціоналу та дизайну web-додатку.
Тестувальник	Бойко О.В.	Відповідає за alpha-тестування функціоналу та дизайну web-додатку.
Керівник проєкту	Бойко О.В.	Відповідає за вибір засобів реалізації.
Менеджер проєкту	Дудченко Я.А.	Відповідає за виконання термінів, розподіл ресурсів та завдань між учасниками. Виконує збір та аналіз даних.

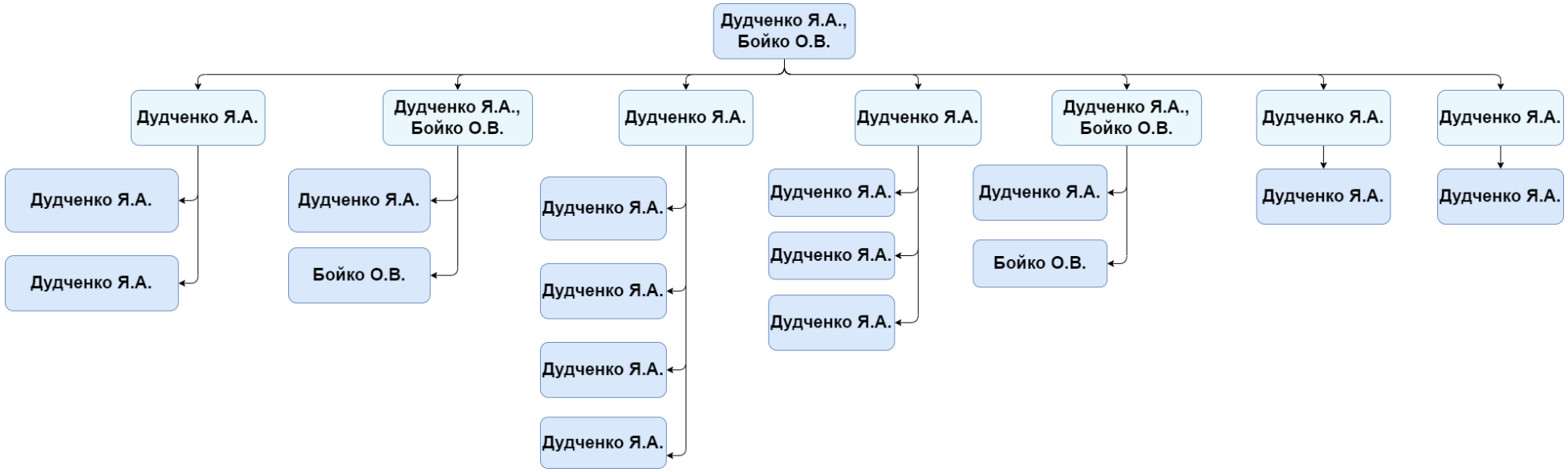


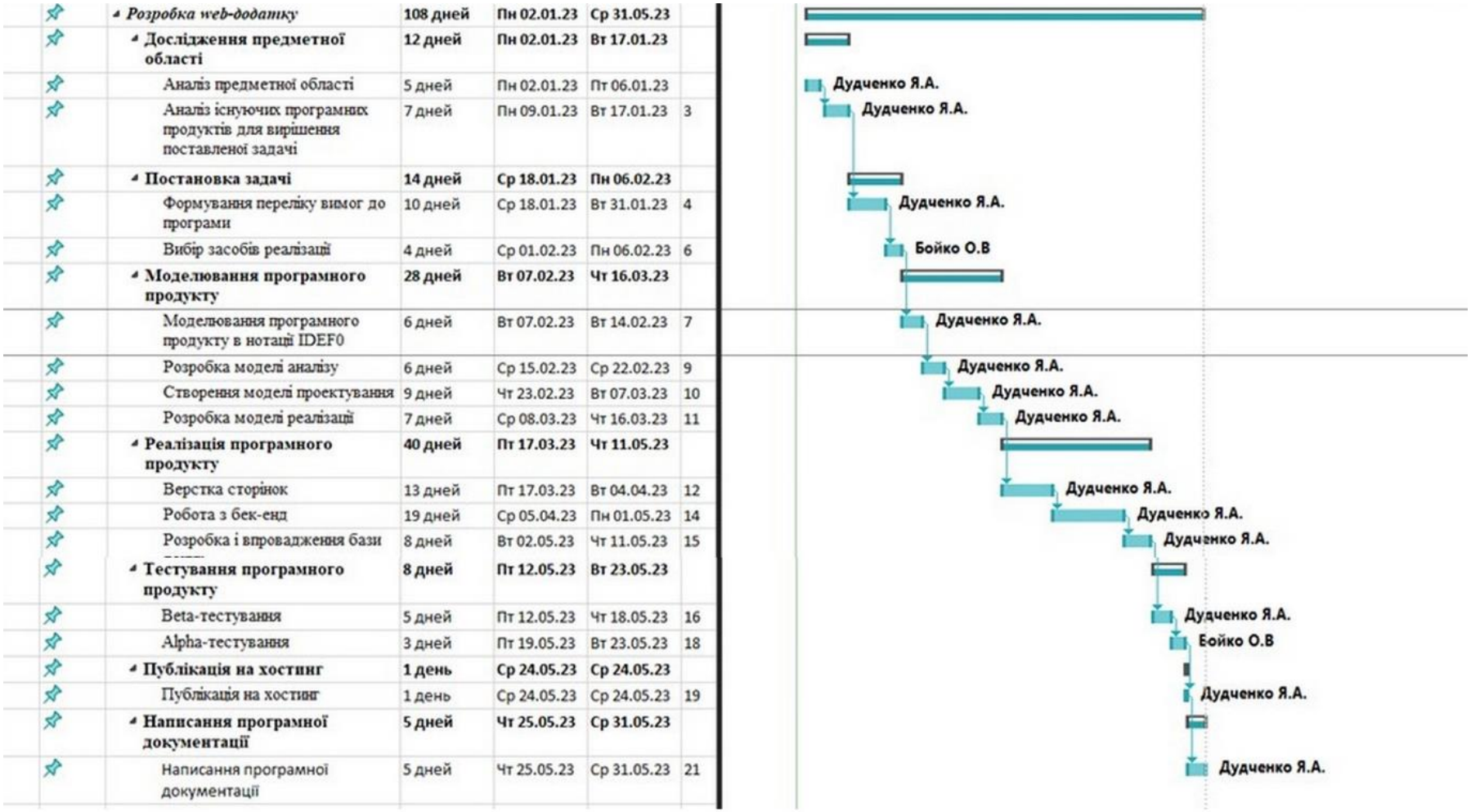
Рисунок Б.2 – OBS - структура робіт проєкту

5 Діаграма Ганта

Візуально являє собою простий набір смужок, що складаються з двох головних осей: справ і часу. Кожному тимчасовому проміжку приписується певна задача, яка повинна бути виконана. На діаграмі, крім основних блоків, може бути присутнім спеціальний додатковий стовпець, який показує відсоток виконання роботи. Особливі відмітки - віхи - застосовуються для виділення двох і декількох завдань і демонстрації послідовності їх виконання.

Графік Ганта є своєрідним стандартом в області управління проєктами, адже саме з його допомогою з'являється можливість показати структуру виконання всіх етапів проекту наочно.

Календарний графік проекту представлено на рисунку Б.3.



