МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

ГОНЧАРУК АНТОН ВІТАЛІЙОВИЧ

РОЗРОБКА ОНЛАЙН-ЧАТУ ЗАСОБАМИ NODE.JS

Спеціальність: 122 Комп'ютерні науки
Освітньо–професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
ЧЕРНЯЩУК НАТАЛІЯ ЛЕОНІДІВНА,
доктор педагогічних наук, професор кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп'ютерних наук та кібербезпеки від
_____ 2025 р.
Завідувач кафедри

(_____) Гришанович Т. О.

ЛУЦЬК 2025

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій спілкування між людьми набуло нових форм, які забезпечують швидкість, зручність і доступність обміну інформацією незалежно від фізичного розташування співрозмовника.

Онлайн–комунікація стала звичним елементом як повсякденного життя, так і професійної діяльності. Зі стрімким розвитком інтернету виникла потреба у нових інструментах взаємодії, що дозволяють користувачам миттєво передавати повідомлення, файли, зображення та інші типи даних.

Одним із найпопулярніших засобів такої взаємодії є онлайн-чати – інтерактивні вебсервіси, що забезпечують обмін повідомленнями в реальному часі. Вони активно застосовуються в електронній комерції, службах підтримки, освітніх платформах, соціальних мережах та корпоративних середовищах. Браузерні чати, зокрема, мають широке використання завдяки своїй доступності: вони не потребують встановлення додаткового програмного забезпечення, що робить їх зручними для користувачів із різних пристроїв.

Упровадження подібних рішень у вебпростір сприяє підвищенню якості обслуговування, покращує зворотній зв’язок і забезпечує персоналізований підхід до комунікації. Окрім того, онлайн-чати часто виступають невід’ємною частиною загального користувацького досвіду, відіграючи важливу роль у формуванні позитивного враження про вебресурс або компанію.

Розробка онлайн-чату включає в себе цілий спектр завдань – від створення привабливого та інтуїтивно зрозумілого інтерфейсу до налаштування серверної логіки обміну повідомленнями та забезпечення захисту переданих даних. Одним із ключових факторів успішної реалізації таких рішень є вибір відповідних інструментів і технологій, які дозволяють ефективно реалізувати функціональність чату та забезпечити надійність його роботи.

У рамках цієї курсової роботи розглядається процес створення браузерного онлайн-чату з використанням сучасних вебтехнологій. Аналізуються особливості побудови архітектури такого застосунку, принципи його роботи та методи реалізації ключових функцій.

Метою роботи є дослідження та реалізація вебзастосунку – онлайн-чату – як інструменту для сучасної онлайн–комунікації.

Для досягнення цієї мети потрібно виконати наступні завдання:

– ознайомитися з основними технологіями, що застосовуються при розробці браузерних чатів;

– розробити графічний інтерфейс користувача для зручної взаємодії з системою;

– реалізувати функціональні можливості чату, включаючи обмін повідомленнями в реальному часі;

– забезпечити належний рівень безпеки переданої інформації;

– оцінити зручність використання та роль онлайн-чатів у сучасному цифровому середовищі.

Об’єкт дослідження – технології створення браузерних чатів для мережевої комунікації на основі сучасних вебфреймворків.

Предмет дослідження – процес розробки та реалізації онлайн-чату з використанням React для клієнтської частини та Node.js для серверної логіки.

Апробація результатів роботи. Результати дослідження опубліковані у вигляді тез на тему «РОЗРОБКА ОНЛАЙН-ЧАТУ ЗАСОБАМИ NODE.JS», які увійшли до збірника матеріалів ІІ Міжнародної науково–практичної конференції «Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем». Конференція проходила 9–11 червня 2025 року на базі практик табору «Гарт» Волинського національного університету імені Лесі Українки (Шацький район, с. Світязь).

РОЗДІЛ 1
ПОНЯТТЯ ІНТЕРАКТИВНОЇ КОМУНІКАЦІЇ ТА ЇЇ ОСНОВНІ ЗАСОБИ

1.1 Еволюція цифрової взаємодії

Інтерактивна комунікація пройшла довгий шлях від простих технологій до сучасних багатофункціональних платформ, що забезпечують миттєвий обмін інформацією у реальному часі. Її розвиток можна розділити на кілька ключових етапів.

Перші кроки у сфері цифрової комунікації були зроблені завдяки електронній пошті та форумам, які дозволяли обмінюватися повідомленнями із затримкою у часі. Хоча цей формат не був миттєвим, він заклав основу для розвитку більш оперативних способів взаємодії.

З появою інтернету широкого доступу у 1990–х роках з'явилися перші месенджери, такі як ICQ, AOL Instant Messenger та MSN Messenger. Вони дозволяли користувачам спілкуватися в реальному часі через текстові повідомлення, що стало проривом у сфері цифрової комунікації [1].

У 2000–х роках соціальні мережі, такі як Facebook та MySpace, зробили онлайн–комунікацію ще доступнішою. Інтеграція функції миттєвого обміну повідомленнями дозволила користувачам спілкуватися безпосередньо через вебінтерфейс, що значно спростило взаємодію.

З появою смартфонів і мобільного інтернету на початку 2010–х років популярність набули мобільні месенджери, такі як WhatsApp, Telegram та Viber. Вони запропонували не лише текстові повідомлення, а й голосові дзвінки, відеочати, можливість надсилання файлів та використання ботів для автоматизації процесів.

Сучасні сервіси інтерактивної комунікації активно використовують штучний інтелект, що дозволяє автоматизувати відповіді, забезпечити персоналізацію спілкування та підвищити ефективність обслуговування клієнтів. Чат–боти та голосові асистенти стають важливими інструментами у різних сферах діяльності.

Основні характеристики сучасних сервісів інтерактивної комунікації:

– миттєвий обмін інформацією – мінімізація часу очікування відповіді;

– кросплатформеність – доступність на різних пристроях та операційних системах;

– автоматизація процесів – використання чат–ботів для швидкої обробки запитів;

– безпека та конфіденційність – захист даних за допомогою шифрування.

Еволюція інтерактивної комунікації продовжує розвиватися, відкриваючи нові можливості для бізнесу, соціальної взаємодії та обслуговування клієнтів.

1.2 Роль онлайн-чатів у сучасній цифровій екосистемі

У сучасному цифровому світі онлайн-чати стали ключовим елементом комунікації як для особистого спілкування, так і для бізнесу. Вони забезпечують швидку та ефективну взаємодію між користувачами, компаніями та автоматизованими системами. Завдяки інтеграції з іншими цифровими сервісами онлайн-чати суттєво впливають на якість обслуговування, маркетинг, продажі та навіть психологічний комфорт користувачів.

1.2.1 Онлайн-чати як інструмент комунікації

Основна перевага онлайн-чатів – це миттєва передача інформації без необхідності телефонних дзвінків чи довгого очікування відповіді електронною поштою. Вони активно використовуються в:

– клієнтському обслуговуванні – компанії впроваджують чати на сайтах для швидкої допомоги клієнтам, що покращує рівень задоволеності та лояльності;

– командній роботі – внутрішні корпоративні месенджери (Slack, Microsoft Teams) спрощують координацію між співробітниками та підвищують продуктивність;

– електронній комерції – чат-боти та консультанти у чатах допомагають користувачам знайти товари, отримати рекомендації та здійснити покупку.

1.2.2 Інтеграція з іншими технологіями

Онлайн-чати еволюціонують, доповнюючись інтелектуальними алгоритмами та новітніми технологіями:

– штучний інтелект – чат–боти з елементами AI можуть аналізувати запити, навчатися на основі взаємодій та надавати персоналізовані відповіді;

– голосові помічники – інтеграція онлайн-чатів із голосовими сервісами (Siri, Google Assistant) спрощує користування та розширює можливості спілкування [2];

– розпізнавання емоцій – новітні алгоритми можуть аналізувати тон повідомлень, що допомагає компаніям краще адаптувати свої відповіді.

1.2.3 Майбутнє онлайн чатів

Подальший розвиток онлайн-чатів буде зосереджений на ще глибшій інтеграції з іншими технологіями, підвищенні рівня автоматизації та персоналізації. Передбачається, що чати стануть ще більш розумними, забезпечуючи користувачам комфортну, безпечну та ефективну взаємодію в будь–якому цифровому середовищі.

1.2.4 Соціальний аспект онлайн-чатів

Онлайн-чати також відіграють важливу роль у соціальному аспекті:

– підтримка спільнот – чати створюють простір для обговорення та взаємодії між людьми з подібними інтересами, що сприяє формуванню онлайн–спільнот.

– психологічна підтримка – багато платформ пропонують чати для психологічної підтримки, де користувачі можуть анонімно звертатися за допомогою до фахівців.

– доступність інформації – онлайн-чати забезпечують швидкий доступ до інформації, що особливо важливо для людей з обмеженими можливостями, які можуть мати труднощі з традиційними формами комунікації.

1.3 Технологічні основи онлайн чатів

1.3.1 Використання WebSocket для реального часу

WebSocket – це протокол, який забезпечує двосторонню комунікацію між клієнтом і сервером через єдине з'єднання. Він був розроблений для подолання обмежень традиційних HTTP–запитів, які є односторонніми. Завдяки WebSocket, дані можуть передаватися в обох напрямках без необхідності повторного встановлення з'єднання, що робить його ідеальним для застосувань, де важлива швидкість і ефективність, таких як онлайн-чати [3].

WebSocket починає свою роботу з ініціалізації з'єднання через стандартний HTTP–запит. Після успішного встановлення з'єднання, протокол переходить у режим WebSocket, що дозволяє обмінюватися даними в реальному часі. Це з'єднання залишається відкритим, поки одна зі сторін не закриє його, що дозволяє зменшити затримки при передачі даних.

Переваги використання WebSocket :

– миттєва передача даних: WebSocket забезпечує миттєвий обмін повідомленнями, що є критично важливим для онлайн-чатів, де користувачі очікують швидкої реакції;

– ефективність: Завдяки збереженню відкритого з'єднання, WebSocket зменшує накладні витрати на повторні запити, що дозволяє знизити навантаження на сервер і зменшити затримки;

– двостороння комунікація: WebSocket дозволяє як клієнту, так і серверу ініціювати передачу даних, що робить його ідеальним для інтерактивних застосунків, де важлива швидка реакція на дії користувача;

– зменшення затримок: У порівнянні з традиційними методами, такими як опитування сервера (polling) або довге опитування (long polling), WebSocket значно знижує затримки, оскільки дані передаються миттєво, без необхідності повторних запитів.

У контексті онлайн-чатів WebSocket дозволяє реалізувати функціональність, яка забезпечує миттєвий обмін повідомленнями між користувачами. Коли один користувач надсилає повідомлення, воно миттєво передається іншим учасникам чату без затримок, що підвищує рівень взаємодії та задоволеності користувачів.

Крім того, WebSocket може бути використаний для реалізації додаткових функцій, таких як:

– сповіщення про нові повідомлення: Користувачі отримують миттєві сповіщення про нові повідомлення, навіть якщо вони не активні в чаті;

– індикатори набору тексту: Коли один користувач набирає повідомлення, інші можуть бачити індикатор, що підвищує інтерактивність;

– обмін медіа: WebSocket дозволяє швидко передавати не лише текстові повідомлення, але й медіа–файли, такі як зображення та відео.

Використання WebSocket у онлайн-чатах є важливим елементом, що забезпечує швидкість, ефективність та інтерактивність. Цей протокол дозволяє створювати сучасні, зручні та функціональні рішення для обміну повідомленнями, що відповідають вимогам користувачів у цифровому середовищі. Завдяки своїм перевагам, WebSocket стає стандартом для розробки реальних комунікаційних платформ, що робить його незамінним інструментом у сучасному веброзробці.

На рисунку 1.1 відображено два підходи до комунікації між клієнтом і вебсервером: традиційне надсилання запитів (polling) і використання WebSocket.

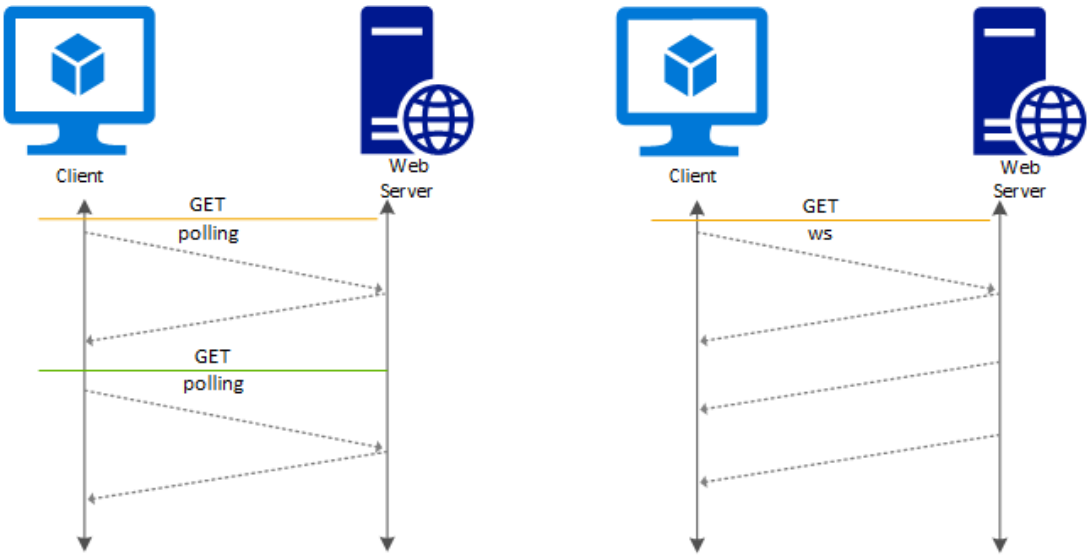


Рисунок 1.1 – Підходи до комунікації між клієнтом і вебсервером

Ліва частина ілюструє класичну модель, де клієнт надсилає запити GET до сервера, щоб отримати дані. Цей процес займає більше часу, оскільки клієнт періодично запитує нову інформацію, навіть коли змін немає. Це може призводити до витрат ресурсів.

Праворуч представлено WebSocket – протокол, який дозволяє встановити постійне з'єднання між клієнтом і сервером. Він забезпечує двосторонню комунікацію, що дозволяє серверу надсилати оновлення без необхідності повторних запитів. Це підвищує ефективність, знижує затримки та оптимізує використання ресурсів.

Зображення добре демонструє різницю між цими двома технологіями, підкреслюючи переваги WebSocket для сучасних вебдодатків.

1.3.2 Архітектура сервера на Node.js

Архітектура сервера на Node.js є важливим аспектом, що визначає ефективність і продуктивність вебдодатків. Node.js – це середовище виконання JavaScript, яке дозволяє створювати серверні додатки, використовуючи асинхронну, подієву модель. Це означає, що Node.js може обробляти велику кількість запитів одночасно, не блокуючи виконання інших операцій, що робить його ідеальним для розробки масштабованих мережевих додатків [4].

