«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»
Навчально-науковий інститут денної освіти

Форма навчання денна

Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри

_____ Олена ОЛЬХОВСЬКА

«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА ВЕБ-ДОДАТКУ СИСТЕМИ МИТТЄВОГО ОБМІНУ ПОВІДОМЛЕННЯМИ Simplify

зі спеціальності 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

ступеня бакалавра

Виконавець роботи Копитов Сергій Сергійович

_____«____»_____ 202 р.

Науковий керівник к. ф.-м. н., доцент, Чілікіна Тетяна Василівна

_____«____»_____ 202 р.

Рецензент _____

ПОЛТАВА 2024

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Олена ОЛЬХОВСЬКА

« ___ » _____ 202_ р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка веб-додатку системи миттєвого обміну повідомленнями Simplify»

зі спеціальності 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

ступеня бакалавр

Прізвище, ім'я, по батькові Копитов Сергій Сергійович

Затверджена наказом ректора № ____-Н від «___» _____ 202_ р.

Термін подання студентом роботи «__» _____ 202_ р.

Зміст пояснювальної записки

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ
ВСТУП
РОЗДІЛ 1: ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2: ІНФОРМАЦІЙНА ЧАСТИНА
2.1. Класифікація систем миттєвих повідомлень та їх функціональність.
2.2. Основні етапи створення систем миттєвого обміну повідомленнями.
2.3. Огляд інструментів і технологій, які будуть використовуватися під час розробки.
2.3.1 Переваги використання PHP.
РОЗДІЛ 3: ТЕОРЕТИЧНА ЧАСТИНА
3.1. Проектування структури бази даних для системи повідомлень.
3.2. Опис загальної архітектури системи миттєвих повідомлень Simplify, включаючи фізичну та логічну структури.
3.3. Алгоритм роботи системи
3.3.1 Опис послідовності дій користувачів при використанні системи.
3.3.2 Розгляд алгоритмів обробки повідомлень, відправлення та отримання.
3.3.3 Опис алгоритмів забезпечення безпеки та аутентифікації користувачів.
РОЗДІЛ 4: ПРАКТИЧНА ЧАСТИНА
4.1. Опис процесу розробки веб-додатку системи миттєвих повідомлень Simplify.
4.2. Тестування додатку.
ВИСНОВКИ
СПИСОК ЛІТЕРАТУРИ
ДОДАТКИ

Перелік графічного матеріалу

3 блок-схеми, інші необхідні ілюстрації.

Консультанти розділів кваліфікаційної роботи

Розділ	ПІБ, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постанова задачі	Чілікіна Т.В.		
Інформаційний огляд	Чілікіна Т.В.		
Теоретична частина	Чілікіна Т.В.		
Практична реалізація	Чілікіна Т.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	**Термін, неділя семестру**	**Фактичне виконання**
1. Вступ	2-й	2-й
2. Вивчення методичних рекомендацій і стандартів та звіт керівнику	3-й	3-й
3. Постановка завдання	4-й	4-й
4. Інформаційний огляд джерел бібліотек та Інтернету	5-й	5-й
5. Теоретична частина	6-й	6-й
6. Практична частина	6-й	6-й
7. Закінчення оформлення	6-10-й	6-10-й
8. Доповідь студента на кафедрі	7-10-й	7-10-й
9. Доопрацювання (за необхідності), рецен-зування	7-10-й	7-10-й

Дата видачі завдання «__» _____ 202_ р.

Здобувач вищої освіти Копитов Сергій Сергійович

Науковий керівник к. ф.-м. н., Чілікіна Т.В.

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

(балів, оцінка за національною шкалою, оцінка за ЄКТС)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.

Секретар ЕК _____ _____

(підпис) (ініціал та прізвище)

Затверджую
Зав. Кафедрою _____
к.ф-м.н. Олена Ольховська
«____» ___ _____ 202__ р.

Погоджено
Науковий керівник _____
к.ф-м.н. Тетяна ЧІЛІКІНА
«____» ___ _____ 202__ р.

План

кваліфікаційної роботи на тему «Розробка веб-додатку системи миттєвого обміну повідомленнями “Simplify”.»