	Назва	Тривалість	Початок	Завершення	Предшественники
	Розробка web-додатку	108 дней	Пн 02.01.23	Ср 31.05.23	
	Дослідження предметної області	12 дней	Пн 02.01.23	Вт 17.01.23	
	Аналіз предметної області	5 дней	Пн 02.01.23	Пт 06.01.23	
	Аналіз існуючих програмних продуктів для вирішення поставленої задачі	7 дней	Пн 09.01.23	Вт 17.01.23	3
	Постановка задачі	14 дней	Ср 18.01.23	Пн 06.02.23	
	Формування переліку вимог до програми	10 дней	Ср 18.01.23	Вт 31.01.23	4
	Вибір засобів реалізації	4 дней	Ср 01.02.23	Пн 06.02.23	6
	Моделювання програмного продукту	28 дней	Вт 07.02.23	Чт 16.03.23	
	Моделювання програмного продукту в нотації IDEF0	6 дней	Вт 07.02.23	Вт 14.02.23	7
	Розробка моделі аналізу	6 дней	Ср 15.02.23	Ср 22.02.23	9
	Створення моделі проектування	9 дней	Чт 23.02.23	Вт 07.03.23	10
	Розробка моделі реалізації	7 дней	Ср 08.03.23	Чт 16.03.23	11
	Реалізація програмного продукту	40 дней	Пт 17.03.23	Чт 11.05.23	
	Верстка сторінок	13 дней	Пт 17.03.23	Вт 04.04.23	12
	Робота з бек-енд	19 дней	Ср 05.04.23	Пн 01.05.23	14
	Розробка і впровадження бази	8 дней	Вт 02.05.23	Чт 11.05.23	15
	Тестування програмного продукту	8 дней	Пт 12.05.23	Вт 23.05.23	
	Beta-тестування	5 дней	Пт 12.05.23	Чт 18.05.23	16
	Alpha-тестування	3 дней	Пт 19.05.23	Вт 23.05.23	18
	Публікація на хостинг	1 день	Ср 24.05.23	Ср 24.05.23	
	Публікація на хостинг	1 день	Ср 24.05.23	Ср 24.05.23	19
	Написання програмної документації	5 дней	Чт 25.05.23	Ср 31.05.23	
	Написання програмної документації	5 дней	Чт 25.05.23	Ср 31.05.23	21

Рисунок Б.3 – Календарний графік проєкту

6 Управління ризиками проекту

Під час виконання якісної оцінки ризиків треба визначити ризики, які мають бути усунені якнайшвидше. В залежності від ступеня важливості ризику – реагування буде відповідне. Наступним етапом є виконання кількісного оцінювання ризиків. Кількісне та якісне оцінювання можуть виконувати одночасно або окремо, що залежить від ступеня забезпечення проекту. У таблиці Б.3 представлено шкалу для класифікації ризиків за величиною впливу на проект та ймовірністю виникнення.

Таблиця Б.3 – Шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	Ймовірність виникнення	Вплив ризику	Тип ризику
1	Низька	Низький	Прийнятні
2	Середня	Середній	Виправдані
3	Висока	Високий	Недопустимі

Для того, щоб знизити негативний вплив ризиків на проект треба виконати планування реагування на них. До нього входить визначення ефективності розробки та оцінка наслідків впливу на проект. Оцінювання виконується за показниками, що описані в таблиці Б.3. У результаті планування реагування було отримано матрицю ймовірності виникнення ризиків та впливу ризику, що зображена на рисунку Б.7. Зеленим кольором на матриці позначають прийнятні ризики, жовтим – виправдані, а червоним – недопустимі.

Ймовірність ризику (Й)	Вплив загрози (ризику)				
	Дуже малий	Малий	Середній	Великий	Дуже великий
	0,05	0,1	0,2	0,4	0,8
0,9					
0,7			R6(0,16), R10(0,16)		
0,5				R4(0,20)	R8(0.45)
0,3	R7(0.03), R1(0.03)		R2(0,06)	R5(0,12), R9(0,12)	
0,1					R3(0,08)

Рисунок Б.7 – Матриця ймовірності

Класифікація ризиків за рівнем, відповідно до отриманого значення індексу, представлена у таблиці Б.4. У таблиці Б.5 описано ризики та стратегії реагування на кожен з них.

Таблиця Б.4 – Шкала оцінювання за рівнем ризику

№	Назва	Межі	Ризики, які входять (номера)
1	Прийнятні	$0,005 \leq R \leq 0,05$	7, 1
2	Виправдані	$0,05 < R \leq 0,14$	2, 4, 5, 10
3	Недопустимі	$0,14 < R \leq 0,72$	3, 6, 8, 9

Таблиця Б.5 Ризики та стратегії реагування

ID ризику	**Статус ризику**	**Опис ризику**	**Ймовірність виникнення**	**Вплив ризику**	**Ранг ризику**	**План А (заходи запобігання виникненню ризику)**	**Тип стратегії реагування**	**План Б (заходи усунення наслідків ризику)**
RS_1	Відкритий	Непорозуміння між розробником та замовником	Низька	Середній	0,03	Створити якомога комфортніших умов для співпраці і комунікації між розробником і замовником.	Попередження	Виявити причину непорозуміння і швидко ліквідувати джерело проблеми.
RS_2	Відкритий	Поява альтернативного продукту	Низька	Середній	0,06	1.Провести попереднє дослідження альтернативних продуктів.	Прийняття	Вибрати унікальну стратегію створення проекту
RS_3	Відкритий	Нечітке завдання на розробку	Низька	Високий	0,08	1. На самому початковому етапі обговорити всі моменти з замовником щоб усунути можливі непорозуміння.	Попередження	При виявленні невідповідностей деяких характеристик продукту заявленим вимогам потрібно уважно та чітко окреслити те, що було виконано невірно та зробити правки.

Продовження таблиці Б.5

RS_4	Відкритий	Низька кваліфікація розробників	Середня	Високий	0,2	1.Орієнтуватися на пошук робітників із належною кваліфікацією до подібних проектів.	Прийняття	Обирати перевірених розробників, котрі мають досвід
RS_5	Відкритий	Неоптимальний розподіл часу	Середня	Високий	0,12	1.Розрахувати час виконання замовлення із запасом.	Попередження	Розподілити завдання на кожного робітника із чітким, визначеним терміном часу.
RS_6	Відкритий	Захворювання учасника команди розробників	Середній	Середній	0,16	Профілактика найпоширеніших сезонних захворювань	Прийняття	
RS_7	Відкритий	Вимкнення електроенергії	Низький	Середній	0,03	Мати в своєму розпорядженні генератори та інші джерела резервного живлення	Попередження	Контролювати графіки відключень електроенергії

Продовження таблиці Б.5

RS_8	Відкритий	Загрози обстрілів/авіаударів	Високий	Високий	0.45	Перемістити офіс подалі від бойових дій. Допомогти учасникам проекту покинути обстрілювану територію.	Ухилення	Перенести строки виконання робіт. Перепланування.
RS_9	Відкритий	Невірно обраний технологічний стек	Малий	Середній	0,12	Ретельно досліджувати технології розробки та перевіряти їх актуальність	Попередження	Систематична перевірка технології розробки
RS_10	Відкритий	Недостатня конфігурація обладнання	Середній	Середній	0,16	Завчасне оновлення обладнання та його модернізація	Попередження	Систематичний контроль обладнання

ДОДАТОК В

Акт впровадження додатку

ЗАТВЕРДЖУЮ
Директор
Сумської філії Харківського
національного університету
внутрішніх справ
_____ Сергій ЛУКАШ
«26» 05 2023 року