Основою архітектури сервера на Node.js є подієва модель, яка дозволяє обробляти запити асинхронно. Це досягається завдяки використанню неблокуючих вводу/виводу (I/O), що дозволяє серверу продовжувати виконання інших завдань, поки чекає на завершення операцій з базою даних або зовнішніми API. Завдяки цьому, Node.js може обслуговувати тисячі одночасних з'єднань, що робить його особливо корисним для реальних додатків, таких як чати, ігри або системи обробки даних у реальному часі.

Крім того, Node.js використовує однопоточну модель, що означає, що всі запити обробляються в одному потоці. Це зменшує накладні витрати на управління потоками, але вимагає від розробників уважності при написанні коду, щоб уникнути блокувань. Для цього в Node.js широко використовуються колбеки, проміси та async/await, що дозволяє зручно управляти асинхронними операціями.

Архітектура сервера на Node.js також передбачає використання модульної системи, що дозволяє розробникам організовувати код у зрозумілі та повторно використовувані модулі. Це сприяє кращій структуризації проекту та полегшує його підтримку. Багато популярних фреймворків, таких як Express.js, спрощують створення серверних додатків, надаючи готові рішення для маршрутизації, обробки запитів та роботи з middleware.

Важливим аспектом архітектури є також інтеграція з базами даних. Node.js підтримує різноманітні бази даних, як реляційні (наприклад, PostgreSQL, MySQL), так і нереляційні (наприклад, MongoDB). Використання ORM (Object–Relational Mapping) або ODM (Object–Document Mapping) бібліотек, таких як Sequelize або Mongoose, дозволяє спростити роботу з базами даних, забезпечуючи зручний інтерфейс для виконання запитів [5].

Завдяки своїй архітектурі, Node.js забезпечує високу продуктивність і швидкість розробки, що робить його популярним вибором для створення сучасних вебдодатків. Однак, для досягнення оптимальних результатів, важливо дотримуватися кращих практик програмування, таких як обробка помилок,

тестування та моніторинг продуктивності, що дозволяє створювати надійні та масштабовані рішення.

1.3.3 Управління базою даних MongoDB

Управління базою даних MongoDB є ключовим аспектом розробки сучасних вебдодатків, особливо в контексті використання Node.js. MongoDB – це документно–орієнтована NoSQL база даних, яка зберігає дані у форматі BSON (Binary JSON). Це дозволяє зберігати складні структури даних, включаючи вкладені документи та масиви, що робить MongoDB гнучким вибором для багатьох типів додатків [6].

Однією з основних переваг MongoDB є її схема, яка є динамічною. Це означає, що розробники можуть змінювати структуру документів без необхідності виконувати складні міграції, як це потрібно в реляційних базах даних. Така гнучкість дозволяє швидше адаптуватися до змін у вимогах проекту, що особливо важливо в умовах швидкої розробки та ітераційного підходу.

Для роботи з MongoDB у Node.js зазвичай використовують бібліотеку Mongoose, яка є об'єктно–документною мапою (ODM). Mongoose спрощує взаємодію з базою даних, надаючи розробникам можливість визначати моделі, схеми та валідацію даних. Це дозволяє зберігати дані у структурованому вигляді, а також забезпечує механізми для обробки помилок і валідації, що підвищує надійність додатків [7].

Коли мова йде про управління даними, MongoDB пропонує потужні можливості для виконання CRUD–операцій (створення, читання, оновлення, видалення). Завдяки простому і зрозумілому API, розробники можуть легко виконувати запити до бази даних, використовуючи JavaScript–стиль синтаксису. Наприклад, для отримання документів з колекції можна використовувати методи, такі як find(), findOne(), а для вставки нових документів – insertOne() або insertMany().

MongoDB також підтримує індексацію, що дозволяє значно підвищити швидкість виконання запитів. Індекси можуть бути створені на будь–якому полі

документа, що дозволяє оптимізувати пошук і фільтрацію даних. Це особливо важливо для великих обсягів даних, де без індексації запити можуть займати значний час.

Крім того, MongoDB забезпечує можливості для роботи з агрегацією даних, що дозволяє виконувати складні запити для отримання статистичних даних або обробки інформації. Агрегаційні функції, такі як $group, $match і $sort, дозволяють об'єднувати, фільтрувати та сортувати дані, що робить MongoDB потужним інструментом для аналітики.

Управління базою даних також включає в себе питання безпеки. MongoDB надає механізми аутентифікації та авторизації, що дозволяє контролювати доступ до даних. Розробники можуть налаштовувати ролі та дозволи для користувачів, що забезпечує захист чутливої інформації.

Важливим аспектом є також резервне копіювання та відновлення даних. MongoDB пропонує різні стратегії для резервного копіювання, включаючи використання mongodump і mongorestore, а також можливість налаштування реплікації для забезпечення високої доступності даних.

Загалом, управління базою даних MongoDB у контексті Node.js є потужним і гнучким рішенням, яке дозволяє розробникам ефективно працювати з даними, забезпечуючи високу продуктивність і масштабованість додатків. Завдяки своїй простоті у використанні та широким можливостям, MongoDB стає все більш популярним вибором для розробки сучасних вебдодатків.

1.3.4 Сучасні Front–end технології: React та Redux

Сучасний веброзробник не може обійтися без знання передових технологій, які забезпечують інтерактивність та динамічність вебдодатків. Однією з найпопулярніших бібліотек для створення користувацьких інтерфейсів є React, а для управління станом додатків часто використовують Redux. Ці технології разом створюють потужний інструментарій для розробки масштабованих і продуктивних фронтенд–рішень [8].

React – це бібліотека JavaScript, розроблена Facebook, яка дозволяє створювати компоненти, що повторно використовуються. Основна ідея React полягає в тому, щоб розбити інтерфейс на невеликі, ізольовані частини, які можна легко комбінувати. Кожен компонент має свій власний стан і логіку, що дозволяє розробникам зосередитися на окремих частинах інтерфейсу, не турбуючись про глобальний стан програми. Це підходить для створення складних інтерфейсів, оскільки компоненти можуть бути легко тестовані та підтримувані.

Однією з ключових особливостей React є віртуальний DOM. Замість того, щоб безпосередньо маніпулювати реальним DOM, React створює віртуальне представлення, яке дозволяє швидше виконувати оновлення. Коли стан компонента змінюється, React спочатку оновлює віртуальний DOM, а потім порівнює його з реальним DOM, вносячи лише необхідні зміни. Це значно підвищує продуктивність, особливо в додатках з великою кількістю елементів.

Redux, у свою чергу, є бібліотекою для управління станом, яка часто використовується разом з React. Основна мета Redux – централізувати стан додатка, що дозволяє уникнути проблем, пов'язаних з управлінням станом у великих додатках. У Redux стан зберігається в одному глобальному об'єкті, що робить його доступним для всіх компонентів. Це спрощує передачу даних між компонентами і дозволяє легко відстежувати зміни стану.

Основні концепції Redux включають дії (actions), редюсери (reducers) та сховище (store). Дії – це прості об'єкти, які описують, що сталося в додатку.

Рисунок 1.2 ілюструє принципи роботи Redux, популярного керування станом у JavaScript–додатках. Процес починається з дії (ACTION), яка є подією, що викликається користувачем, наприклад, натисканням кнопки. Дія передається у редюсер (REDUCER), який обробляє цю інформацію, використовуючи поточний стан (STORE) додатка. Редюсер визначає, як оновити стан на основі отриманої дії.

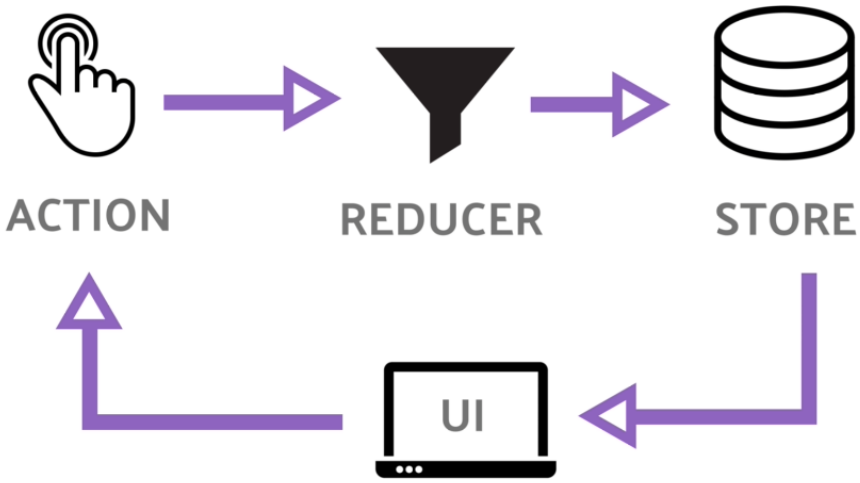


Рисунок 1.2 – Принцип роботи Redux

Після обробки у редюсері оновлений стан зберігається в магазині даних (STORE), звідки його може отримувати інтерфейс користувача (UI). Інтерфейс у свою чергу може реагувати на зміни стану, оновлюючи відображення для користувача. Таким чином, система працює за циклом: дія викликає зміну стану, а стан впливає на інтерфейс користувача.

Використання React і Redux разом дозволяє створювати складні, але організовані додатки. React відповідає за відображення інтерфейсу, тоді як Redux управляє станом, що робить архітектуру додатка більш зрозумілою і підтримуваною. Це особливо важливо в умовах командної розробки, де різні розробники можуть працювати над різними компонентами, але при цьому дотримуватися єдиної структури управління станом.

Завдяки своїй популярності та активній спільноті, React і Redux продовжують розвиватися, впроваджуючи нові функції та поліпшення. Це робить їх надійним вибором для розробки сучасних веб–додатків, які потребують високої продуктивності, масштабованості та зручності в обслуговуванні.

1.3.5 Комунікація в реальному часі за допомогою Socket.io

Socket.io є потужною бібліотекою, яка полегшує реалізацію двосторонньої взаємодії між веб–клієнтами та серверами. Вона забезпечує ефективну передачу

даних в режимі реального часу, що є критично важливим для багатьох сучасних вебдодатків, таких як чати, ігри та системи сповіщень. Використовуючи WebSocket та інші технології, Socket.io дозволяє клієнтам та серверам обмінюватися даними без затримок, що значно підвищує користувацький досвід [9].

При інтеграції Socket.io в проект початково необхідно налаштувати серверну частину, де сервер слухає підключення клієнтів. Після встановлення з'єднання сервер і клієнт можуть обмінюватися повідомленнями в режимі реального часу, що дозволяє анулювати традиційні запити на сервер. Це особливо корисно при обробці подій, таких як натискання кнопок або введення тексту, коли важливо отримувати миттєві оновлення. Наприклад, у чат–додатку, коли один користувач надсилає повідомлення, всі інші користувачі відразу отримують цю інформацію без потреби оновлення сторінки.

Socket.io також підтримує кімнати та простори, що дозволяє групувати користувачів за інтересами або функціями у додатку. Це відкриває нові можливості для інтерактивності, адже користувачі можуть бути учасниками певних „ кімнат “ для обговорень або ігор, отримуючи тільки ту інформацію, яка відноситься до їх групи. Відзначимо, що еволюція „залежить“ від стану UI, що є інтегровано через Redux або інші сучасні бібліотеки управління станом. Завдяки цій синергії, UI швидко оновлюється у відповідь на нові дані, що надходять з сервера, створюючи динамічний інтерфейс користувача.

На зображенні 1.3 представлено діаграму комунікації між клієнтом та сервером, яка ілюструє основні етапи обміну даними. Діаграма зазначає, як клієнт надсилає запити, а сервер на них реагує. Спочатку клієнт ініціює GET запит до сервера, який слугує для отримання інформації. Після цього клієнт виконує декілька POST запитів. Запити типу POST використовуються для надсилання даних на сервер, наприклад, коли користувач заповнює форму на вебсторінці.

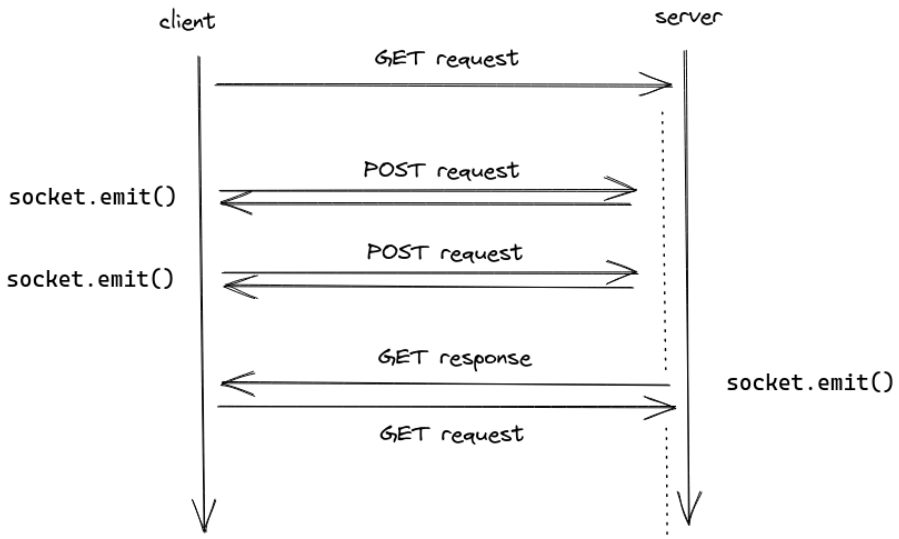


Рисунок 1.3 – Комунікація клієнт–сервера за допомогою Socket.io

Далі сервера відповідає на один з GET запитів, надсилаючи назад дані у формі GET відповіді. Важливим елементом є функція socket.emit(), яка відображає момент, коли клієнт або сервер взаємодіють через сокети для обміну даними в реальному часі. У діаграмі видно, що цей механізм взаємодії повторюється, свідчачи про динамічну комунікацію між клієнтом і сервером.

Таким чином, діаграма наочно демонструє принципи роботи запитів та відповідей у вебкомунікації, підкреслюючи важливість сокетів для реалізації інтерактивних функцій.

1.4 Архітектурні аспекти онлайн-чатів

Важливо реалізувати механізми автентифікації, щоб запобігти несанкціонованому доступу. Використання токенів доступу або сесійних Cookie є звичними практиками для захисту даних користувачів.

Додатково, для користувацького досвіду важливо реалізувати інтерфейс, що дозволяє легко переміщатися між чатами і мати можливість переглядати історію повідомлень. Це підвищує задоволеність користувачів та дозволяє підтримувати активність у чаті.

Зростаюча кількість одночасних підключень може призвести до перевантаження сервера. Для цього важливо реалізувати механізми

балансування навантаження та оптимізувати серверні ресурси, що забезпечить стабільну роботу системи при зростаючому інтересі з боку користувачів.

Щоб забезпечити кросплатформну сумісність, важливо тестувати систему на різних пристроях і браузерах, аби гарантувати безперебійну роботу сервісу для всіх користувачів.

Затримки в передачі даних можуть негативно вплинути на швидкість реакції чат–системи. Це потребує використання оптимізованих методів обробки запитів, таких як кешування та впровадження технологій, що забезпечують швидше отримання даних.

Таким чином, знання про основні аспекти розробки та усунення потенційних проблем здатні значно підвищити ефективність чат–систем.