спеціальності 122 Комп’ютерні науки

освітня програма 122 Комп’ютерні науки

ступеня бакалавр

Прізвище, ім’я, по батькові: Копитов Сергій Сергійович

Здобувач вищої освіти _____ _____

(Підпис)

«____» _____ 202__ р.

РЕФЕРАТ

Записка: 70 стор., 3 блок-схеми, 4 таблиці, 46 рис., джерел - 19.

Об’єкт розробки – веб-додаток системи миттєвого обміну повідомленнями "Simplify".

Мета роботи – розробка веб-додатку системи миттєвого обміну повідомленнями "Simplify" з метою покращення засобів комунікації у сучасному світі.

Методи реалізації – програмна реалізація веб-додатку була виконана з використанням HTML для структури сторінок, CSS для оформлення, JavaScript для взаємодії з користувачем, PHP для обробки серверної логіки, AJAX для асинхронного обміну даними з сервером, XAMPP для створення та запуску локального веб-сервера, і SQL для роботи з базою даних. Розробка та тестування додатку проводилася в середовищі XAMPP.

У результаті тестування виявлено, що система миттєвого обміну повідомленнями працює справно, без помилок та відповідає поставленій задачі. Важливим досягненням цього проекту є вирішення актуальної проблеми інформаційного перенавантаження в сфері миттєвого обміну повідомленнями. Шляхом створення "Simplify", системи, яка не лише ефективно функціонує і відповідає вимогам, але й спрощує та прискорює процес комунікації, проект вносить істотний внесок у поліпшення засобів взаємодії в інформаційному середовищі.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Умовні позначення, символи,скорочення, терміни	Пояснення умовних позначень, символів, скорочень
HTML	HyperText Markup Language — мова розмітки гіпертексту
CSS	Cascading Style Sheets - Каскадні таблиці стилів
PHP	Скриптова мова програмування, для виконання задач на стороні сервера.
XAMPP	XAMPP — Платформа для розробки та тестування веб-додатків, яка включає в себе сервер Apache, базу даних MySQL, редактор PHP і інші інструменти.
JavaScript	JavaScript (JS) — динамічна, об’єктно-орієнтовна прототипна мова програмування.
Ajax	AJAX (Asynchronous JavaScript And XML) — підхід до побудови користувацьких інтерфейсів вебзастосунків, за яких вебсторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані.
SQL	Structured query language — мова структурованих запитів до бази даних.

Вступ

У сучасному світі йде стрімкий розвиток сучаних технологій, у цьому контексті важливою є роль інструментів, які сприяють швидкому та ефективному обміну інформацією, однією з найпоширеніших форм цієї взаємодії є системи миттєвих повідомлень.

Головне завдання створення системи миттєвих повідомлень виникає з реальних потреб та вимог сучасного суспільства, з урахуванням швидкого темпу життя та зростаючої потреби в оперативній комунікації з'являється необхідність у створенні веб-додатку системи миттєвих повідомлень під назвою "Simplify", основною метою проекту є створення зручного та надійного інструменту для обміну повідомленнями, який би задовольняв потреби користувачів у швидкій та безпечній комунікації.

Системи обміну миттєвих повідомлень поділяються на декілька типів, включаючи особисті чати, групові чати, комунікацію через зображення та відеозв'язок, функціонал системи "Simplify" включає можливості обміну текстовими повідомленнями та зберігання історії повідомлень, перед розробкою системи потрібно вивчити потреби користувачів, дослідження показало, що основними вимогами є швидкість доставки повідомлень, зручний інтерфейс користувача, підтримка різноманітних мультимедійних форматів, конфіденційність та безпека інформації.

При розробці веб-додатку "Simplify" будуть використані сучасні технології, включаючи мови програмування та бази даних. Основна увага буде приділена забезпеченню надійності, безпеки та ефективності роботи системи.

Метою кваліфікаційної роботи є створення зручного та інтуїтивно зрозумілого інструменту для обміну повідомленнями між людьми, в системі "Simplify" користувачі матимуть можливість спілкуватися в

режимі реального часу і обмінюватися текстовими повідомленнями, це дозволить користувачам мати контакт з іншими людьми за допомогою веб-додатку

Предметом розробки є створення та реалізація веб-додатку "Simplify", що міститиме в собі функціонал системи миттєвих повідомлень. У фокусі розробки — забезпечення користувачам ефективного та безпечного інструменту для швидкого обміну текстовими повідомленнями в режимі реального часу.