А К Т
про впровадження web-додатку «Dialogify»
здобувача вищої освіти у Сумському
державному університеті
Дудченка Ярослава Андрійовича

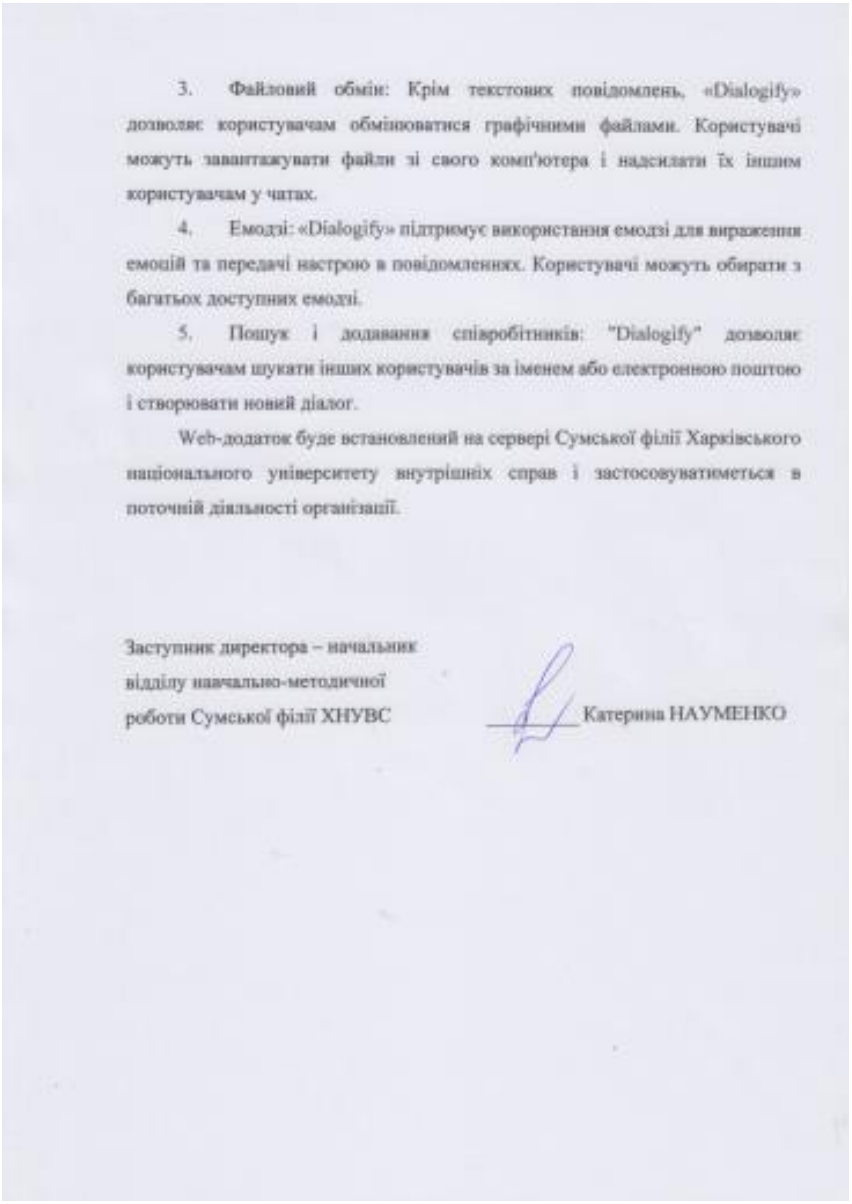
Цим актом підтверджується майбутнє впровадження і використання у Сумській філії Харківського національного університету внутрішніх справ (м. Суми, вул. Миру, буд. 24, 40007; https://sumy.univd.edu.ua/uk) web-додатку обміну повідомленнями «Dialogify».

Web-додаток обміну повідомленнями "Dialogify" є інтерактивним інструментом, розробленим для забезпечення зручної комунікації та обміну інформацією між користувачами. Характеристики та можливості web-додатку обміну повідомленнями «Dialogify»:

1. Реєстрація та профілі користувачів: Користувачі можуть створювати облікові записи в «Dialogify» і налаштовувати свої профілі, включаючи особисту інформацію та фотографії.

2. Обмін повідомленнями: «Dialogify» дозволяє користувачам обмінюватися текстовими повідомленнями в режимі реального часу. Користувачі можуть ініціювати чати з іншими користувачами або створювати групові чати для обговорення спільних проектів або тем.

Рисунок В.1 - Акт впровадження



3. Файловий обмін: Крім текстових повідомлень, «Dialogify» дозволяє користувачам обмінюватися графічними файлами. Користувачі можуть завантажувати файли зі свого комп'ютера і надсилати їх іншим користувачам у чатах.

4. Емодзі: «Dialogify» підтримує використання емодзі для вираження емоцій та передачі настрою в повідомленнях. Користувачі можуть обирати з багатьох доступних емодзі.

5. Пошук і додавання співробітників: "Dialogify" дозволяє користувачам шукати інших користувачів за іменем або електронною поштою і створювати новий діалог.

Web-додаток буде встановлений на сервері Сумської філії Харківського національного університету внутрішніх справ і застосовуватиметься в поточній діяльності організації.

Заступник директора – начальник
відділу навчально-методичної
роботи Сумської філії ХНУВС Катерина НАУМЕНКО

Рисунок В.2 - Акт впровадження (продовження)

ДОДАТОК Г
ЛІСТИНГ ПРОГРАМНОГО КОДУ

app.js

```
require('dotenv').config()

module.exports = {
    appKey: process.env.APP_KEY,
    appUrl: process.env.APP_URL,
    appPort: process.env.APP_PORT
}
```

database.js

```
require('dotenv').config()

module.exports = {
    "development": {
        "username": process.env.DB_USER,
        "password": process.env.DB_PASSWORD,
        "database": process.env.DB_DATABASE,
        "host": process.env.DB_HOST,
        "dialect": "postgres",
        "logging": false
    },
    "test": {
        "username": process.env.DB_USER,
        "password": process.env.DB_PASSWORD,
        "database": process.env.DB_DATABASE,
        "host": process.env.DB_HOST,
        "dialect": "postgres",
        "logging": false
    },
    "production": {
        "username": process.env.DB_USER,
        "password": process.env.DB_PASSWORD,
        "database": process.env.DB_DATABASE,
        "host": process.env.DB_HOST,
        "dialect": "postgres",
        "logging": false
    }
```

```
}
```

authController.js

```
const User = require('../models').User
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const config = require('../config/app')

exports.login = async (req, res) => {
    const { email, password } = req.body

    try {

        // const secret = require('crypto').randomBytes(64).toString('hex')

        // find the user
        const user = await User.findOne({
            where: {
                email
            }
        })

        // check if user found
        if (!user) return res.status(404).json({ message: 'User not found!'});

        // check if password matches
                if   (!bcrypt.compareSync(password,   user.password))   return
res.status(401).json({ message: 'Incorrect password!' })

        // generate auth token
        const userWithToken = generateToken(user.get({ raw: true }))
        userWithToken.user.avatar = user.avatar

        return res.send(userWithToken)

    } catch (e) {
        return res.status(500).json({ message: e.message })
    }
}

exports.register = async (req, res) => {
```

```
    try {
        const user = await User.create(req.body)

        const userWithToken = generateToken(user.get({ raw: true }))
        return res.send(userWithToken)
    } catch (e) {
        return res.status(500).json({ message: e.message })
    }
}

const generateToken = (user) => {

    delete user.password

    const token = jwt.sign(user, config.appKey, { expiresIn: 86400 })

    return { ...{ user }, ...{ token } }
}
```