1.5 Порівняльний аналіз популярних месенджерів

У таблиці 1.1 наведено порівняння трьох популярних месенджерів – Telegram, Messenger та Slack – з технічного боку. Вона дозволяє оцінити їхні можливості та відмінності, що важливо при виборі платформи для спілкування чи роботи [10].

Таблиця 1.1 – Порівняльний аналіз месенджерів

Характеристика	**Telegram**	**Messenger**	**Slack**
Платформи	Windows, macOS, Linux, iOS, Android, Web	iOS, Android, Web	Windows, macOS, Linux, iOS, Android, Web
Шифрування	End–to–End (у секретних чатах)	End–to–End (тільки для особистих повідомлень)	TLS, але немає повного End–to–End
Максимальний розмір файлу	2 ГБ	25 МБ	1 ГБ (у платній версії)
Підтримка ботів	Так (Потужний API)	Ні	Так (Через інтеграції)
Групові чати	До 200 000 учасників	До 250 учасників	До 1000 учасників (у платній версії)

Відеодзвінки	До 1000 глядачів	До 50 учасників	До 50 учасників (у платній версії)
Інтеграції з іншими сервісами	Обмежені	Мінімальні	Широкий вибір (Trello, Google Drive тощо)
Відкритий код	Частково (Клієнтський код відкритий)	Ні	Ні
Модерація контенту	Мінімальна	Сувора	Гнучка (власні правила для робочих просторів)
Основне призначення	Швидкі повідомлення та канали	Соцмережевий месенджер	Корпоративні комунікації

Одним із ключових аспектів є підтримка платформ. Telegram і Slack доступні на всіх основних пристроях, включаючи десктопні операційні системи та вебверсію, тоді як Messenger орієнтований переважно на мобільні пристрої.

Безпека теж відрізняється. Наприклад, Telegram забезпечує повне шифрування лише у секретних чатах, Messenger використовує його для приватних розмов, а Slack не має End–to–End шифрування, але пропонує загальний рівень захисту через протокол TLS.

Також варто звернути увагу на обмеження щодо файлів. Telegram дозволяє надсилати файли до 2 ГБ, що значно перевищує можливості Messenger і Slack.

Якщо говорити про ботів та інтеграції, Telegram має потужний API для створення ботів, Slack підтримує інтеграцію з багатьма сервісами для командної роботи, а Messenger у цьому плані поступається іншим платформам.

Групові чати, відеодзвінки та модерація контенту також мають свої особливості. Telegram дозволяє створювати великі спільноти, Slack орієнтований на корпоративне використання, а Messenger підходить більше для неформального спілкування.

Таким чином, Telegram більше підходить для загального використання та медіаконтенту, Messenger – для особистих розмов, а Slack – для командної роботи.

РОЗДІЛ 2
ТЕХНОЛОГІЇ РОЗРОБКИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

2.1. Постановка задачі, призначення та вимоги до розробки

Розробка сучасного онлайн-чату потребує комплексного підходу, який враховує як технічні, так і користувацькі аспекти функціонування системи. У добу активного використання цифрових технологій користувачі очікують від таких сервісів не лише зручності, але й надійності, високої швидкості обміну інформацією та захисту персональних даних. Онлайн-чати стали невід'ємною частиною як особистого спілкування, так і професійної взаємодії, тому до їх реалізації висуваються підвищені вимоги. Особливої уваги вимагають питання безпеки, інтерактивності та оптимізації під різні пристрої, що дозволяє забезпечити комфортне користування як на мобільних телефонах, так і на стаціонарних комп'ютерах.

Щоб забезпечити ефективну роботу онлайн-чату, важливо реалізувати низку функцій, які охоплюють як базові можливості для обміну повідомленнями, так і додаткові інструменти, що підвищують рівень взаємодії та довіри до платформи. Особливу роль відіграє застосування сучасних технологій – таких як React та Node.js, які забезпечують швидку роботу інтерфейсу, зручну архітектуру та масштабованість проєкту.

З урахуванням зазначеного, для створення ефективного, функціонального та безпечного онлайн-чату необхідно врахувати низку ключових вимог. Вони охоплюють як функціональні, так і нефункціональні аспекти, які спрямовані на забезпечення зручності, безпеки та високої продуктивності платформи:

– забезпечення безпечного входу до системи за допомогою JSONWebToken (JWT) [11];

– шифрування паролів користувачів для захисту даних;

– реалізація перевірки електронної пошти за допомогою одноразових паролів (OTP), що гарантує реєстрацію та вхід справжніх користувачів [12];

– забезпечення миттєвої передачі текстових повідомлень у чаті "один на один";

– відображення індикатора набору тексту для забезпечення зворотнього зв’язку в режимі реального часу;

– надсилання та отримання фотографій і відео без втрати якості;

– відображення статусу онлайн/офлайн контактів;

– можливість оновлення даних у профілі (зміна аватара, опис);

– адаптивний дизайн для всіх пристроїв;

– надсилання аудіоповідомлень із можливістю їх легко записувати та програвати.

2.2. Загальний опис проєкту

Онлайн-чат як вебзастосунок є багатофункціональною сучасною платформою, призначеною для швидкого, зручного та безпечного обміну інформацією між користувачами в режимі реального часу. У добу стрімкого розвитку цифрових технологій, коли миттєвий доступ до інформації стає критично важливим, такі системи стають основою ефективної мережевої комунікації. Основна мета створеного застосунку полягає в організації інтерактивного середовища, що сприяє активному, динамічному та водночас безпечному спілкуванню.

Технічна реалізація вебзастосунку базується на використанні передових технологій, зокрема таких як WebSocket, що дозволяє забезпечити передачу даних у режимі реального часу без затримок, створюючи ефект живого спілкування. Для досягнення високого рівня захисту конфіденційної інформації впроваджено шифрування паролів, а також перевірку електронної пошти за допомогою одноразових кодів (OTP), що дозволяє уникнути несанкціонованого доступу та зловживань.

Користувацький інтерфейс застосунку продуманий таким чином, аби гарантувати повну адаптацію до різних типів пристроїв – від настільних комп’ютерів до мобільних телефонів і планшетів. Ставка зроблена на простоту, інтуїтивність і зручність, що дозволяє користуватись чатом навіть людям без глибоких технічних знань. Важливо, що інтерфейс забезпечує не лише легкий

доступ до всіх функцій, але й естетичну привабливість, що позитивно впливає на загальний досвід користувача.

Застосунок має універсальний характер і підходить як для особистого листування, так і для використання в бізнес–середовищі, навчальних платформах або службах підтримки клієнтів. Гнучка архітектура дає змогу легко масштабувати систему під потреби різних проєктів та аудиторій.

На рисунку 2.1 представлено структурну діаграму проєкту, яка демонструє ключові етапи та логіку взаємодії користувача з системою. Вона включає основні процеси – від реєстрації з OTP–верифікацією до надсилання повідомлень, редагування профілю, завантаження медіафайлів та взаємодії з іншими учасниками чату. Діаграма чітко окреслює внутрішню логіку проєкту та ілюструє взаємозв'язки між його основними компонентами, що полегшує розуміння архітектурної будови рішення.

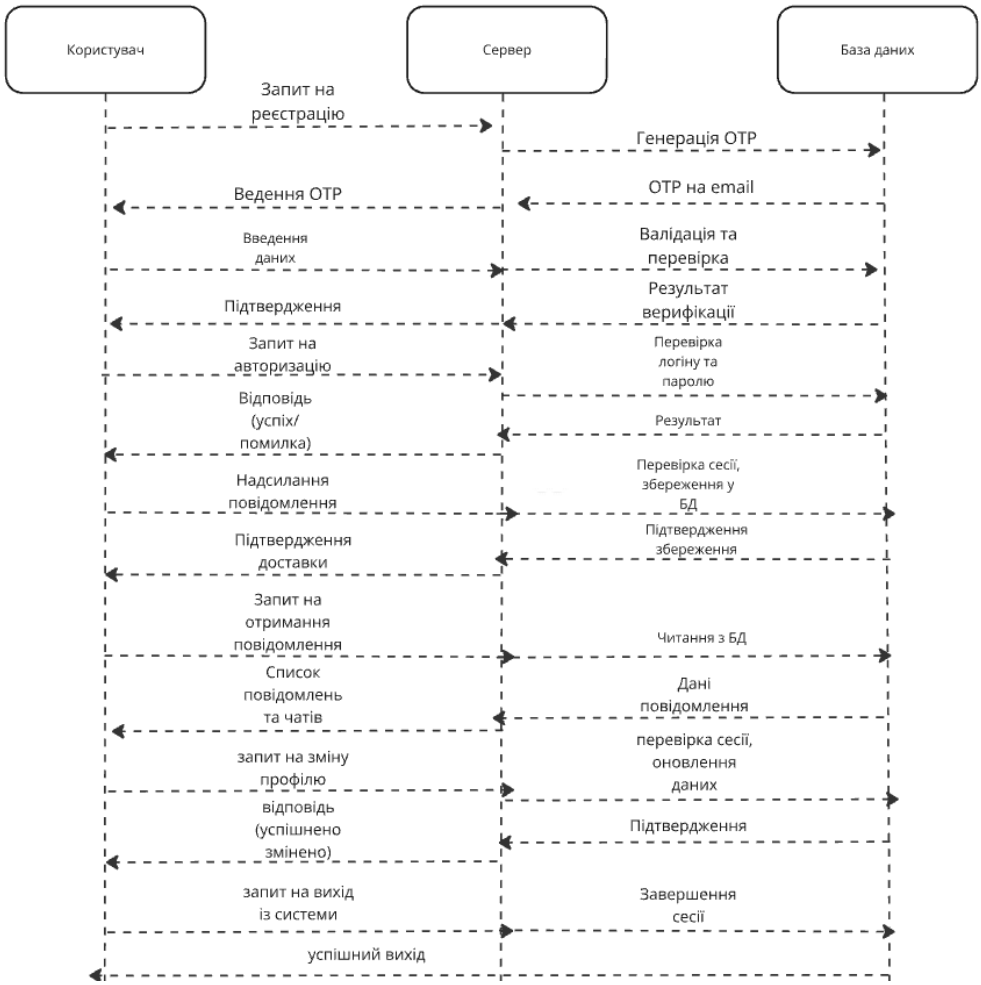


Рисунок 2.1 – Архітектура проєкту

2.3. Вибір моделі розробки

У процесі створення вебзастосунку онлайн-чату було обрано ітеративну модель розробки, яка найкраще відповідає характеру та динаміці проєкту. Ця модель дозволяє реалізовувати програмне забезпечення поетапно, поступово розширюючи функціонал за рахунок повторюваних циклів – ітерацій. Кожна з них передбачає проходження повного циклу розробки: від аналізу вимог до тестування та вдосконалення створеного модуля.

Розробка проєкту відбувалася у кілька ключових етапів. Спочатку було створено базовий дизайн інтерфейсу, після чого реалізовано серверну частину застосунку за допомогою Node.js. Наступним етапом стало розроблення клієнтської частини з використанням бібліотеки React, що дозволило досягти високої інтерактивності та зручності для користувача.

Важливою особливістю реалізації став поступовий приріст функціональності: у процесі розробки постійно додавались нові можливості, які не були передбачені початковими вимогами. Наприклад, уже після створення основної структури чату було впроваджено функцію надсилання аудіоповідомлень, що потребувало окремої роботи з мікрофоном, записом звуку та програванням. Такі зміни свідчать про гнучкий підхід до розробки та адаптацію проєкту до змінних потреб.

Ітеративна модель стала доцільною ще й тому, що дозволяла регулярно тестувати реалізований функціонал, оперативно виявляти недоліки та вносити необхідні коригування. Після кожного завершеного етапу відбувалося тестування частини застосунку, що дозволяло перевірити її працездатність та якість ще до інтеграції в загальну систему.

Загалом, вибір саме ітеративної моделі забезпечив гнучкість у розробці, можливість поступового вдосконалення застосунку та швидку реакцію на зміну вимог. Такий підхід сприяв створенню надійного й функціонального продукту, який відповідає сучасним очікуванням користувачів та забезпечує якісну взаємодію в режимі реального часу.

2.4. Обґрунтування вибору інструментальних засобів розробки

Для реалізації онлайн-чату було використано сучасний набір технологій, що забезпечують стабільну роботу, масштабованість, високу продуктивність та зручність у розробці.

Клієнтська частина застосунку створена з використанням React.js – популярної бібліотеки JavaScript з компонентною архітектурою. Такий підхід дозволяє легко створювати динамічні, адаптивні інтерфейси та ефективно керувати станом компонентів. Завдяки віртуальному DOM, React забезпечує швидке оновлення інтерфейсу без потреби повного перезавантаження сторінки, що є критично важливим для застосунків типу чатів, де обмін повідомленнями та зміна статусів повинні відбуватись у режимі реального часу. Крім того, у процесі розробки активно використовувалися численні сторонні бібліотеки для React (наприклад, React Router, Formik, React Icons тощо), які забезпечують додатковий функціонал, спрощують створення форм, навігації, стилізації та інтеграцію з іншими сервісами.

Серверна частина реалізована з використанням Node.js, що завдяки своїй подієво–орієнтованій і асинхронній архітектурі дозволяє ефективно обробляти велику кількість одночасних підключень. Для реалізації обміну повідомленнями в реальному часі було інтегровано Socket.IO, що забезпечує двосторонню комунікацію між клієнтом і сервером з мінімальними затримками.

MongoDB була обрана як база даних для зберігання інформації про користувачів, повідомлення, історію чатів тощо. Ця документо–орієнтована база даних дозволяє зберігати інформацію у форматі JSON, що ідеально підходить для структури чат–застосунку та дає змогу швидко адаптувати модель даних відповідно до змін у логіці проєкту.

Для зберігання мультимедійних файлів (зображень, аватарів, аудіо) використовувалась платформа Supabase, яка надає не лише інструменти для зберігання та обробки файлів, а й потужну серверну інтеграцію з базою даних та API для керування доступом до даних [13].

Збірка фронтенд–частини здійснювалася за допомогою Vite – сучасного високошвидкісного збирача, який забезпечує швидкий старт проєкту, підтримку гарячої заміни модулів (HMR) та миттєве оновлення змін у браузері. Це значно скорочує час на розробку та тестування, особливо при частих змінах у коді [14].

Увесь процес розробки вівся в середовищі Visual Studio Code (VSCode) – легкому, але потужному редакторі коду з широкими можливостями розширення, інтеграцією з Git, зручним дебаггінгом та підтримкою великої кількості мов програмування [15].

Таким чином, обраний стек технологій дозволив створити ефективний, надійний і масштабований застосунок, що відповідає сучасним вимогам до веброзробки та забезпечує зручну взаємодію між користувачами в режимі реального часу.

2.5. Особливості програмної реалізації

Розробка програмного забезпечення “Онлайн-чат” відбувається поетапно, при цьому кожен етап відіграє важливу роль у формуванні повноцінної, стабільної та зручної у використанні системи.

На початковому етапі визначаються ключові функціональні можливості майбутнього застосунку: система автентифікації, обмін повідомленнями в реальному часі, інструменти для адміністрування, забезпечення захисту даних тощо. Паралельно проводиться аналіз цільової аудиторії та добір відповідних технологій для реалізації задуманої функціональності.