Об’єктом розробки даної роботи є веб-додаток системи миттєвого обміну повідомленнями "Simplify". Цей веб-додаток є ключовим елементом, спрямованим на вдосконалення засобів комунікації в інформаційному просторі. Його розробка та впровадження мають на меті вирішення актуальних завдань забезпечення швидкого та ефективного обміну повідомленнями у сучасному світі.

Для високоякісної реалізації даного проекту ми зробимо детальний огляд, розкриваючи інформацію та основні завдання відповідно до розділів, визначених у данній кваліфікаційній роботі, розробка системи миттєвих повідомлень "Simplify" є актуальним завданням у сучасному інформаційному суспільстві, надзвичайно важливою є можливість спілкування та обміну інформацією у режимі реального часу, системи миттєвих повідомлень відкривають безмежні можливості для такого обміну, спрощуючи комунікацію та роблячи її більш ефективною.

Під час розробки "Simplify" ми будемо використовуватимемо передові технології та програмні рішення, які дозволять створити надійну та ефективну систему, важливими аспектами будуть безпека, конфіденційність даних та зручний інтерфейс користувача.

Функціональні можливості "Simplify" також розробляються з урахуванням зростаючих потреб користувачів у різноманітних форматах обміну повідомленнями.

Загалом, проектування системи миттєвих повідомлень "Simplify" відображає актуальність та важливість комунікаційних інновацій у сучасному світі, а також нашу відданість створенню інструменту, який полегшує спілкування та забезпечує безпеку даних у цифровій ері.

Подальший зміст роботи буде представлений в таких розділах:

- Вступ
- Постановка задачі
- Інформаційна частина
- Теоретична частина
- Практична частина
- Висновок

Кожен з цих розділів міститиме важливу інформацію та огляд, спрямований на досягнення поставлених цілей та завдань.

Результатом виконанням кваліфікаційної роботи є створена та готова до роботи система обміну миттєвих повідомлень веб-додаток «Simplify».

1. Постановка задачі

Основною задачею роботи було створити ефективний і надійний інструмент для обміну повідомленнями, задовольняючи потреби користувачів у швидкій та безпечній комунікації.

Основні завдання, які виникли при реалізації проекта наступні **:**

1. **Аналіз і вивчення існуючих систем миттєвих повідомлень:** Вивчити функціональність та особливості роботи інших систем миттєвих повідомлень для зрозуміння їх переваг та недоліків.
2. **Проектування структури бази даних:** Розробити структуру бази даних для зберігання повідомлень, профілів користувачів та інших необхідних даних.
3. **Розробка інтерфейсу користувача:** Створити інтуїтивний та зручний інтерфейс для взаємодії користувачів з додатком, включаючи можливість обміну текстовими та фото повідомленнями.
4. **Використання сучасних технологій:** Використовувати сучасні технології розробки, такі як мови програмування та інструменти, для створення надійної та ефективної системи.
5. **Тестування:** Провести тестування додатку "Simplify" для перевірки його відповідності вимогам та очікуванням користувачів.

Завдання та цілі цього проекту будуть докладно розглянуті та реалізовані в подальших розділах даного проекту, включаючи аналіз інструментів і технологій, огляд структури та функціональності системи, та розробку деталей додатку "Simplify."

2. ІНФОРМАЦІЙНА ЧАСТИНА

2.1. Класифікація систем миттєвих повідомлень та їх функціональність.

Системи миттєвих повідомлень (Instant Messaging, IM) - це програмне забезпечення, яке дозволяє користувачам обмінюватися текстовими, голосовими та відео повідомленнями в режимі реального часу. Системи миттєвих повідомлень можна класифікувати за різними критеріями, такими як тип, функціональність, платформа та протокол.