chatController.js

```
const models = require('../models')
const { Op } = require('sequelize')
const { sequelize } = require('../models')
const User = models.User
const Chat = models.Chat
const ChatUser = models.ChatUser
const Message = models.Message

exports.index = async (req, res) => {

    const user = await User.findOne({
        where: {
            id: req.user.id
        },
        include: [
            {
                model: Chat,
                include: [
                    {
                        model: User,
```

```
                        where: {
                            [Op.not]: {
                                id: req.user.id
                            }
                        }
                    },
                    {
                        model: Message,
                        include: [
                            {
                                model: User
                            }
                        ],
                        limit: 20,
                        order: [['id', 'DESC']]
                    }
                ]
            }
        ]
    })

    return res.json(user.Chats)
}

exports.create = async (req, res) => {

    const { partnerId } = req.body

    const t = await sequelize.transaction()

    try {

        const user = await User.findOne({
            where: {
                id: req.user.id
            },
            include: [
                {
                    model: Chat,
                    where: {
                        type: 'dual'
                    },
```

```
                include: [
                    {
                        model: ChatUser,
                        where: {
                            userId: partnerId
                        }
                    }
                ]
            }
        ]
    })

    if (user && user.Chats.length > 0)
        return res.status(403).json({ status: 'Error', message: 'Chat with
this user already exists!' })

    const chat = await Chat.create({ type: 'dual' }, { transaction: t })

    await ChatUser.bulkCreate([
        {
            chatId: chat.id,
            userId: req.user.id
        },
        {
            chatId: chat.id,
            userId: partnerId
        }
    ], { transaction: t })


    await t.commit()

    // const chatNew = await Chat.findOne({
    //     where: {
    //         id: chat.id
    //     },
    //     include: [
    //         {
    //             model: User,
    //             where: {
    //                 [Op.not]: {
    //                     id: req.user.id
```

```
        //                  }
        //              }
        //          },
        //          {
        //              model: Message
        //          }
        //      ]
        // })

        const creator = await User.findOne({
            where: {
                id: req.user.id
            }
        })

        const partner = await User.findOne({
            where: {
                id: partnerId
            }
        })

        const forCreator = {
            id: chat.id,
            type: 'dual',
            Users: [partner],
            Messages: []
        }

        const forReceiver = {
            id: chat.id,
            type: 'dual',
            Users: [creator],
            Messages: []
        }


        return res.json([forCreator, forReceiver])

    } catch (e) {
        await t.rollback()
        return res.status(500).json({ status: 'Error', message: e.message })
    }
```

```
}

exports.messages = async (req, res) => {

    const limit = 10
    const page = req.query.page || 1
    const offset = page > 1 ? page * limit : 0

    const messages = await Message.findAndCountAll({
        where: {
            chatId: req.query.id
        },
        include: [
            {
                model: User
            }
        ],
        limit,
        offset,
        order: [['id', 'DESC']]
    })

    const totalPages = Math.ceil(messages.count / limit)

    if (page > totalPages) return res.json({ data: { messages: [] } })

    const result = {
        messages: messages.rows,
        pagination: {
            page,
            totalPages
        }
    }

    return res.json(result)
}

exports.imageUpload = (req, res) => {
    if (req.file) {
        return res.json({ url: req.file.filename })
    }
```

```
    return res.status(500).json('No image uploaded')
}

exports.addUserToGroup = async (req, res) => {
    try {

        const { chatId, userId } = req.body

        const chat = await Chat.findOne({
            where: {
                id: chatId
            },
            include: [
                {
                    model: User,
                },
                {
                    model: Message,
                    include: [
                        {
                            model: User
                        }
                    ],
                    limit: 20,
                    order: [['id', 'DESC']]
                }
            ]
        })

        chat.Messages.reverse()

        // check if already in the group
        chat.Users.forEach(user => {
            if (user.id === userId) {
                return res.status(403).json({ message: 'User already in the
group!' })
            }
        })

        await ChatUser.create({ chatId, userId })

        const newChatter = await User.findOne({
```

```
            where: {
                id: userId
            }
        })

        if (chat.type === 'dual') {
            chat.type = 'group'
            chat.save()
        }

        return res.json({ chat, newChatter })

    } catch (e) {
        return res.status(500).json({ status: 'Error', message: e.message })
    }
}

exports.deleteChat = async (req, res) => {

    const { id } = req.params

    try {
        const chat = await Chat.findOne({
            where: {
                id
            },
            include: [
                {
                    model: User
                }
            ]
        })

        const notifyUsers = chat.Users.map(user => user.id)

        await chat.destroy()
        return res.json({ chatId: id, notifyUsers })

    } catch (e) {
        return res.status(500).json({ status: 'Error', message: e.message })
    }
}
```

```
exports.leaveCurrentChat = async (req, res) => {

    try {
        const { chatId } = req.body
        const chat = await Chat.findOne({
            where: {
                id: chatId
            },
            include: [
                {
                    model: User
                }
            ]
        })

        if (chat.Users.length === 2) {
            return res.status(403).json({ status: 'Error', message: 'You cannot
leave this chat' })

        }

        if (chat.Users.length === 3) {
            chat.type = 'dual'
            chat.save()
        }

        await ChatUser.destroy({
            where: {
                chatId,
                userId: req.user.id
            }
        })

        await Message.destroy({
            where: {
                chatId,
                fromUserId: req.user.id
            }
        })

        const notifyUsers = chat.Users.map(user => user.id)
```

```
        return res.json({ chatId: chat.id, userId: req.user.id, currentUserId:
req.user.id, notifyUsers })

    } catch (e) {
        return res.status(500).json({ status: 'Error', message: e.message })
    }
}
```

userController.js

```
const User = require('../models').User
const sequelize = require('sequelize')

exports.update = async (req, res) => {

    if (req.file) {
        req.body.avatar = req.file.filename
    }

    if (typeof req.body.avatar !== 'undefined' && req.body.avatar.length === 0)
delete req.body.avatar

    try {

        const [rows, result] = await User.update(req.body,
            {
                where: {
                    id: req.user.id
                },
                returning: true,
                individualHooks: true
            }
        )

        const user = result[0].get({ raw: true })
        user.avatar = result[0].avatar
        delete user.password

        return res.send(user)

    } catch (e) {
```

```
        return res.status(500).json({ error: e.message })
    }
}

exports.search = async (req, res) => {

    try {

        const users = await User.findAll({
            where: {
                [sequelize.Op.or]: {
                    namesConcated: sequelize.where(
                        sequelize.fn('concat',  sequelize.col('firstName'),  '
', sequelize.col('lastName')),
                        {
                            [sequelize.Op.iLike]: `%${req.query.term}%`
                        }
                    ),
                    email: {
                        [sequelize.Op.iLike]: `%${req.query.term}%`
                    }
                },
                [sequelize.Op.not]: {
                    id: req.user.id
                }
            },
            limit: 10
        })

        return res.json(users)

    } catch (e) {
        return res.status(500).json({ error: e.message })
    }
}
```