Формується загальна структура системи, включаючи клієнтську та серверну частини, базу даних і серверне середовище. Особливу увагу приділяється тому, щоб архітектура була гнучкою, масштабованою та готовою до майбутніх змін.

1. Спочатку розробляється інтерфейс користувача, який має бути зрозумілим, привабливим і адаптованим для роботи на різних пристроях. До нього входить реалізація основних функцій, таких як перегляд повідомлень, введення тексту, індикація статусу користувачів тощо.

2. Серверна частина забезпечує обробку клієнтських запитів, збереження повідомлень, управління користувацькими сесіями, а також обробку реального часу та взаємодію з базою даних.

3. Після створення окремих компонентів здійснюється їх інтеграція – об’єднання клієнтської та серверної частин у єдину систему з повною функціональністю. На цьому етапі перевіряється коректність взаємодії між компонентами.

4. Фінальний етап передбачає всебічне тестування: юніт–тестування, перевірка взаємодії модулів та тестування з боку кінцевого користувача. Виявлені недоліки усуваються, а система оптимізується для покращення продуктивності та стабільності.

На початковому етапі було створено новий проєкт за допомогою Vite – інструменту швидкої збірки для застосунків на React. Після ініціалізації проєкту були встановлені всі необхідні залежності з використанням npm – основного менеджера пакунків для JavaScript, що дозволяє швидко підключати перевірені й підтримувані бібліотеки [16].

Серед ключових бібліотек, які були інтегровані у проєкт:

– React та React–DOM – для побудови інтерфейсу користувача на основі компонентного підходу;

– Redux разом із React–Redux – для ефективного централізованого управління станом застосунку;

– React Router DOM – для налаштування маршрутизації, що забезпечує перемикання між сторінками;

– Socket.IO Client – для реалізації обміну повідомленнями в режимі реального часу;

– Axios – для виконання HTTP–запитів до серверної частини;

– Tailwind CSS – для створення сучасного та адаптивного дизайну;

– Supabase – як бекенд–рішення для зберігання даних, автентифікації користувачів та роботи з мультимедіа.

Окрім цього, використовувались і додаткові бібліотеки, що сприяють покращенню зручності використання та взаємодії користувача із системою.

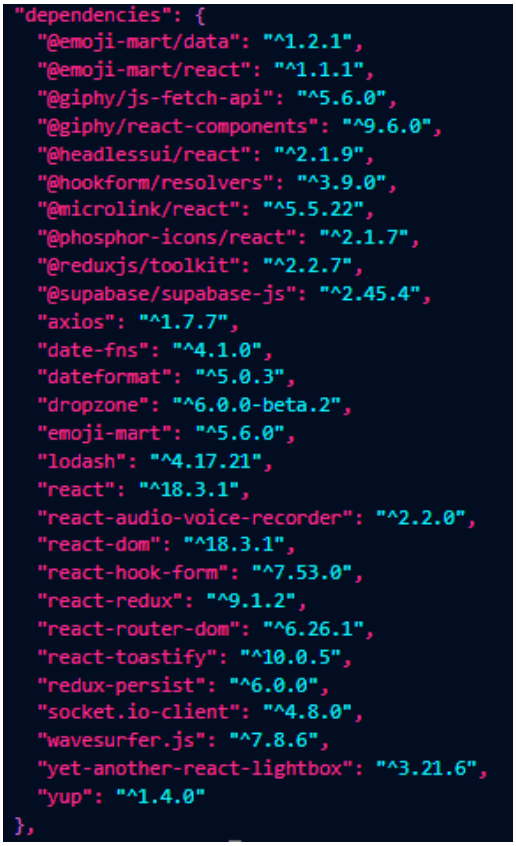


Рисунок 2.2 – Всі бібліотеки та модулі проєкту

Проєкт складається з численних компонентів, які логічно пов’язані між собою. Кожен компонент займає певне місце в загальній структурі інтерфейсу і може функціонувати окремо або в комплексі з іншими. Комунікація між компонентами відбувається через передавання пропсів та керування станом. Така модульна організація забезпечує гнучкість, зручність у масштабуванні та легкість у внесенні змін і додаванні нового функціоналу.

На рисунку 2.3 налаштовується сервер для реального часу, використовуючи бібліотеку Socket.IO, яка дозволяє створювати інтерактивні додатки, в яких користувачі можуть обмінюватися повідомленнями без необхідності оновлювати сторінку.

Код починається з імпорту необхідних модулів. Модуль authSocket відповідає за автентифікацію користувачів, перевіряючи їхні права на підключення до серверу. Інші модулі є обробниками різних подій: відключення

користувача, отримання історії чату, надсилання нових повідомлень і обробка індикаторів друку.

Далі створюється сервер за допомогою бібліотеки Socket.IO, який підключається до основного HTTP–серверу. Сервер налаштовується таким чином, щоб дозволяти запити з будь–яких джерел через CORS (перехресний доступ), що важливо для роботи з клієнтськими додатками, розміщеними на різних доменах. Для кожного з'єднання з сервером виконується автентифікація за допомогою middleware, який перевіряє права користувача.

Якщо автентифікація успішна, сервер реагує на події. Основна подія connection запускається, коли користувач успішно підключається до серверу. При цьому сервер записує ідентифікатор сокету користувача. Після підключення користувач може взаємодіяти з сервером через різні події. Наприклад, коли користувач відправляє нове повідомлення, подія new–message обробляється за допомогою функції newMessageHandler, яка надсилає повідомлення іншому користувачеві чи групі. Також є подія direct–chat–history, яка обробляє запити на отримання історії чату.

Для зручності користувачів на сервері реалізовано механізм індикації активності. Коли користувач починає друкувати повідомлення, сервер посилає індикатор через подію start–typing іншому користувачеві, який може побачити, що інша особа активно набирає текст. Коли введення припиняється, викликається подія stop–typing, щоб зупинити індикатор друку.

Для того, щоб стежити за статусом користувачів (онлайн чи офлайн), використовується метод setInterval, який кожні 8 секунд оновлює інформацію про з'єднання користувачів. Цей механізм дозволяє ефективно працювати з динамічними з'єднаннями і оновлювати статуси без необхідності вручну оновлювати сторінки.

Весь цей процес відбувається на сервері, і забезпечує коректну роботу чату в реальному часі, де користувачі можуть обмінюватися повідомленнями, переглядати історію чату, отримувати індикатори активності та працювати з мультимедійними файлами.

```
const registerSocketServer = (server) => {
  const io = require("socket.io")(server, {
    cors: {
      origin: "*",
      method: ["GET", "POST"],
    },
  });

  io.use((socket, next) => {
    authSocket(socket, next);
  });

  io.on("connection", (socket) => {
    console.log("user connected");
    console.log(socket.id);

    // newConnectionHandler
    newConnectionHandler(socket, io);

    socket.on("disconnect", () => {
      disconnectHandler(socket);
    });

    socket.on("new-message", (data) => {
      newMessageHandler(socket, data, io);
    });

    socket.on("direct-chat-history", (data) => {
      chatHistoryHandler(socket, data);
    });

    socket.on("start-typing", (data) => {
      startTypingHandler(socket, data, io);
    });

    socket.on("stop-typing", (data) => {
      stopTypingHandler(socket, data, io);
    });
  });

  setInterval(() => {
    // emit online user
  }, [1000 * 8]);
};

module.exports = { registerSocketServer };
```

Рисунок 2.3 – Код налаштування серверу Socket.io

Сторінка реєстрації реалізована у вигляді компонента SignUp.jsx. Тут користувач вводить свої особисті дані: ім’я, адресу електронної пошти та підтвердження пароля. Для перевірки коректності введених даних застосовується бібліотека yup [17]. Вона перевіряє, чи заповнені всі поля, чи має

пошта правильний формат, чи складається пароль щонайменше з шести символів і чи збігається підтвердження пароля з основним паролем.

```
const schema = yup.object().shape({
  name: yup.string().required("Name is required"),
  email: yup
    .string()
    .email("Invalid email format")
    .required("Email is required"),
  password: yup
    .string()
    .min(6, "Password must be at least 6 characters")
    .required("Password is required"),
  confirmPassword: yup
    .string()
    .oneOf([yup.ref("password")], "Passwords must match")
    .required("Please confirm your password"),
});
```

Рисунок 2.4 – Код валідації даних користувача

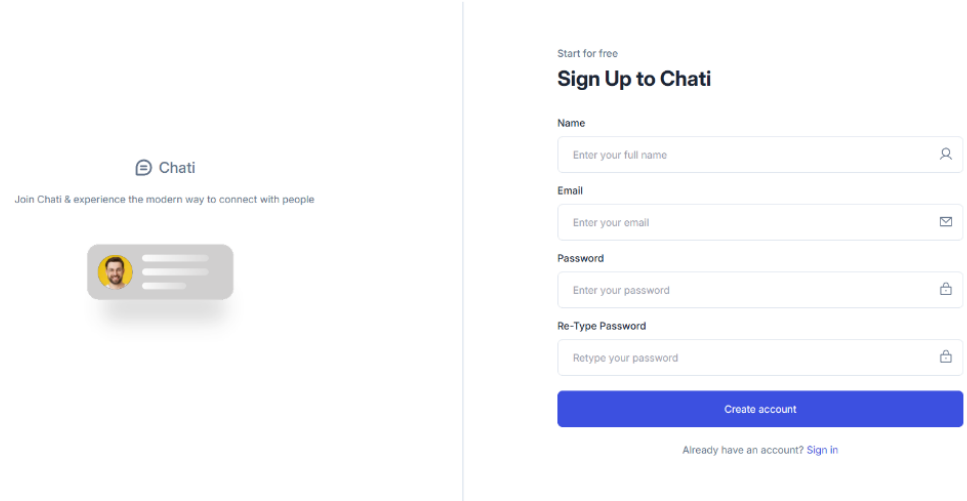


Рисунок 2.5 – Сторінка реєстрації у браузері

Реєстрація користувачів у системі реалізована за допомогою бази даних MongoDB, яка підключається до серверної частини через бібліотеку Mongoose. На рисунку 2.6 зображено код, що відповідає за підключення до бази даних, де за допомогою змінного середовища MONGO_URI встановлюється з’єднання. У разі успішного підключення до бази сервер запускається та прослуховує вхідні запити.

```
const app = require("./app");
const dotenv = require("dotenv");

const mongoose = require("mongoose");
const socketServer = require("./socketServer");

dotenv.config({ path: "./config.env" });

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

const http = require("http");
const server = http.createServer(app);
socketServer.registerSocketServer(server);

// console.log(process.env.MONGO_URI);

mongoose
  .connect(process.env.MONGO_URI)
  .then(() => {
    console.log("MongoDB connection successful!");
    server.listen(PORT, () => {
      console.log(`Server is listening on ${PORT}`);
    });
  })
  .catch((err) => {
    console.log("database connection failed. Server not started");
    console.log(err);
  });
```

Рисунок 2.6 – Підключення до бази даних

Збереження даних нового користувача реалізовано за допомогою схеми userSchema (рис. 2.7), яка описує основні поля: ім’я, посада, біо, країна, аватар, електронна пошта, пароль, статус, OTP (одноразовий код), а також технічні змінні, як–от час створення чи оновлення запису. Схема також включає вбудовані механізми валідації, наприклад перевірку формату електронної пошти та унікальність цього поля.

```
const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Name is required"],
      trim: true,
    },
    // jobTitle
    jobTitle: {
      type: String,
      trim: true,
    },
    // bio
    bio: {
      type: String,
      trim: true,
    },
    // country
    country: {
      type: String,
    },
    // Avatar
    avatar: {
      type: String,
    },
    email: {
      type: String,
      required: [true, "Name is required"],
      validate: {
        validator: function (email) {
          return validator.isEmail(email);
        },
        message: (props) => `Email (${props.value}) is invalid!`,
      },
      unique: true,
    },
    password: {
      type: String,
    },
    passwordChangedAt: {
      type: Date,
    },
    verified: {
      type: Boolean, // true, false
      default: false,
    },
    otp: {
      type: String,
    },
    otp_expiry_time: {
      type: Date,
    },
```

Рисунок 2.7 – userSchema реєстрації

Перед збереженням нового користувача в базі даних виконується „ pre–save “ хук – це функція, яка автоматично хешує пароль та OTP, якщо вони були змінені. Це забезпечує додатковий рівень безпеки, оскільки у базі не зберігаються відкриті паролі. Також у схемі реалізовано методи перевірки правильності пароля або OTP під час аутентифікації.

У результаті система реєстрації дозволяє безпечно зберігати облікові дані користувача, валідувати введені дані та забезпечувати їх надійний захист за допомогою шифрування.

Рисунок 2.8 – Запис користувача у базу даних

Далі йде сторінка верифікації поштової адреси (компонент Verification.jsx), на якій користувач повинен ввести одноразовий пароль (OTP), що був надісланий на його електронну пошту за допомогою Nodemailer [18].

Для надсилання одноразових паролів (OTP) на електронну пошту користувача використовується бібліотека Nodemailer. У проєкті створено окремий модуль, який відповідає за формування та відправку листів. Спочатку за допомогою dotenv завантажуються конфіденційні дані з .env файлу, зокрема логін та пароль поштового акаунту, що використовуються для авторизації в сервісі Gmail.

Налаштовується об’єкт transporter через nodemailer.createTransport, в якому вказано сервіс (Gmail) та облікові дані. Далі створюється функція Mailer, яка приймає ім’я користувача, його email та згенерований OTP. Ця функція формує лист, використовуючи HTML–шаблон (згенерований окремим компонентом), і надсилає його користувачу. У разі помилки відправки вона фіксується у консолі та генерується відповідне повідомлення про помилку.

Таким чином, цей модуль забезпечує безпечну і стабільну відправку кодів підтвердження на електронну адресу для верифікації облікового запису.

```
const nodemailer = require("nodemailer");
const OTPTemplate = require("./../Template/OTP");
const dotenv = require("dotenv");

dotenv.config({ path: "./config.env" });

// const NODEMAILER_USER = process.env.NODEMAILER_USER;
// const NODEMAILER_APP_PASSWORD = process.env.NODEMAILER_APP_PASSWORD;

let transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: process.env.GMAIL_USER,
    pass: process.env.GMAIL_APP_PASSWORD,
  },
});

const Mailer = async ({ name, otp, email }) => {
  let mailOptions = {
    to: email, // Кому відправляється лист
    subject: "Verify your Chati Account",
    html: OTPTemplate({ name, otp }), // HTML-шаблон для листа
  };

  try {
    let info = await transporter.sendMail(mailOptions);
    console.log("Email sent: %s", info.messageId);
  } catch (error) {
    console.error("Error sending email: ", error);
    throw new Error("Error sending mail");
  }
};

module.exports = Mailer;
```

Рисунок 2.9 – Код для відправки OTP

Для введення OTP створено спеціальну форму, яка перевіряє, чи складається код з чотирьох цифр. У випадку, якщо код введено неправильно, користувач отримає повідомлення про помилку.

Сторінка включає кілька основних елементів: логотип, заголовок, форму для введення OTP та кнопку для його відправки. Також передбачена кнопка для повторної відправки коду, яку можна активувати, якщо користувач не отримав повідомлення з OTP. Ця кнопка має таймер, який обмежує можливість повторного запиту протягом 60 секунд. Після закінчення цього часу кнопка повторної відправки стає доступною для нового запиту.

Цей компонент забезпечує зручний і плавний процес верифікації електронної пошти, а також обробляє помилки у разі неправильного введення коду або проблем з повторною відправкою OTP.

На рисунку 2.10 зображено інтерфейс сторінки верифікації в браузері, а на рисунку 2.11 – лист із кодом для підтвердження, який отримує користувач.

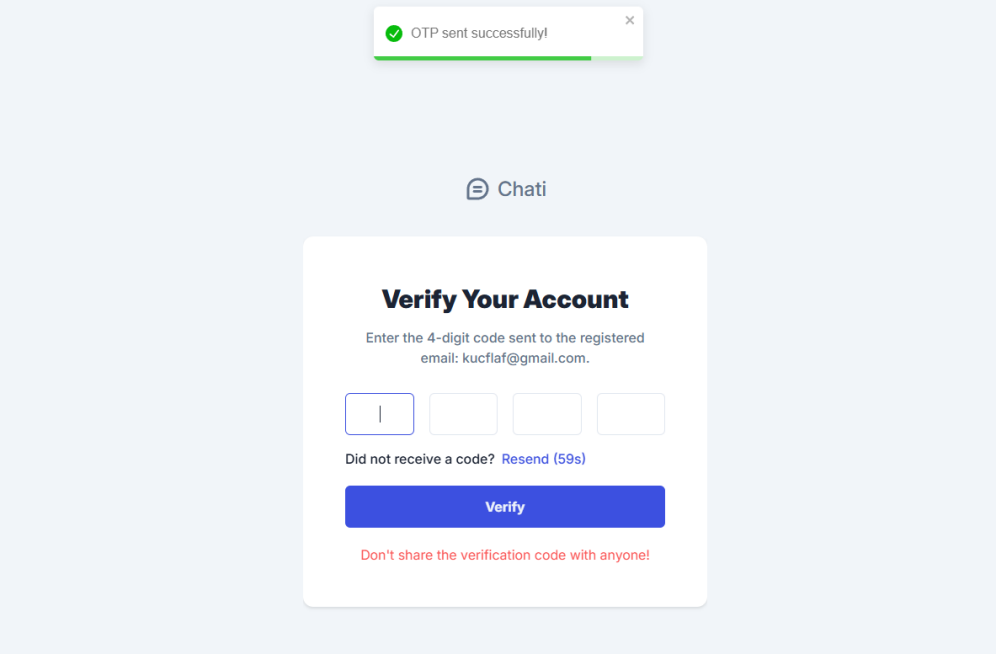


Рисунок 2.10 – Інтерфейс сторінки верифікації

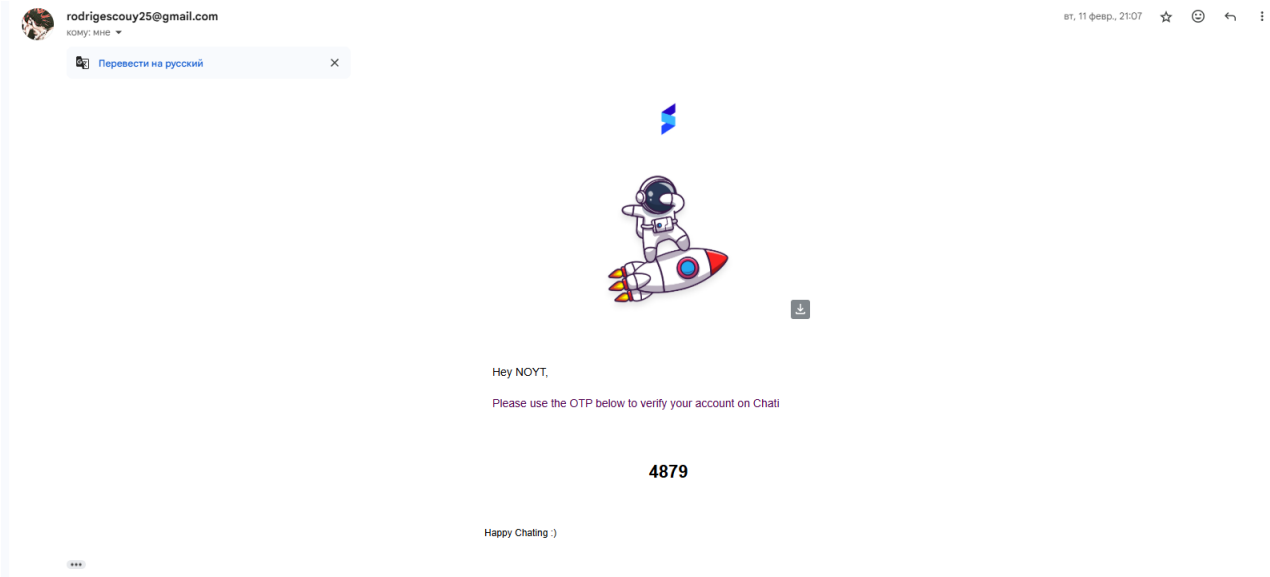


Рисунок 2.11 – Лист з кодом підтвердження

Після успішної верифікації користувач потрапляє на сторінку чату, де відображається список чатів (поки що порожній). Користувач може одразу налаштувати свій профіль, щоб інші користувачі могли його побачити. Інтерфейс налаштування профілю показано на рисунку 2.12.

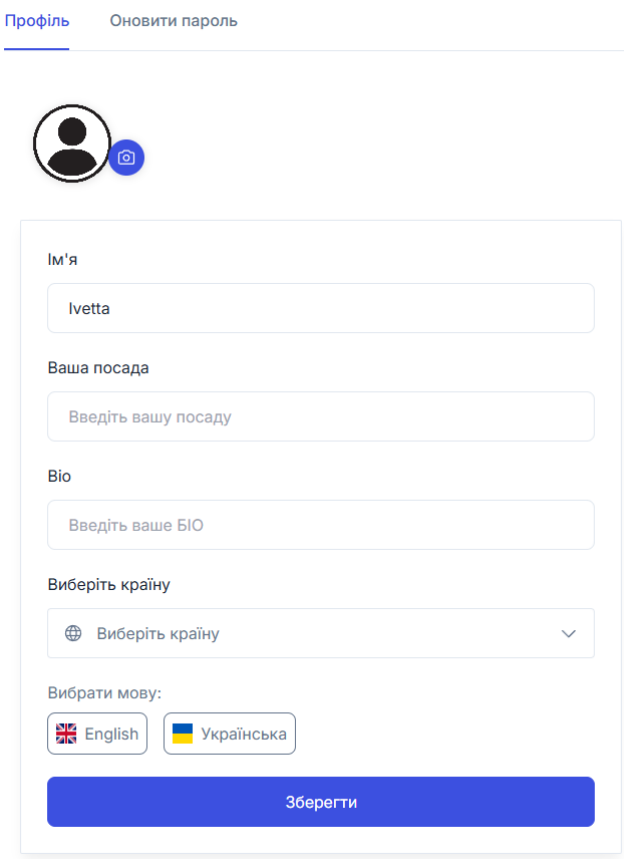


Рисунок 2.12 – Інтерфейс сторінки налаштування профілю

На сторінці налаштувань профілю користувач може змінити основні персональні дані: ім’я, посаду, біографічну інформацію (біо), обрати країну зі списку та змінити мову інтерфейсу. Мова реалізована за допомогою бібліотеки i18next [19], яка дозволяє перемикати інтерфейс між англійською та українською мовами динамічно – без перезавантаження сторінки. Обрана мова зберігається за допомогою localStorage, тому навіть після оновлення сторінки вибір мови залишається незмінним, і весь інтерфейс автоматично підлаштовується під відповідну локалізацію. Це створює зручний досвід користувача, забезпечуючи сталість у відображенні контенту тією мовою, яка була обрана раніше.

```jsx
import React from "react";
import { useTranslation } from "react-i18next";
import i18n from "./Lang.jsx"; //

const LanguageSelector = () => {
  const { t } = useTranslation();

  const changeLanguage = (lng) => {
    i18n.changeLanguage(lng);
  };

  return (
    <div className="flex items-center space-x-4">
      <button
        onClick={() => changeLanguage("en")}
        className="flex items-center p-2 border border-gray-300 rounded-lg hover:bg-gray-100 transition"
      >
        <img
          src="https://upload.wikimedia.org/wikipedia/commons/thumb/8/83/Flag_of_the_United_Kingdom_%283-5%29.svg/800px-Flag_of_the_United_Kingdom_%283-5%29.svg.png"
          alt="English"
          className="w-5 h-5 mr-2"
        />
        {t("English")}
      </button>
      <button
        onClick={() => changeLanguage("uk")}
        className="flex items-center p-2 border border-gray-300 rounded-lg hover:bg-gray-100 transition"
      >
        <img
          src="https://upload.wikimedia.org/wikipedia/commons/4/49/Flag_of_Ukraine.svg"
          alt="Ukrainian"
          className="w-5 h-5 mr-2"
        />
        {t("Українська")}
      </button>
    </div>
  );
};

export default LanguageSelector;
```

Рисунок 2.13 – Код компонента LangSelector.jsx

Також передбачена окрема вкладка для зміни пароля, яка доступна у будь–який момент. Це дозволяє користувачеві самостійно оновити свій пароль з міркувань безпеки або у разі, якщо виникла потреба його змінити. Така можливість підвищує загальний рівень безпеки облікового запису та дає користувачеві більше контролю над своїми даними (рис. 2.14).

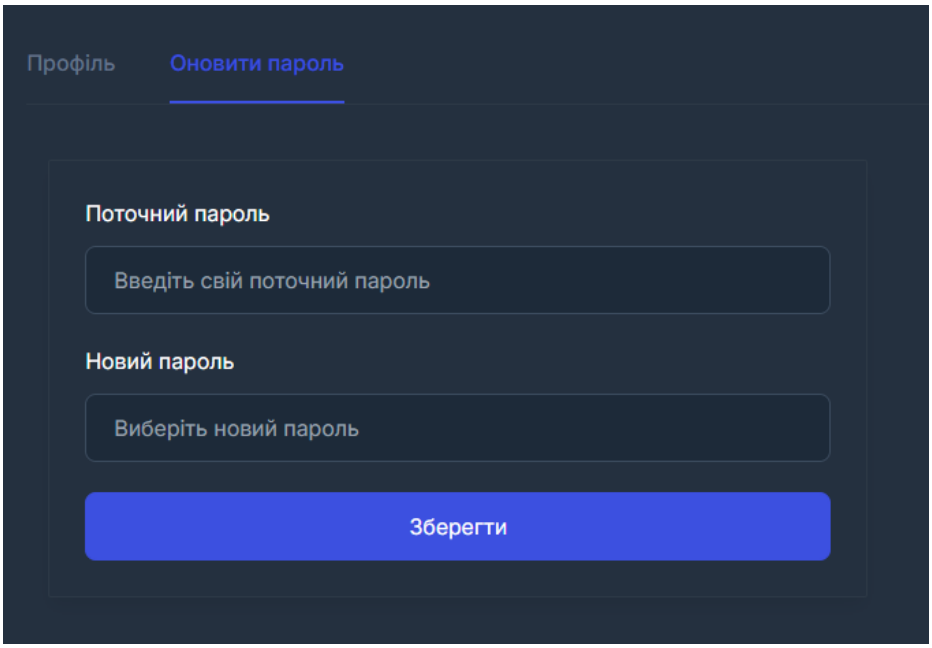


Рисунок 2.14 – Інтерфейс сторінки зміни паролю

Після завершення реєстрації та налаштування профілю користувач переходить на сторінку зі списком інших зареєстрованих користувачів. Це дає можливість обрати співрозмовника для подальшого спілкування. Відповідальність за відображення та керування цим списком несе компонент ChatList.jsx (рис. 2.15). Він забезпечує функціональність пошуку серед чатів, вибір активної розмови та створення нових чатів.

```
export default function ChatList({ open, handleClose }) {
  const dispatch = useDispatch();
  const [addConversation, setAddConversation] = useState(false);
  const [searchTerm, setSearchTerm] = useState("");
  const [debouncedSearchTerm, setDebouncedSearchTerm] = useState("");
  const [deletedConversations, setDeletedConversations] = useState([]);
  const [menuOpen, setMenuOpen] = useState(null); // State to manage the open
  const { t } = useTranslation();
  const { conversations, currentConversation } = useSelector(
    (state) => state.chat
  );
  const { user } = useSelector((state) => state.user);

  useEffect(() => {
    dispatch(GetConversations());
  }, [dispatch]);

  const handleToggleConversation = () => {
    setAddConversation((p) => !p);
  };

  const handleSelectConversation = (id) => {
    dispatch(SetCurrentConversation(id));
  };

  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedSearchTerm(searchTerm);
    }, 500);

    return () => clearTimeout(handler);
  }, [searchTerm]);

  const handleSearchChange = (e) => {
    setSearchTerm(e.target.value);
  };

  const handleDeleteConversation = (id) => {
    setDeletedConversations((prev) => [...prev, id]);
    setMenuOpen(null); // Close the menu after deletion
  };

  const filteredConversations = conversations
    .map((el) => {
      const other_user = el.participants.find((e) => e._id !== user._id);
      if (!other_user || deletedConversations.includes(el._id)) return null;
      return {
        key: el._id,
        id: el._id,
        name: other_user.name,
        imgSrc: other_user.avatar,
        message: "",
        status: other_user.status,
      };
    })
    .filter(Boolean);
```

Рисунок 2.15 – Код компонента ChatList.jsx

Після вибору співрозмовника (рис. 2.16) відкривається вікно чату, де користувачі можуть вести діалог у режимі реального часу. Такий підхід забезпечує зручну, динамічну та безперебійну комунікацію між учасниками.

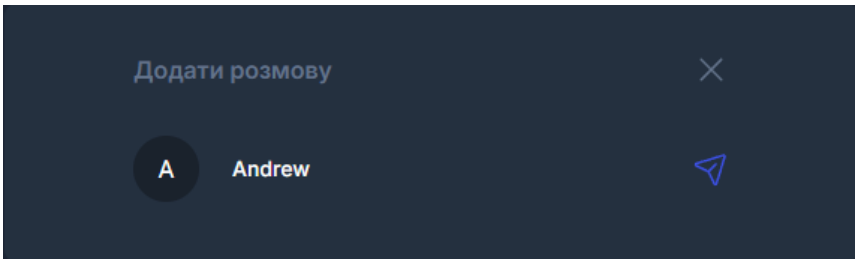


Рисунок 2.16 – Модальне вікно вибору співрозмовника

У чаті користувачі мають можливість не лише обмінюватися текстовими повідомленнями, але й надсилати різноманітні мультимедійні файли, що робить спілкування значно гнучкішим та емоційно насиченішим. Система підтримує відправлення голосових повідомлень, що є зручним способом передавання інформації, особливо коли користувач не має змоги писати. Окрім цього, передбачено надсилання фотографій, відео, гіфок, а також будь–яких інших медіафайлів.