create-user.js

```
'use strict';
module.exports = {
    up: async (queryInterface, Sequelize) => {
        await queryInterface.createTable('Users', {
```

```
            id: {
                allowNull: false,
                autoIncrement: true,
                primaryKey: true,
                type: Sequelize.INTEGER
            },
            firstName: {
                type: Sequelize.STRING
            },
            lastName: {
                type: Sequelize.STRING
            },
            email: {
                type: Sequelize.STRING,
                unique: true
            },
            password: {
                type: Sequelize.STRING
            },
            gender: {
                type: Sequelize.STRING
            },
            avatar: {
                type: Sequelize.STRING
            },
            createdAt: {
                allowNull: false,
                type: Sequelize.DATE,
                defaultValue: Sequelize.literal('NOW()')
            },
            updatedAt: {
                allowNull: false,
                type: Sequelize.DATE,
                defaultValue: Sequelize.literal('NOW()')
            }
        });
    },
    down: async (queryInterface, Sequelize) => {
        await queryInterface.dropTable('Users');
    }
};
```

create-chat.js

```
'use strict';
module.exports = {
    up: async (queryInterface, Sequelize) => {
        await queryInterface.createTable('Chats', {
            id: {
                allowNull: false,
                autoIncrement: true,
                primaryKey: true,
                type: Sequelize.INTEGER
            },
            type: {
                type: Sequelize.STRING,
                defaultValue: 'dual'
            },
            createdAt: {
                allowNull: false,
                type: Sequelize.DATE,
                defaultValue: Sequelize.literal('NOW()')
            },
            updatedAt: {
                allowNull: false,
                type: Sequelize.DATE,
                defaultValue: Sequelize.literal('NOW()')
            }
        });
    },
    down: async (queryInterface, Sequelize) => {
        await queryInterface.dropTable('Chats');
    }
};
```

create-chat-user.js

```
'use strict';
module.exports = {
    up: async (queryInterface, Sequelize) => {
        await queryInterface.createTable('ChatUsers', {
            id: {
                allowNull: false,
                autoIncrement: true,
                primaryKey: true,
```

```
                type: Sequelize.INTEGER
            },
            chatId: {
                type: Sequelize.INTEGER,
                allowNull: false,
                references: {
                    model: 'Chats',
                    key: 'id'
                },
                onDelete: 'CASCADE'
            },
            userId: {
                type: Sequelize.INTEGER,
                allowNull: false,
                references: {
                    model: 'Users',
                    key: 'id'
                },
                onDelete: 'CASCADE'
            },
            createdAt: {
                allowNull: false,
                type: Sequelize.DATE,
                defaultValue: Sequelize.literal('NOW()')
            },
            updatedAt: {
                allowNull: false,
                type: Sequelize.DATE,
                defaultValue: Sequelize.literal('NOW()')
            }
        });
    },
    down: async (queryInterface, Sequelize) => {
        await queryInterface.dropTable('ChatUsers');
    }
};
```

create-message.js

```
'use strict';
module.exports = {
    up: async (queryInterface, Sequelize) => {
```

```
await queryInterface.createTable('Messages', {
    id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
    },
    type: {
        type: Sequelize.STRING,
        defaultValue: 'text'
    },
    message: {
        type: Sequelize.TEXT
    },
    chatId: {
        type: Sequelize.INTEGER,
        allowNull: false,
        references: {
            model: 'Chats',
            key: 'id'
        },
        onDelete: 'CASCADE'
    },
    fromUserId: {
        type: Sequelize.INTEGER,
        allowNull: false,
        references: {
            model: 'Users',
            key: 'id'
        },
        onDelete: 'CASCADE'
    },
    createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: Sequelize.literal('NOW()')
    },
    updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: Sequelize.literal('NOW()')
    }
```

```
        });
    },
    down: async (queryInterface, Sequelize) => {
        await queryInterface.dropTable('Messages');
    }
};
```

login.js

```
const { body } = require('express-validator')

exports.rules = (() => {
    return [
        body('email').isEmail()
    ]
})()
```

login.js

```
const { body } = require('express-validator')

exports.rules = (() => {
    return [
        body('firstName').notEmpty(),
        body('lastName').notEmpty(),
        body('gender').notEmpty(),
        body('email').isEmail(),
        body('password').isLength({ min: 5 })
    ]
})()
```

update.js

```
const { body } = require('express-validator')

exports.rules = (() => {
    return [
        body('firstName').notEmpty(),
        body('lastName').notEmpty(),
        body('gender').notEmpty(),
        body('email').isEmail(),
        body('password').optional().isLength({ min: 5 })
    ]
})()
```

index.js

```
const { validationResult } = require('express-validator')

exports.validate = (req, res, next) => {
  const errors = validationResult(req)

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() })
  }

  next()
}
```

App.js

```
import React from 'react';
import Login from './components/Auth/Login'
import Register from './components/Auth/Register'
import Chat from './components/Chat/Chat'
import ProtectedRoute from './components/Router/ProtectedRoute'

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom'

import './App.scss';

import { library } from '@fortawesome/fontawesome-svg-core'
import { faSmile, faImage } from '@fortawesome/free-regular-svg-icons'
import { faSpinner, faEllipsisV, faUserPlus, faSignOutAlt, faTrash,
faCaretDown, faUpload, faTimes, faBell } from '@fortawesome/free-solid-svg-
icons'
library.add(faSmile, faImage, faSpinner, faEllipsisV, faUserPlus, faSignOutAlt,
faTrash, faCaretDown, faUpload, faTimes, faBell)

function App() {
    return (
        <Router>
            <div className="App">
                <Switch>
                    <ProtectedRoute exact path='/' component={Chat} />
                    <Route path='/login' component={Login} />
                    <Route path='/register' component={Register} />
                    <Route render={() => <h1>404 Сторінка не знайдена</h1>} />
                </Switch>
```

```
            </div>
        </Router>
    );
}

export default App;
```

Login.js

```
import React, { useState } from 'react'
import loginImage from '../../assets/images/login.svg'
import { Link } from 'react-router-dom'

import { useDispatch } from 'react-redux'
import { login } from '../../store/actions/auth'

import './Auth.scss'

const Login = ({ history }) => {

    const dispatch = useDispatch()

    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')

    const submitForm = (e) => {
        e.preventDefault()

        dispatch(login({ email, password }, history))
    }

    return (
        <div id='auth-container'>
            <div id='auth-card'>
                <div className='card-shadow'>
                    <div id='image-section'>
                        <img src={loginImage} alt='Login' />
                    </div>

                    <div id='form-section'>
                        <h2>З поверненням!</h2>
```

```
                        <form onSubmit={submitForm}>
                            <div className='input-field mb-1'>
                                <input
                                    onChange={e => setEmail(e.target.value)}
                                    value={email}
                                    required='required'
                                    type='text'
                                    placeholder='Email' />
                            </div>

                            <div className='input-field mb-2'>
                                <input
                                    onChange={e                                    =>
setPassword(e.target.value)}
                                    value={password}
                                    required='required'
                                    type='password'
                                    placeholder='Пароль' />
                            </div>

                            <button>ВХІД</button>
                        </form>

                        <p>Все ще не маєте акаунта? <Link to='/register'
class="link">Реєстрація</Link></p>
                    </div>
                </div>
            </div>
        </div>
    );
}

export default Login
```