Також підтримуються смайлики – користувачі можуть обирати їх із доступної колекції, що дозволяє виражати емоції без потреби в додатковому тексті. Це робить спілкування більш живим і неформальним.

Однією з важливих функцій є індикатор статусу користувача – в реальному часі відображається, чи співрозмовник перебуває в мережі (Online), або наразі відсутній (Offline). Це дозволяє краще орієнтуватися в тому, коли очікувати на відповідь.

Ще одна корисна функція – це індикатор набору тексту. Коли користувач починає вводити повідомлення, його співрозмовник бачить, що він друкує – наприклад, у вигляді повідомлення “Користувач друкує...”. Це створює ефект живого діалогу та очікування відповіді в реальному часі.

Усі повідомлення – як текстові, так і медіа – зберігаються в історії чату. Завдяки цьому користувачі мають змогу у будь–який момент повернутись до

попередніх розмов, переглянути важливі моменти або віднайти необхідну інформацію.

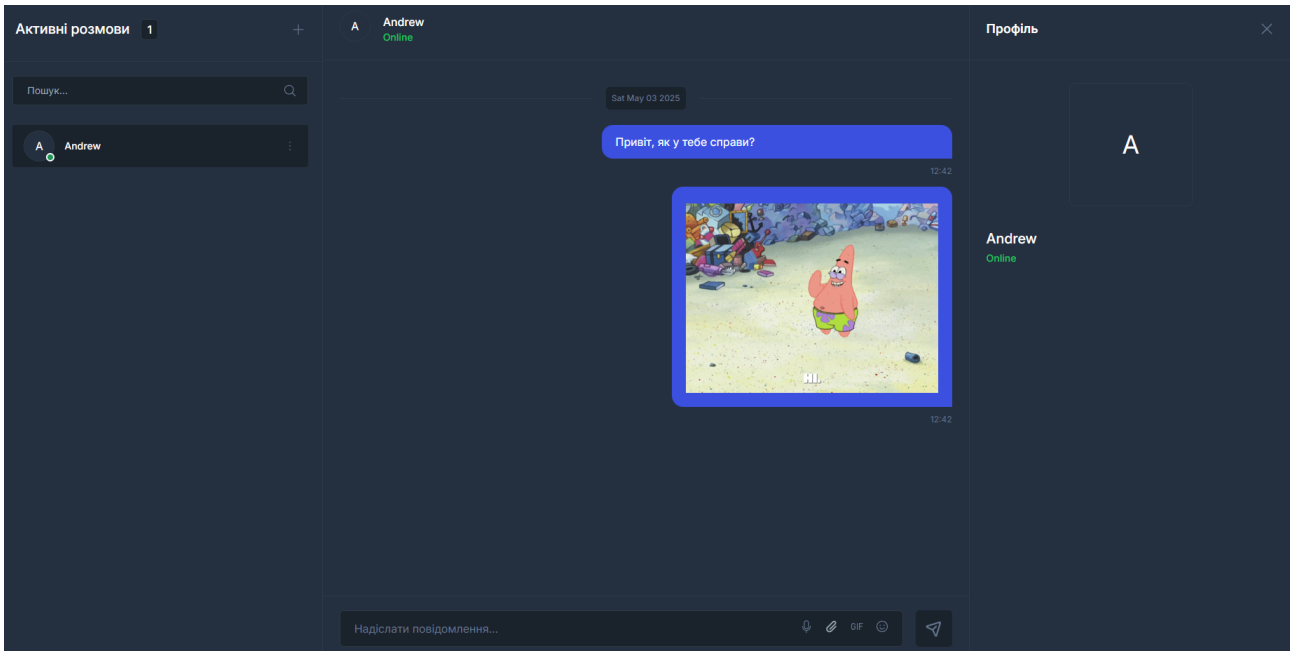


Рисунок 2.17 – Інтерфейс чату

Процес відправки повідомлень включає кілька етапів, особливо коли мова йде про мультимедійний контент, та його зберігання. Повідомлення, що містять текст, голосові повідомлення, фото, відео, документи або гіфки, спочатку обробляються та зберігаються у відповідній базі даних. Для цього використовується модель повідомлення в MongoDB, яка описує структуру даних для кожного повідомлення.

В залежності від типу контенту, у повідомлення можуть бути вкладені різні медіа–елементи:

– Медіафайли (фото, відео) зберігаються як URL–адреси, що вказують на файли у Supabase;

– Голосові повідомлення зберігаються у вигляді аудіофайлів, з посиланням на їх розташування у Supabase;

– Документи (наприклад, PDF–файли) зберігаються окремо, зберігаючи посилання на сам файл;

– Гіфки зберігаються за допомогою гіфок, де також є URL на зовнішні ресурси.

Вся ця інформація зберігається у базі даних MongoDB, де для кожного повідомлення є відповідний запис з усіма метаданими (час відправки, тип медіа, автор, тощо).

Медіафайли не зберігаються безпосередньо в MongoDB, а відправляються на сервер зберігання файлів Supabase. Тут файли зберігаються у спеціальному бакеті для чатів, і після завантаження користувач отримує публічний URL для доступу до файлу.

Процес завантаження файлів на Supabase виконується за допомогою методу uploadToSupabase, який відповідає за очистку імені файлів від неприпустимих символів, визначення MIME типу та безпосереднє завантаження на сервер зберігання. Після цього повертається публічний URL файлу, який зберігається у відповідному полі для медіа в MongoDB.

Це все відбувається на стороні сервера, що забезпечує ефективне та безпечне зберігання медіафайлів та текстових повідомлень, дозволяючи відправляти повідомлення з різними типами контенту без зайвого навантаження на базу даних.

```
import { createClient } from "@supabase/supabase-js";

const supabaseUrl = "https://sbovhcxxgivzvwmtrgqe.supabase.co";
const supabaseKey =
  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InNib3ZoY3h4Z2l2enZ3bXR
const supabase = createClient(supabaseUrl, supabaseKey);

export const uploadToSupabase = async (file) => {
  // Очищаємо ім'я файлу від неприпустимих символів
  const sanitizedFileName = `uploads/${Date.now()}_${file.name.replace(
    /[^a-zA-Z0-9.-_]/g,
    "_"
  )}`;

  // Визначаємо MIME тип файлу
  const contentType = file.type || "application/octet-stream"; // За замовчуванням, якщо MIME т

  const { data, error } = await supabase.storage
    .from("chat")
    .upload(sanitizedFileName, file, {
      upsert: false,
      cacheControl: "3600",
      contentType, // Встановлюємо MIME тип
    });

  if (error) {
    console.error("Error details:", error); // Лог для додаткової інформації про помилку
    throw new Error("Error uploading file: " + error.message);
  }

  const urlData = supabase.storage.from("chat").getPublicUrl(sanitizedFileName);

  return urlData?.data?.publicUrl; // Повертаємо публічний URL файлу
};
```

Рисунок 2.18 – Код для збереження мультимедійного контенту

2.6 Тестування та налагодження програмної розробки

У процесі розробки “Онлайн-чату” було проведено ґрунтовне тестування та налагодження системи, що дозволило виявити й успішно усунути низку критичних проблем. Після проведення юніт–тестів, інтеграційного тестування, перевірки користувацького інтерфейсу та тестування в режимі реального часу, система була вдосконалена та оптимізована для стабільної роботи.

Зокрема, усунуто помилки у взаємодії між фронтендом і бекендом – тепер усі повідомлення, надіслані через WebSocket, коректно передаються іншим користувачам без затримок. Також було оптимізовано серверну обробку запитів, що дозволило зменшити навантаження і забезпечити швидке доставлення повідомлень.

Валідація введених даних на фронтенді була посилена: система тепер блокує порожні або некоректні повідомлення, а логіка автентифікації на бекенді

була повністю перероблена, що виключає можливість несанкціонованого входу до чату.

Інтерфейс програми також зазнав суттєвих покращень. Було оновлено CSS–стилі та адаптовано верстку для мобільних пристроїв, завдяки чому всі елементи коректно відображаються на різних розмірах екранів, а анімації працюють плавно і без збоїв.

Таким чином, після внесених змін і доопрацювань, продукт повністю готовий до використання. Його функціональність стабільна, інтерфейс зручний і адаптивний, а система – безпечна та надійна для користувачів.

2.7. Рекомендації по впровадженню та використання програмної розробки

Застосунок “Онлайн-чат” відкриває широкі можливості для впровадження в різноманітні галузі – від електронної комерції до корпоративного обслуговування. Він здатний виступати ефективним каналом для миттєвого зв’язку як між користувачами, так і між клієнтами та технічною підтримкою. Такий інструмент дозволяє підвищити оперативність обробки звернень, скоротити час відповіді й значно покращити загальний користувацький досвід.

Завдяки гнучкій архітектурі, чат легко інтегрується як у вебплатформи, так і в мобільні застосунки. Його можна використовувати як у внутрішніх системах компаній для координації роботи співробітників, так і у відкритих платформах для комунікації з клієнтами в реальному часі.

Для безперебійного функціонування системи важливо враховувати низку технічних умов. Насамперед, стабільне та швидкісне інтернет–з'єднання є ключовою передумовою для забезпечення якісного обміну повідомленнями, особливо у випадках великого трафіку або під час пікових навантажень.

Щодо серверної інфраструктури, необхідно використовувати потужне апаратне забезпечення, здатне обробляти паралельні з’єднання та запити користувачів. Бажано мати сервер із щонайменше 4–ядерним процесором, 8 ГБ оперативної пам’яті, достатнім дисковим простором (приблизно 100 ГБ) для

зберігання історії чатів, логів і технічних даних. Рекомендована операційна система – Linux (наприклад, Ubuntu чи CentOS), а як вебсервер – Nginx або Apache. Для управління базами даних оптимальними є MySQL чи PostgreSQL.

З боку користувача система повинна функціонувати без збоїв у найпопулярніших браузерах (Google Chrome, Mozilla Firefox, Safari), а також на мобільних пристроях. Щодо мобільних платформ, підтримка має охоплювати версії Android не нижче 8.0 і iOS не нижче 11.0. Особливу увагу слід приділити адаптивному дизайну інтерфейсу, щоб забезпечити зручність користування незалежно від типу пристрою.

Загалом, “Онлайн-чат” може стати невіддільною частиною будь–якої цифрової екосистеми, підвищуючи рівень комунікації, ефективності та довіри з боку користувачів. У подальшому система може бути розширена за рахунок додаткових функцій, таких як чат–боти, автоматичне збереження історії переписок, інтеграція з CRM–системами або аналітика поведінки користувачів у чаті.

ВИСНОВКИ

У межах кваліфікаційної роботи було здійснено повний цикл розробки програмного продукту типу „Онлайн-чат“, який призначений для забезпечення оперативної, зручної та безперебійної комунікації між користувачами та службами підтримки. Основна мета роботи – створити інструмент для інтерактивного спілкування, який можна легко інтегрувати в вебплатформи або мобільні додатки – була успішно досягнута.

На етапі дослідження було проведено аналітичний огляд сучасних рішень у сфері онлайн–комунікацій. Вивчено переваги та недоліки існуючих чат–систем, що дало змогу сформулювати вимоги до власного програмного рішення. Особлива увага приділялася забезпеченню стабільності, сумісності з різними платформами та простоті користування.

У процесі розробки була реалізована архітектура клієнт–сервер, яка включає повноцінний бекенд для обробки повідомлень і запитів користувачів, а також адаптивний інтерфейс для фронтенду. Проведено проєктування бази даних для зберігання історії чатів, логів та іншої службової інформації. Особливо важливо, що програмний продукт протестовано на відповідність сучасним вимогам безпеки та стабільності роботи під навантаженням.

Важливим результатом є те, що чат–клієнт було оптимізовано для різних браузерів (Google Chrome, Mozilla Firefox, Safari) та мобільних операційних систем (Android, iOS). Це дозволяє забезпечити максимально широкий доступ для користувачів незалежно від їхнього пристрою.

Створений програмний продукт може бути успішно використаний в різноманітних галузях – від служби підтримки інтернет–магазинів до внутрішніх корпоративних систем зв’язку. Його впровадження дозволяє підвищити рівень обслуговування, зменшити витрати часу на обробку звернень, а також поліпшити загальний досвід користувача.

У майбутньому розробка може бути доповнена новим функціоналом, зокрема:

– впровадженням чат–ботів з елементами штучного інтелекту;

– додаванням системи автоматичних повідомлень та шаблонів відповідей;

– інтеграцією з CRM або ERP–системами компанії;

– використанням аналітичних інструментів для моніторингу активності користувачів.

Отже, розроблений чат–сервіс демонструє практичну цінність, технічну ефективність та потенціал для масштабування. Він є сучасним, функціональним і відповідає актуальним потребам цифрового ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Як месенджери змінили спілкування. *Домени – перевірка та реєстрація доменів в Україні | Imena.ua*. URL: https://www.imena.ua/blog/the–messenger–changed–humanity/ (дата звернення: 05.05.2025).

2. Голосові помічники: що це і навіщо вони HR [Ok Google, Alexa, Siri, Cortana] | HURMA. *HURMA*. URL: https://hurma.work/blog/voice–assistants–shho–cze–i–navishho–voni–hr–ok–google–alexa–siri–cortana/ (дата звернення: 05.05.2025).

3. WebSocket. *Сучасний підручник з JavaScript*. URL: https://uk.javascript.info/websocket (дата звернення: 05.05.2025).

4. Index | Node.js v23.11.0 Documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 05.05.2025).

5. Типи, особливості та сфери застосування баз даних. *IT Education Center Blog*. URL: https://itedu.center/ua/blog/articles/databases–types–and–features/?srsltid=AfmBOopEXA1vqmgBzhzgrrr5UMu5Yl5EjVccnmVqORCQ4hYlrTmn5hOy (дата звернення: 05.05.2025).

6. MongoDB with JavaScript – MongoDB Documentation – MongoDB Docs. *MongoDB: The World's Leading Modern Database | MongoDB*. URL: https://www.mongodb.com/docs/languages/javascript/ (дата звернення: 05.05.2025).

7. Mongoose v8.14.1: Schemas. *Mongoose ODM v8.14.1*. URL: https://mongoosejs.com/docs/guide.html (дата звернення: 05.05.2025).

8. Getting Started with React Redux | React Redux. *React Redux | React Redux*. URL: https://react–redux.js.org/introduction/getting–started (дата звернення: 05.05.2025).

9. Introduction | Socket.IO. *Socket.IO*. URL: https://socket.io/docs/v4/ (дата звернення: 05.05.2025).

10. Порівняння популярних корпоративних месенджерів | HURMA. *HURMA*. URL: https://hurma.work/blog/porivnyannya–populyarnih–korporativnih–mesendzheriv/ (дата звернення: 05.05.2025).

11. JSON Web Tokens. *Auth0 Docs*.
URL: https://auth0.com/docs/secure/tokens/json–web–tokens (дата звернення: 05.05.2025).

12. Одноразовий пароль (OTP) для авторизації: чи гарантує він повний захист даних?. *Business SMS service – SMS API. Send bulk SMS and Voice Globally.* URL: https://decisiontele.com/uk/news/one–time–password–otp–for–authorization–does–it–guarantee–complete–data–protection.html (дата звернення: 05.05.2025).

13. Use Supabase with React | Supabase Docs. *Supabase | The Open Source Firebase Alternative*. URL: https://supabase.com/docs/guides/getting–started/quickstarts/reactjs (дата звернення: 05.05.2025).

14. Getting Started. *vitejs*. URL: https://vite.dev/guide/ (дата звернення: 05.05.2025).

15. Microsoft. Tutorial: Get started with Visual Studio Code. *Visual Studio Code – Code Editing. Redefined*.
URL: https://code.visualstudio.com/docs/getstarted/getting–started (дата звернення: 05.05.2025).

16. About npm | npm Docs. *npm Docs*. URL: https://docs.npmjs.com/about–npm/ (дата звернення: 05.05.2025).

17. yup. *npm*. URL: https://www.npmjs.com/package/yup (дата звернення: 05.05.2025).

18. Nodemailer | Nodemailer. *Nodemailer | Nodemailer*.
URL: https://nodemailer.com/ (дата звернення: 05.05.2025).

19. Getting started | i18next documentation. *Introduction | i18next documentation*. URL: https://www.i18next.com/overview/getting–started (дата звернення: 05.05.2025).

ДОДАТКИ
ДОДАТОК А
Технічне завдання

1. Опис проєкту

Проєкт представляє собою багатофункціональну вебплатформу для обміну повідомленнями в режимі реального часу. Користувачі мають можливість вести текстове листування, надсилати медіафайли (зображення, відео, GIF, голосові повідомлення), документи різного формату, використовувати смайлики та переглядати історію чатів.

2. Функціональні вимоги

2.1 Основний функціонал чату

– Надсилання текстових повідомлень у реальному часі.
– Надсилання фото, відео, голосових повідомлень.
– Підтримка GIF та смайликів.
– Надсилання файлів різних форматів (PDF, DOCX, ZIP тощо).
– Відображення історії переписки.

2.2. Функціональність користувача

– Реєстрація та авторизація.
– Редагування профілю.
– Зміна паролю.
– Перемикання мови інтерфейсу.
– Можливість створення чату.
– Технічні вимоги

3.1. Технологічні розробки

– Frontend: React, Tailwind, JavaScript
– Backend: Node.js (Express)
– База даних: MongoDB
– Інтеграція: Supabase (окремі функції)
– Інші бібліотеки: Socket.io (для обміну повідомленнями в реальному часі), Multer (для завантаження файлів), i18next (для інтернаціоналізації)

3.2. Платформа та сумісність

- Кросплатформений інтерфейс (адаптивна верстка)
- Сумісність з сучасними веббраузерами: Chrome,Firefox,Safari, Edge
- Коректна робота на мобільних пристроях (iOS,Android)

3. Безпека

- Захищена аутентифікація користувачів
- Хешування паролів (bcrypt)
- Валідація форм вводу
- Контроль доступу до файлів і даних користувачів

ДОДАТОК Б
Інструкція користувачу

Даний посібник містить докладну інформацію, необхідну для зручного та результативного використання сайту. Він допоможе користувачам ознайомитися з ключовими діями та особливостями роботи з функціоналом вебресурсу.

Щоб розпочати роботу з проєктом, слід виконати декілька базових кроків:

1. Увійдіть до свого облікового запису GitHub;

2. Знайдіть репозиторій, у якому зберігаються файли проєкту;

3. Скопіюйте URL–адресу цього репозиторію;

4. Відкрийте термінал на вашому комп'ютері та виконайте відповідні команди для обох папок (див. рис. Б1).

```
1. cd <folder name>
2. npm install
3. npm run dev (для фронтенд), npm start (для серверу)
```

Рисунок Б1 – Команди для запуску програми локально

Після цього автоматично відкриється браузер із вказаною адресою. Перед початком встановлення проєкту важливо переконатися, що ваш пристрій відповідає наступним технічним вимогам:

– Операційна система: Windows 10, macOS 11 або актуальна версія будь–якого сучасного дистрибутиву Linux;

– Процесор: щонайменше двоядерний із частотою не нижче 2.0 ГГц;

– Оперативна пам'ять: мінімум 4 ГБ (рекомендовано 8 ГБ для стабільної роботи);

Надійне інтернет–з'єднання для роботи з Git та завантаженням залежностей.

– Необхідне програмне забезпечення:

– Node.js (бажано версія з довготривалою підтримкою – LTS);

– Git для роботи з репозиторієм;

– Зручне середовище для редагування коду (наприклад, Visual Studio Code або WebStorm);

– Пакетний менеджер npm або yarn (за потреби).

Після встановлення всіх необхідних компонентів та успішного налаштування проєкту, ви можете переходити до використання чату.

Початковою сторінкою проєкту є форма реєстрації, яка дає змогу користувачам створити обліковий запис для подальшого доступу до функцій чату (див. рис. Б2). Вона включає обов’язкові поля для заповнення. Після введення даних система здійснює перевірку їхньої правильності, зокрема формат електронної пошти та рівень складності пароля.

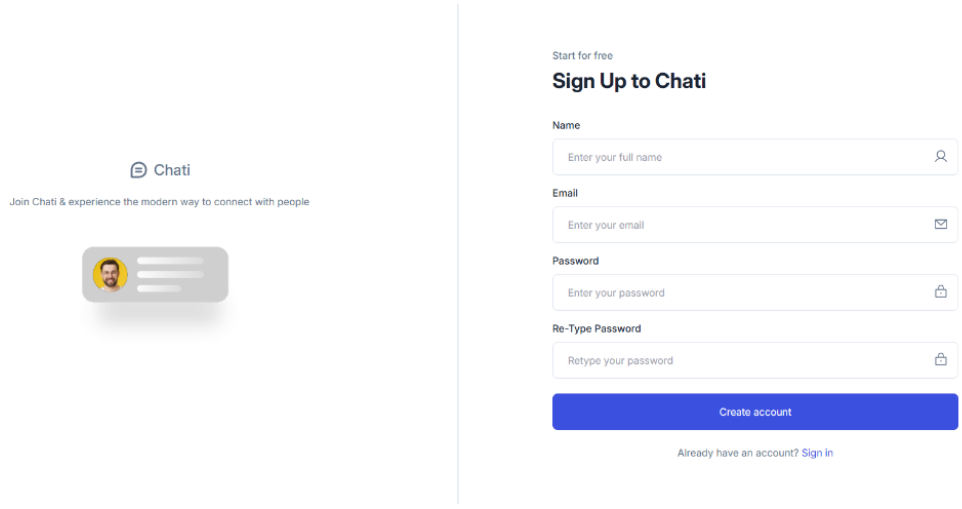


Рисунок Б2 – Сторінка реєстрації

Якщо всі дані введено коректно, на вказану електронну адресу буде надіслано лист для підтвердження облікового запису (див. рис. Б3). У випадку помилок система відобразить повідомлення поруч із відповідними полями введення. Для користувачів, які вже зареєстровані, на сторінці доступне посилання, що дозволяє швидко перейти до форми входу (рис. Б5).

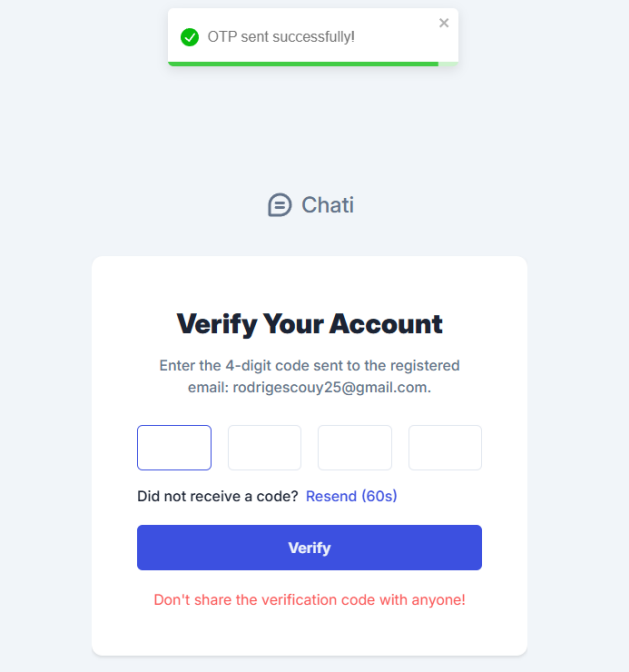


Рисунок Б3 – Сторінка верифікації електронної адреси

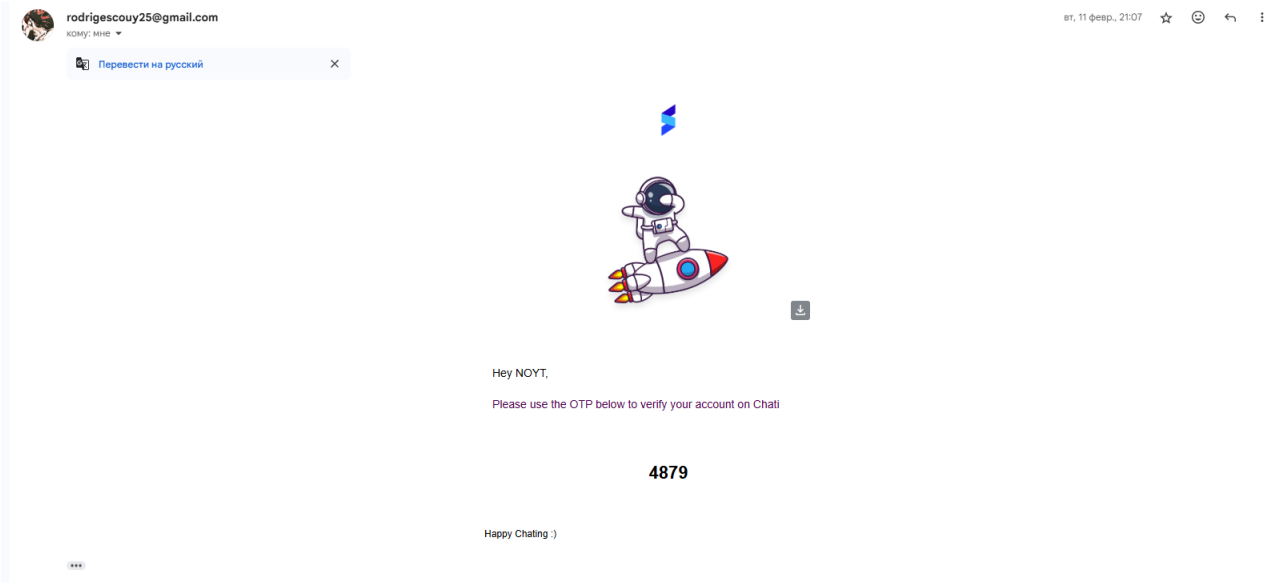


Рисунок Б4 – Лист із підтвердженням

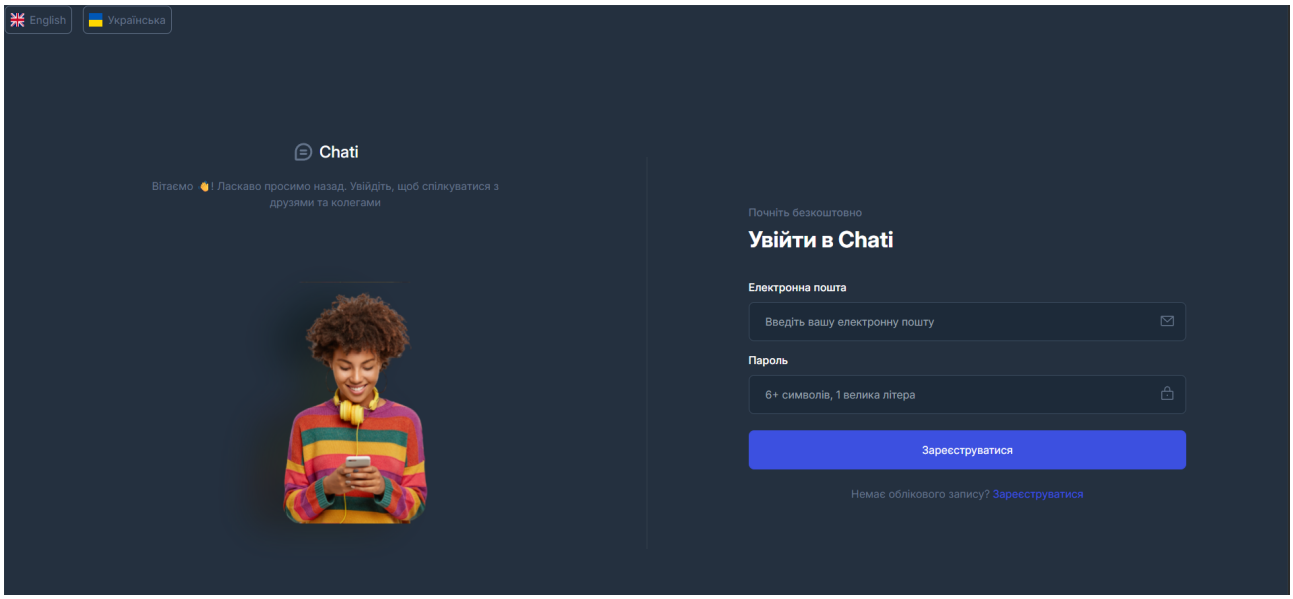


Рисунок Б5 – Сторінка авторизації

Після успішного підтвердження електронної пошти або входу в систему, користувач автоматично перенаправляється на головну сторінку застосунку. Тут відображається перелік доступних чатів, до яких користувач має доступ і може долучитися (див. рис. Б7).