Register.js

```
import React, { useState } from 'react'
import registerImage from '../../assets/images/register.svg'
import { Link } from 'react-router-dom'
import { useDispatch } from 'react-redux'
import { register } from '../../store/actions/auth'
import './Auth.scss'
```

```
const Register = ({ history }) => {

    const dispatch = useDispatch()

    const [firstName, setFirstName] = useState('')
    const [lastName, setLastName] = useState('')
    const [email, setEmail] = useState('')
    const [gender, setGender] = useState('male')
    const [password, setPassword] = useState('')

    const submitForm = (e) => {
        e.preventDefault()

        dispatch(register({ firstName, lastName, email, gender, password },
history))
    }

    return (
        <div id='auth-container'>
            <div id='auth-card'>
                <div className='card-shadow'>
                    <div id='image-section'>
                        <img src={registerImage} alt='Register' />
                    </div>

                    <div id='form-section'>
                        <h2>Створення акаунту</h2>

                        <form onSubmit={submitForm}>
                            <div className='input-field mb-1'>
                                <input
                                    onChange={e                                    =>
setFirstName(e.target.value)}
                                    value={firstName}
                                    required='required'
                                    type='text'
                                    placeholder="Ім'я" />
                            </div>

                            <div className='input-field mb-1'>
                                <input
```

```
                                    onChange={e                                 =>
setLastName(e.target.value)}
                                    value={lastName}
                                    required='required'
                                    type='text'
                                    placeholder='Прізвище' />
                            </div>

                            <div className='input-field mb-1'>
                                <input
                                    onChange={e => setEmail(e.target.value)}
                                    value={email}
                                    required='required'
                                    type='text'
                                    placeholder='Email' />
                            </div>

                            <div className='input-field mb-1'>
                                <select
                                    className='select-field'
                                    onChange={e => setGender(e.target.value)}
                                    value={gender}
                                    required='required'
                                >
                                    <option value='male'>Чоловік</option>
                                    <option value='female'>Жінка</option>
                                </select>
                                </div>

                            <div className='input-field mb-2'>
                                <input
                                    onChange={e                                 =>
setPassword(e.target.value)}
                                    value={password}
                                    required='required'
                                    type='password'
                                    placeholder='Пароль' />
                            </div>

                                <button>РЕЄСТРАЦІЯ</button>
                        </form>
```

```
                        <p>Вже маєте акаунт? <Link to='/login'
class="link">Вхід</Link></p>
                    </div>
                </div>
            </div>
        </div>
    );
}

export default Register
```

ChatHeader.js

```
import React, { Fragment, useState } from 'react'
import { userStatus } from '../../../../utils/helpers'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { useSelector } from 'react-redux'
import Modal from '../../../Modal/Modal'
import ChatService from '../../../../services/chatService'
import './ChatHeader.scss'

const ChatHeader = ({ chat }) => {

    const [showChatOptions, setShowChatOptions] = useState(false)
    const [showAddFriendModal, setShowAddFriendModal] = useState(false)
    const [suggestions, setSuggestions] = useState([])

    const socket = useSelector(state => state.chatReducer.socket)


    const searchFriends = (e) => {
        ChatService.searchUsers(e.target.value)
            .then(res => setSuggestions(res))
    }

    const addNewFriend = (id) => {
        ChatService.addFriendToGroupChat(id, chat.id)
            .then(data => {
                socket.emit('add-user-to-group', data)
                setShowAddFriendModal(false)
            }).catch(err => console.log(err))
    }
```

```
    const leaveChat = () => {
        ChatService.leaveCurrentChat(chat.id)
            .then(data => {
                socket.emit('leave-current-chat', data)
            })
            .catch(err => console.log(err))
    }

    const deleteChat = () => {
        ChatService.deleteCurrentChat(chat.id)
            .then(data => {
                socket.emit('delete-chat', data)
            })
    }

    return (
        <Fragment>
            <div id='chatter'>
                {
                    chat.Users.map(user => {
                        return <div className='chatter-info' key={user.id}>
                            <h3>{user.firstName} {user.lastName}</h3>
                            <div className='chatter-status'>
                                <span                 className={`online-status
${userStatus(user)}`}></span>
                            </div>
                        </div>
                    })
                }
            </div>
            <FontAwesomeIcon
                onClick={() => setShowChatOptions(!showChatOptions)}
                icon={['fas', 'ellipsis-v']}
                className='fa-icon'
            />
            {
                showChatOptions
                    ? <div id='settings'>
                        <div onClick={() => setShowAddFriendModal(true)}>
                            <FontAwesomeIcon
                                icon={['fas', 'user-plus']}
```

```
                        className='fa-icon'
                    />
                    <p>Додати користувача до чату</p>
                </div>

                {
                    chat.type === 'group'
                        ? <div onClick={() => leaveChat()}>
                            <FontAwesomeIcon
                                icon={['fas', 'sign-out-alt']}
                                className='fa-icon'
                            />
                            <p>Покинути чат</p>
                        </div>
                        : null
                }

                {
                    chat.type === 'dual' ?
                        <div onClick={() => deleteChat()}>
                            <FontAwesomeIcon
                                icon={['fas', 'trash']}
                                className='fa-icon'
                            />
                            <p>Видалити чат</p>
                        </div>
                        : null
                }
            </div>
            : null
    }
    {
        showAddFriendModal &&
        <Modal click={() => setShowAddFriendModal(false)}>
            <Fragment key='header'>
                <h3 className='m-0'>Додати друга до групового чату</h3>
            </Fragment>

            <Fragment key='body'>
                <div id='add-friends'>
                <p>Знайдіть друзів, ввівши їхні імена нижче</p>
                <input
```

```
                            onInput={e => searchFriends(e)}
                            type='text'
                            placeholder='Пошук...'
                        />
                            <div id='suggestions'>
                                {
                                    suggestions.map(user => {
                                        return       <div       key={user.id}
className='suggestion'>
                                            <p                  className='m-
0'>{user.firstName} {user.lastName}</p>
                                            <button       onClick={()       =>
addNewFriend(user.id)}>ДОДАТИ</button>
                                        </div>
                                    })
                                }
                            </div>
                        </div>
                    </Fragment>
                </Modal>
            }
        </Fragment>
    )
}

export default ChatHeader
```

Friend.js

```
import React from 'react'
import { useSelector } from 'react-redux'
import { userStatus } from '../../../../utils/helpers'
import './Friend.scss'

const Friend = ({ chat, click }) => {

    const currentChat = useSelector(state => state.chatReducer.currentChat)

    const isChatOpened = () => {
        return currentChat.id === chat.id ? 'opened' : ''
    }
```

```
    const lastMessage = () => {
        if (chat.Messages.length === 0) return ''

        const message = chat.Messages[chat.Messages.length - 1]
        return message.type === 'image' ? 'image uploaded' : message.message
    }

    return (
        <div onClick={click} className={`friend-list ${isChatOpened()}`}>
            <div>
                <img    width='40'    height='40'    src={chat.Users[0].avatar}
alt='User avatar' />
                <div className='friend-info'>
                    <h4              className='m-0'>{chat.Users[0].firstName}
{chat.Users[0].lastName}</h4>
                    <h5 className='m-0'>{lastMessage()}</h5>
                </div>
            </div>
            <div className='friend-status'>
                <span                                  className={`online-status
${userStatus(chat.Users[0])}`}></span>
            </div>
        </div>
    )
}

export default Friend
```

FriendList.js

```
import React, { useState, Fragment } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import Friend from '../Friend/Friend'
import { setCurrentChat } from '../../../../store/actions/chat'
import Modal from '../../../Modal/Modal'
import ChatService from '../../../../services/chatService'
import './FriendList.scss'

const FriendList = () => {

    const dispatch = useDispatch()
    const chats = useSelector(state => state.chatReducer.chats)
```

```
    const socket = useSelector(state => state.chatReducer.socket)

    const [showFriendsModal, setShowFriendsModal] = useState(false)
    const [suggestions, setSuggestions] = useState([])

    const openChat = (chat) => {
        dispatch(setCurrentChat(chat))
    }

    const searchFriends = (e) => {
        ChatService.searchUsers(e.target.value)
            .then(res => setSuggestions(res))
    }

    const addNewFriend = (id) => {
        ChatService.createChat(id)
            .then(chats => {
                socket.emit('add-friend', chats)
                setShowFriendsModal(false)
            }).catch(err => console.log(err))
    }

    return (
        <div id='friends' className='shadow-light'>
            <div id='title'>
                <h3 className='m-0'>Друзі</h3>
                <button                    onClick={()                    =>
setShowFriendsModal(true)}>ДОДАТИ</button>
            </div>

            <hr />

            <div id='friends-box'>
                {
                    chats.length > 0
                        ? chats.map(chat => {
                            return   <Friend   click={()   =>   openChat(chat)}
chat={chat} key={chat.id} />
                        })
                        : <p id='no-chat'>Немає доданих друзів</p>
                }
            </div>
```

```
            {
                showFriendsModal &&
                <Modal click={() => setShowFriendsModal(false)}>
                    <Fragment key='header'>
                        <h3 className='m-0'>Створити новий чат</h3>
                    </Fragment>

                    <Fragment key='body'>
                        <p>Знайдіть друзів, ввівши їхні імена нижче</p>
                        <input
                            onInput={e => searchFriends(e)}
                            type='text'
                            placeholder='Пошук...'
                        />
                        <div id='suggestions'>
                            {
                                suggestions.map(user => {
                                    return          <div          key={user.id}
className='suggestion'>
                                        <p    className='m-0'>{user.firstName}
{user.lastName}</p>
                                        <button         onClick={()         =>
addNewFriend(user.id)}>ДОДАТИ</button>
                                    </div>
                                })
                            }
                        </div>
                    </Fragment>
                </Modal>
            }
        </div>
    )
}

export default FriendList
```

Message.js

```
import React from 'react'
import './Message.scss'

const Message = ({ user, chat, index, message }) => {
```

```
    const determineMargin = () => {

        if (index + 1 === chat.Messages.length) return

        return message.fromUserId === chat.Messages[index + 1].fromUserId ?
'mb-5' : 'mb-10'
    }

    return (
        <div className={`message ${determineMargin()} ${message.fromUserId ===
user.id ? 'creator' : ''}`}>
            <div className={message.fromUserId === user.id ? 'owner' : 'other-
person'}>
                {
                    message.fromUserId !== user.id
                        ?     <h6     className='m-0'>{message.User.firstName}
{message.User.lastName}</h6>
                        : null
                }                {
                    message.type === 'text'
                        ? <p className='m-0'>{message.message}</p>
                        : <img src={message.message} alt='User upload' />
                }
            </div>
        </div>
    )
}

export default Message
```

MessageBox.js

```
import React, { useEffect, useRef, useState } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import Message from '../Message/Message'
import { paginateMessages } from '../../../../store/actions/chat'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import './MessageBox.scss'

const MessageBox = ({ chat }) => {
```

```
const dispatch = useDispatch()

const user = useSelector(state => state.authReducer.user)
const scrollBottom = useSelector(state => state.chatReducer.scrollBottom)
const senderTyping = useSelector(state => state.chatReducer.senderTyping)
const [loading, setLoading] = useState(false)
const [scrollUp, setScrollUp] = useState(0)

const msgBox = useRef()

const scrollManual = (value) => {
    msgBox.current.scrollTop = value
}

const handleInfiniteScroll = (e) => {
    if (e.target.scrollTop === 0) {

        setLoading(true)
        const pagination = chat.Pagination
        const page = typeof pagination === 'undefined' ? 1 : pagination.page

        dispatch(paginateMessages(chat.id, parseInt(page) + 1))
            .then(res => {
                if (res) {
                    setScrollUp(scrollUp + 1)
                }
                setLoading(false)
            }).catch(err => {
                setLoading(false)
            })
    }
}


useEffect(() => {
    setTimeout(() => {
        scrollManual(Math.ceil(msgBox.current.scrollHeight * 0.10))
    }, 100)
}, [scrollUp])

useEffect(() => {
```

```
        if (senderTyping.typing && msgBox.current.scrollTop > msgBox.current.scrollHeight * 0.30) {
            setTimeout(() => {
                scrollManual(msgBox.current.scrollHeight)
            }, 100)
        }
    }, [senderTyping])

    useEffect(() => {
        if (!senderTyping.typing) {
            setTimeout(() => {
                scrollManual(msgBox.current.scrollHeight)
            }, 100)
        }
    }, [scrollBottom])

    return (
        <div onScroll={handleInfiniteScroll} id='msg-box' ref={msgBox}>
            {
                loading
                    ? <p className='loader m-0'><FontAwesomeIcon icon='spinner' className='fa-spin' /></p>
                    : null
            }
            {
                chat.Messages.map((message, index) => {
                    return <Message
                        user={user}
                        chat={chat}
                        message={message}
                        index={index}
                        key={message.id}
                    />
                })
            }
            {
                senderTyping.typing && senderTyping.chatId === chat.id
                    ? <div className='message mt-5p'>
                        <div className='other-person'>
                            <p className='m-0'>{senderTyping.fromUser.firstName} {senderTyping.fromUser.lastName}...</p>
                        </div>
```

```
                    </div>
                    : null
            }
        </div>
    )
}

export default MessageBox
```