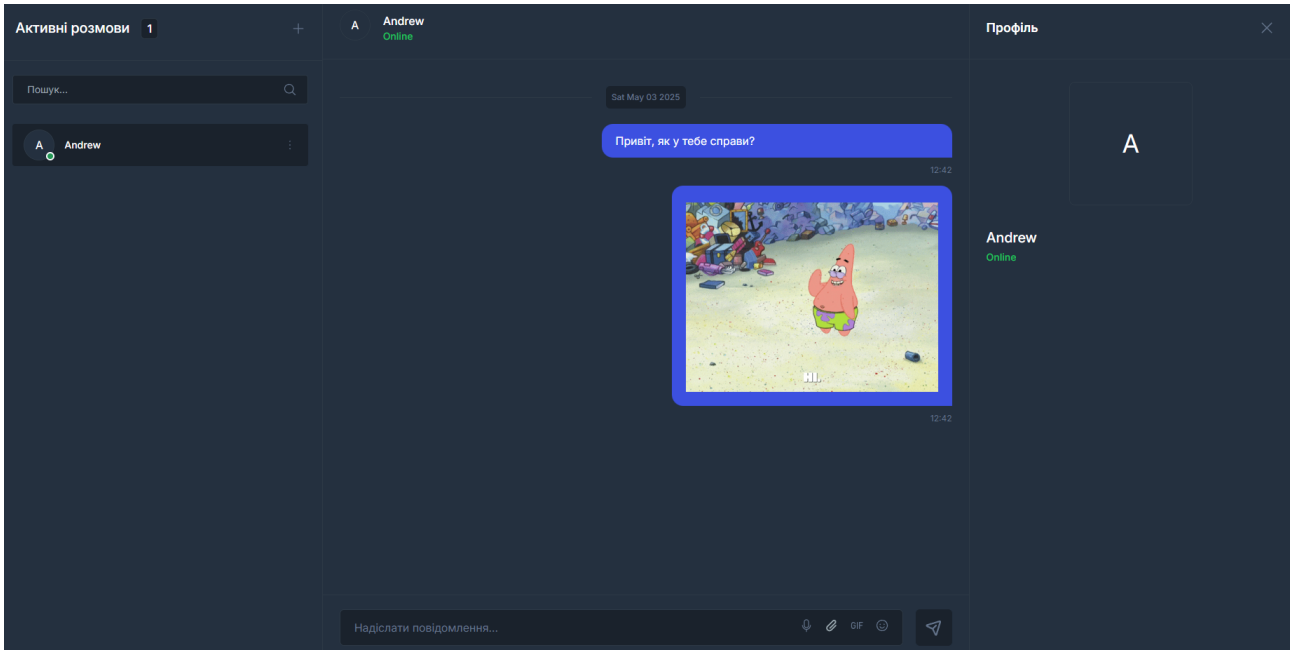


Рисунок Б6 – Головна сторінка чату

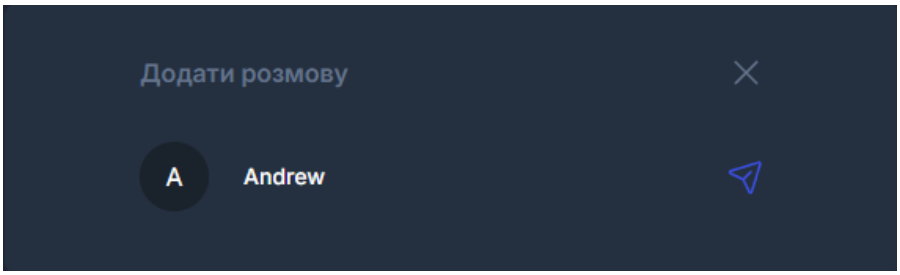


Рисунок Б7 – Вибір співрозмовника

Кожен чат має назву та, за наявності, аватар. Інтерфейс спроєктовано так, щоб користувач міг легко знайти потрібний чат, переглянути попередні повідомлення або розпочати нову розмову. Дизайн головної сторінки адаптований під різні типи пристроїв, що забезпечує комфортне користування незалежно від роздільної здатності екрана.

Сторінка налаштувань профілю дає змогу переглядати та змінювати особисті дані користувача, а також оновлювати пароль (див. рис. Б8).

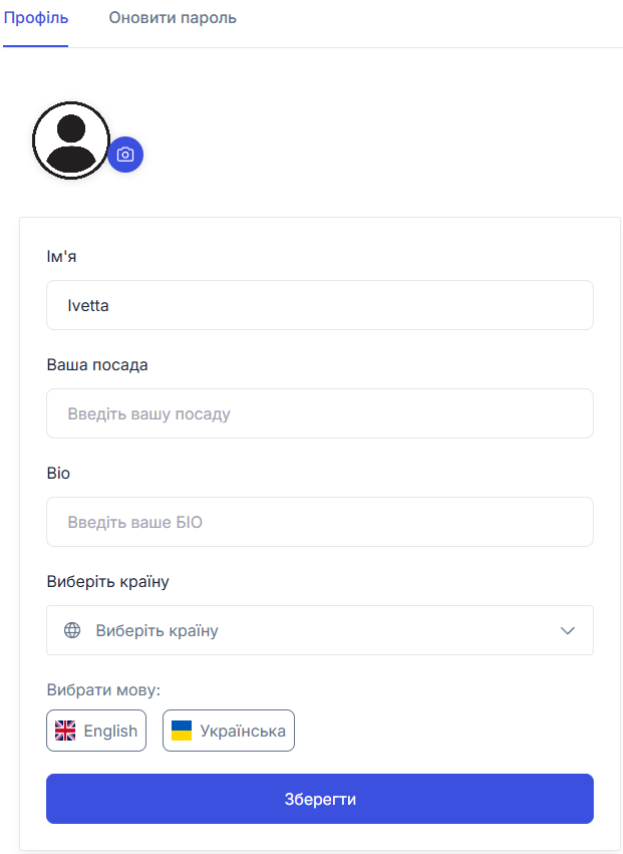


Рисунок Б8 – Сторінка налаштувань профілю користувача

На цій сторінці також передбачена функція зміни пароля (див. рис. Б9). Щоб оновити пароль, користувач має ввести свій поточний пароль і зазначити новий. Після підтвердження система оновлює дані, і користувач отримує повідомлення про успішну зміну пароля.

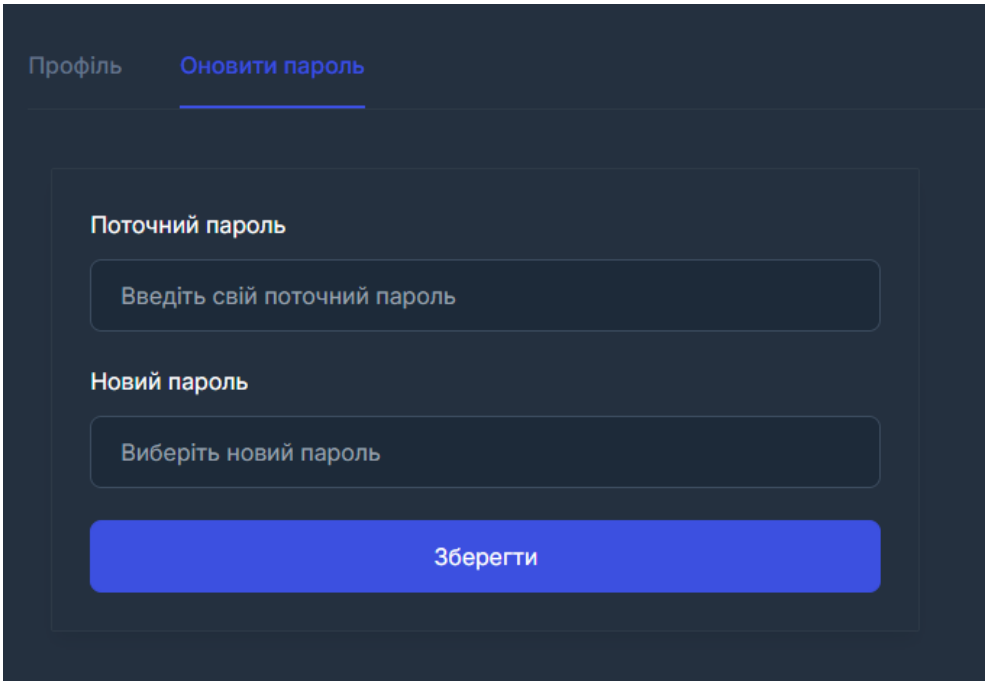


Рисунок Б9 – Сторінка зміни паролю

ДОДАТОК В
Код до програмної реалізації

```
const app = require("./app");
const dotenv = require("dotenv");

const mongoose = require("mongoose");
const socketServer = require("./socketServer");

dotenv.config({ path: "./config.env" });

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

const http = require("http");
const server = http.createServer(app);
socketServer.registerSocketServer(server);

// console.log(process.env.MONGO_URI);

mongoose
  .connect(process.env.MONGO_URI)
  .then(() => {
    console.log("MongoDB connection successful!");
    server.listen(PORT, () => {
      console.log(`Server is listening on ${PORT}`);
    });
  })
  .catch((err) => {
    console.log("database connection failed. Server not started");
    console.log(err);
  });
```

Рисунок В.1 – Підключення до Mongo.DB

```
const nodemailer = require("nodemailer");
const OTPTemplate = require("./../Template/OTP");
const dotenv = require("dotenv");

dotenv.config({ path: "./config.env" });

// const NODEMAILER_USER = process.env.NODEMAILER_USER;
// const NODEMAILER_APP_PASSWORD = process.env.NODEMAILER_APP_PASSWORD;

let transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: process.env.GMAIL_USER,
    pass: process.env.GMAIL_APP_PASSWORD,
  },
});

const Mailer = async ({ name, otp, email }) => {
  let mailOptions = {
    to: email, // Кому відправляється лист
    subject: "Verify your Chati Account",
    html: OTPTemplate({ name, otp }), // HTML-шаблон для листа
  };

  try {
    let info = await transporter.sendMail(mailOptions);
    console.log("Email sent: %s", info.messageId);
  } catch (error) {
    console.error("Error sending email: ", error);
    throw new Error("Error sending mail");
  }
};

module.exports = Mailer;
```

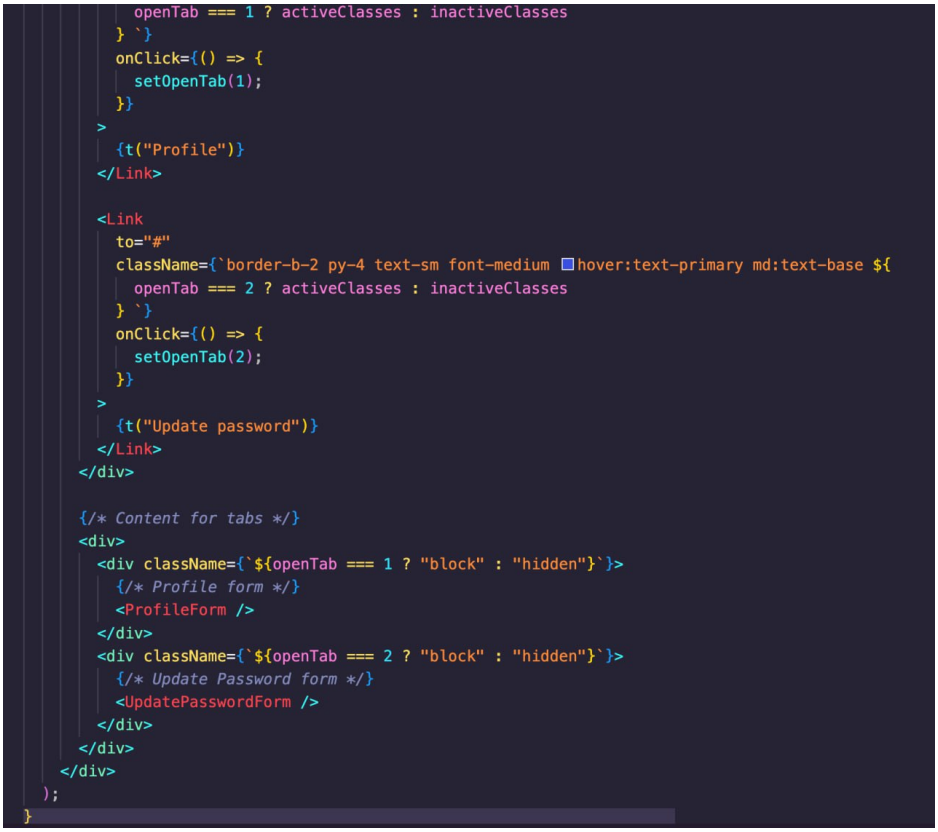
Рисунок В.2 – Код для відправки листа на пошту

```jsx
export default function App() {
  useEffect(() => {
    const colorMode = JSON.parse(window.localStorage.getItem("color-theme"));

    const className = "dark";

    const bodyClass = window.document.body.classList;

    colorMode === "dark"
      ? bodyClass.add(className)
      : bodyClass.remove(className);
  }, []);

  return (
    <Routes>
      {/* Redirect / to /auth/login */}

      <Route path="/" element={<Navigate to="/auth/login" />} />

      {/* <Route index={true} element={<Messages />} /> */}
      <Route path="/auth/login" element={<Login />} />
      <Route path="/auth/signup" element={<Signup />} />
      <Route path="/auth/verify" element={<Verification />} />

      <Route path="/dashboard" element={<Layout />}>
        <Route index element={<Messages />} />

        <Route path="profile" element={<ProfilePage />} />
      </Route>
    </Routes>
  );
}
```

Рисунок В.3 – Головний файл App.jsx

```jsx
io.use((socket, next) => {
  authSocket(socket, next);
});

io.on("connection", (socket) => {
  console.log("user connected");
  console.log(socket.id);

  // newConnectionHandler
  newConnectionHandler(socket, io);

  socket.on("disconnect", () => {
    disconnectHandler(socket);
  });

  socket.on("new-message", (data) => {
    newMessageHandler(socket, data, io);
  });

  socket.on("direct-chat-history", (data) => {
    chatHistoryHandler(socket, data);
  });

  socket.on("start-typing", (data) => {
    startTypingHandler(socket, data, io);
  });

  socket.on("stop-typing", (data) => {
    stopTypingHandler(socket, data, io);
  });
});

setInterval(() => {
  // emit online user
}, [1000 * 8]);
};
```

Рисунок В.4 – Код підключення сокетів



```
            openTab === 1 ? activeClasses : inactiveClasses
          } `}
          onClick={() => {
            setOpenTab(1);
          }}
        >
          {t("Profile")}
        </Link>

        <Link
          to="#"
          className={`border-b-2 py-4 text-sm font-medium hover:text-primary md:text-base ${
            openTab === 2 ? activeClasses : inactiveClasses
          } `}
          onClick={() => {
            setOpenTab(2);
          }}
        >
          {t("Update password")}
        </Link>
      </div>

      {/* Content for tabs */}
      <div>
        <div className={`${openTab === 1 ? "block" : "hidden"}`}>
          {/* Profile form */}
          <ProfileForm />
        </div>
        <div className={`${openTab === 2 ? "block" : "hidden"}`}>
          {/* Update Password form */}
          <UpdatePasswordForm />
        </div>
      </div>
    </div>
  );
}
```

Рисунок В.5 – Файл “Profile.jsx” для редагування профілю

```
const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Name is required"],
      trim: true,
    },
    // jobTitle
    jobTitle: {
      type: String,
      trim: true,
    },
    // bio
    bio: {
      type: String,
      trim: true,
    },
    // country
    country: {
      type: String,
    },
    // Avatar
    avatar: {
      type: String,
    },
    email: {
      type: String,
      required: [true, "Name is required"],
      validate: {
        validator: function (email) {
          return validator.isEmail(email);
        },
        message: (props) => `Email (${props.value}) is invalid!`,
      },
```

Рисунок В.6 – Схема для правильного заповнення даних користувачем

АНОТАЦІЯ

Гончарук А. В. – Розробка онлайн-чату засобами Node.js – Рукопис. Кваліфікаційна робота за спеціальністю 122 Комп’ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025 р.

Робота присвячена розробці та впровадженню чату для вебсайту, з використанням сучасних технологій JavaScript, Node.js, MongoDB та інтеграції з платформами як Supabase. Мета дослідження – створення високоефективного, безпечного та масштабованого чат–сервісу для забезпечення зручної комунікації між користувачами. Основні результати роботи демонструють, що використання ефективної структури даних, інтеграція з реальним часом та налаштування гнучкого інтерфейсу дозволяють досягти високої швидкості роботи, стабільності та зручності для кінцевого користувача.

Ключові слова: чат, вебсайт, JavaScript, Node.js, MongoDB, Supabase, реальний час, безпека, масштабованість, інтеграція, ефективність, користувацький інтерфейс.