MessageInput.js

```
import React, { useState, useRef, useEffect } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import ChatService from '../../../../services/chatService'
import { Picker } from 'emoji-mart'
import { incrementScroll } from '../../../../store/actions/chat'
import 'emoji-mart/css/emoji-mart.css'
import './MessageInput.scss'

const MessageInput = ({ chat }) => {

    const dispatch = useDispatch()
    const user = useSelector(state => state.authReducer.user)
    const socket = useSelector(state => state.chatReducer.socket)
    const newMessage = useSelector(state => state.chatReducer.newMessage)

    const fileUpload = useRef()
    const msgInput = useRef()

    const [message, setMessage] = useState('')
    const [image, setImage] = useState('')
    const [showEmojiPicker, setShowEmojiPicker] = useState(false)
    const [showNewMessageNotification, setShowNewMessageNotification] =
useState(false)

    const handleMessage = (e) => {
        const value = e.target.value
        setMessage(value)

        const receiver = {
            chatId: chat.id,
```

```
            fromUser: user,
            toUserId: chat.Users.map(user => user.id)
        }

        if (value.length === 1) {
            receiver.typing = true
            socket.emit('typing', receiver)
        }

        if (value.length === 0) {
            receiver.typing = false
            socket.emit('typing', receiver)
        }

        // notify other users that this user is typing something
    }

    const handleKeyDown = (e, imageUpload) => {
        if (e.key === 'Enter') sendMessage(imageUpload)
    }

    const sendMessage = (imageUpload) => {

        if (message.length < 1 && !imageUpload) return

        const msg = {
            type: imageUpload ? 'image' : 'text',
            fromUser: user,
            toUserId: chat.Users.map(user => user.id),
            chatId: chat.id,
            message: imageUpload ? imageUpload : message
        }

        setMessage('')
        setImage('')
        setShowEmojiPicker(false)

        // send message with socket
        socket.emit('message', msg)
    }

    const handleImageUpload = () => {
```

```
        const formData = new FormData()
        formData.append('id', chat.id)
        formData.append('image', image)

        ChatService.uploadImage(formData)
            .then(image => {
                sendMessage(image)
            })
            .catch(err => console.log(err))
    }

    const selectEmoji = (emoji) => {
        const startPosition = msgInput.current.selectionStart
        const endPosition = msgInput.current.selectionEnd
        const emojiLength = emoji.native.length
        const value = msgInput.current.value
        setMessage(value.substring(0,  startPosition)  +  emoji.native  +
value.substring(endPosition, value.length))
        msgInput.current.focus()
        msgInput.current.selectionEnd = endPosition + emojiLength
    }

    useEffect(() => {
        const msgBox = document.getElementById('msg-box')
        if  (!newMessage.seen  &&  newMessage.chatId  ===  chat.id  &&
msgBox.scrollHeight !== msgBox.clientHeight) {
            if (msgBox.scrollTop > msgBox.scrollHeight * 0.30) {
                dispatch(incrementScroll())
            } else {
                setShowNewMessageNotification(true)
            }
        } else {
            setShowNewMessageNotification(false)
        }
    }, [newMessage, dispatch])

    const showNewMessage = () => {
        dispatch(incrementScroll())
        setShowNewMessageNotification(false)
    }

    return (
```

```
        <div id='input-container'>
            <div id='image-upload-container'>
                <div>
                    {
                        showNewMessageNotification
                            ?          <div          id='message-notification'
onClick={showNewMessage}>
                                <FontAwesomeIcon  icon='bell'  className='fa-
icon' />
                                <p className='m-0'>Нове повідомлення</p>
                            </div>
                            : null
                    }
                </div>

                <div id='image-upload'>
                    {
                        image.name ?
                            <div id='image-details'>
                                <p className='m-0'>{image.name}</p>
                                <FontAwesomeIcon
                                    onClick={handleImageUpload}
                                    icon='upload'
                                    className='fa-icon'
                                />
                                <FontAwesomeIcon
                                    onClick={() => setImage('')}
                                    icon='times'
                                    className='fa-icon'
                                />
                            </div>
                            : null
                    }
                    <FontAwesomeIcon
                        onClick={() => fileUpload.current.click()}
                        icon={['far', 'image']}
                        className='fa-icon'
                    />
                </div>
            </div>
            <div id='message-input'>
                <input
```

```
                    ref={msgInput}
                    value={message}
                    type='text'
                    placeholder='Повідомлення...'
                    onChange={e => handleMessage(e)}
                    onKeyDown={e => handleKeyDown(e, false)}
                />
                <FontAwesomeIcon
                    onClick={() => setShowEmojiPicker(!showEmojiPicker)}
                    icon={['far', 'smile']}
                    className='fa-icon'
                />
            </div>

            <input id='chat-image' ref={fileUpload} type='file' onChange={e =>
setImage(e.target.files[0])} />

            {
                showEmojiPicker
                    ? <Picker
                        title='Оберіть emoji...'
                        emoji='point_up'
                        style={{ position: 'absolute', bottom: '20px', right:
'20px' }}
                        onSelect={selectEmoji}
                    />
                    : null
            }

        </div>
    )
}

export default MessageInput
```

Messenger.js

```
import React from 'react'
import { useSelector } from 'react-redux'
import ChatHeader from '../ChatHeader/ChatHeader'
import MessageBox from '../MessageBox/MessageBox'
import MessageInput from '../MessageInput/MessageInput'
```

```
import './Messenger.scss'

const Messenger = () => {

    const chat = useSelector(state => state.chatReducer.currentChat)

    const activeChat = () => {
        return Object.keys(chat).length > 0
    }

    return (
        <div id='messenger' className='shadow-light'>
            {
                activeChat()
                    ? <div id='messenger-wrap'>
                        <ChatHeader chat={chat} />
                        <hr />
                        <MessageBox chat={chat} />
                        <MessageInput chat={chat} />
                    </div>
                    : <p>Немає активних чатів</p>
            }
        </div>
    )
}

export default Messenger
```

Navbar.js

```
import React, { useState, Fragment } from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { logout } from '../../../../store/actions/auth'
import Modal from '../../../Modal/Modal'
import { updateProfile } from '../../../../store/actions/auth'
import './Navbar.scss'

const Navbar = () => {

    const dispatch = useDispatch()
    const user = useSelector(state => state.authReducer.user)
```

```
    const [showProfileOptions, setShowProfileOptions] = useState(false)
    const [showProfileModal, setShowProfileModal] = useState(false)

    const [firstName, setFirstName] = useState(user.firstName)
    const [lastName, setLastName] = useState(user.lastName)
    const [email, setEmail] = useState(user.email)
    const [gender, setGender] = useState(user.gender)
    const [password, setPassword] = useState('')
    const [avatar, setAvatar] = useState('')

    const submitForm = (e) => {
        e.preventDefault()

        const form = { firstName, lastName, email, gender, avatar }
        if (password.length > 0) form.password = password

        const formData = new FormData()

        for (const key in form) {
            formData.append(key, form[key])
        }

        dispatch(updateProfile(formData)).then(()                                    =>
setShowProfileModal(false))
    }

    return (
        <div id='navbar' className='card-shadow'>
            <h2>Dialogify</h2>
            <div  onClick={()  =>  setShowProfileOptions(!showProfileOptions)}
id='profile-menu'>
                <img width="40" height="40" src={user.avatar} alt='Avatar' />
                <p>{user.firstName} {user.lastName}</p>
                <FontAwesomeIcon icon='caret-down' className='fa-icon' />

                {
                    showProfileOptions &&
                    <div id='profile-options'>
                        <p  onClick={()  =>  setShowProfileModal(true)}>Оновити
профіль</p>
                        <p onClick={() => dispatch(logout())}>Вийти</p>
```

```
                    </div>
                }

                {
                    showProfileModal &&
                    <Modal click={() => setShowProfileModal(false)}>

                        <Fragment key='header'>
                            <h3 className='m-0'>Оновити профіль</h3>
                        </Fragment>

                        <Fragment key='body'>
                            <form>
                                <div className='input-field mb-1'>
                                    <input
                                        onChange={e                          =>
setFirstName(e.target.value)}
                                        value={firstName}
                                        required='required'
                                        type='text'
                                        placeholder="Ім'я" />
                                </div>

                                <div className='input-field mb-1'>
                                    <input
                                        onChange={e                          =>
setLastName(e.target.value)}
                                        value={lastName}
                                        required='required'
                                        type='text'
                                        placeholder='Прізвище' />
                                </div>

                                <div className='input-field mb-1'>
                                    <input
                                        onChange={e                          =>
setEmail(e.target.value)}
                                        value={email}
                                        required='required'
                                        type='text'
                                        placeholder='Email' />
                                </div>
```

```
                                <div className='input-field mb-1'>
                                    <select
                                        className='select-field'
                                        onChange={e                          =>
setGender(e.target.value)}
                                        value={gender}
                                        required='required'
                                    >
                                        <option value='male'>Чоловік</option>
                                        <option value='female'>Жінка</option>
                                    </select>
                                </div>

                                <div className='input-field mb-2'>
                                    <input
                                        onChange={e                          =>
setPassword(e.target.value)}
                                        value={password}
                                        required='required'
                                        type='password'
                                        placeholder='Пароль' />
                                </div>
                                <p>Оберіть зображення аватару:</p>
                                <div className='input-field mb-2'>
                                    <input
                                        onChange={e                          =>
setAvatar(e.target.files[0])}
                                        type='file' />
                                </div>
                            </form>
                        </Fragment>

                        <Fragment key='footer'>
                            <button                    className='btn-success'
onClick={submitForm}>ОНОВИТИ</button>
                        </Fragment>

                    </Modal>
                }

            </div>
```

```
        </div>
    );
}

export default Navbar
```