За типом системи миттєвих повідомлень можна поділити на:

- **Персональні:** системи миттєвих повідомлень, які дозволяють користувачам обмінюватися повідомленнями один з одним. Це найпоширеніший тип систем миттєвих повідомлень.
- **Групові:** системи миттєвих повідомлень, які дозволяють користувачам обмінюватися повідомленнями з групою людей. Групи можуть бути створені користувачами або заздалегідь визначені.
- **Мережеві:** системи миттєвих повідомлень, які дозволяють користувачам обмінюватися повідомленнями з користувачами інших систем миттєвих повідомлень. Цей тип систем миттєвих повідомлень використовує стандартні протоколи, такі як SIP.

За функціональністю системи миттєвих повідомлень можна поділити на:

- **Текстові:** системи миттєвих повідомлень, які дозволяють користувачам обмінюватися лише текстовими повідомленнями. Це найпростіші системи миттєвих повідомлень.
- **Мультимедійні:** системи миттєвих повідомлень, які дозволяють користувачам обмінюватися текстовими, голосовими та відео повідомленнями. Це більш досконалі системи миттєвих повідомлень, які пропонують більш широкий спектр можливостей для спілкування.
- **Розширені:** системи миттєвих повідомлень, які надають додаткові функції, такі як голосовий зв'язок, відеозв'язок, файлообмін, ігровий сервіс тощо. Це найскладніші системи миттєвих повідомлень, які

пропонують широкий спектр можливостей для спілкування та співпраці.

За платформою системи миттєвих повідомлень можна поділити на:

- **Веб-базовані:** системи миттєвих повідомлень, які працюють в браузері. Це найзручніший тип систем миттєвих повідомлень, оскільки вони не вимагають установки програмного забезпечення.
- **Настільні:** системи миттєвих повідомлень, які встановлюються на комп'ютер. Цей тип систем миттєвих повідомлень пропонує більш широкий спектр можливостей і функцій, ніж веб-базовані системи миттєвих повідомлень.
- **Мобільні:** системи миттєвих повідомлень, які працюють на мобільних пристроях. Цей тип систем миттєвих повідомлень дозволяє користувачам спілкуватися з будь-якого місця.

За протоколом системи миттєвих повідомлень можна поділити на:

- **TCP/IP:** системи миттєвих повідомлень, які використовують протокол TCP/IP для передачі повідомлень. Цей протокол є стандартним протоколом для передачі даних в Інтернеті.
- **Peer-to-peer:** системи миттєвих повідомлень, які використовують протокол Peer-to-Peer для передачі повідомлень. Цей протокол дозволяє користувачам обмінюватися повідомленнями безпосередньо один з одним, без використання центрального сервера.

Системи миттєвих повідомлень можуть включати в себе широкий спектр функцій і можливостей, які дозволяють користувачам спілкуватися в режимі реального часу.

Основні функції систем миттєвих повідомлень включають:

- **Обмін текстовими повідомленнями:** це основна функція систем миттєвих повідомлень. Користувачі можуть обмінюватися текстовими повідомленнями в режимі реального часу.
- **Обмін голосовими повідомленнями:** ця функція дозволяє користувачам обмінюватися голосовими повідомленнями в режимі реального часу.
- **Обмін відео повідомленнями:** ця функція дозволяє користувачам обмінюватися відео повідомленнями в режимі реального часу.
- **Файлообмін:** ця функція дозволяє користувачам обмінюватися файлами в режимі реального часу.

- **Ігровий сервіс:** деякі системи миттєвих повідомлень пропонують ігровий сервіс, який дозволяє користувачам грати в ігри разом.
- **Аватарки:** аватарки - це зображення, яке користувачі використовують для представлення себе в системах миттєвих повідомлень.
- **Смайлики:** смайлики - це графічні зображення, які використовуються для вираження емоцій в повідомленнях.
- **Пошук користувачів:** ця функція дозволяє користувачам знаходити інших користувачів систем миттєвих повідомлень.
- **Групи:** групи - це спосіб організації користувачів систем миттєвих повідомлень.
- **Каталоги:** каталоги - це спосіб пошуку користувачів систем миттєвих повідомлень за певними критеріями.

2.2. Основні етапи створення систем миттєвого обміну повідомленнями

Створення систем миттєвого обміну повідомленнями (Instant Messaging, IM) є складним процесом, який включає в себе кілька етапів.

Перший етап - аналіз потреб та вимог користувачів. На цьому етапі необхідно вивчити потреби та вимоги користувачів, щоб розробити систему, яка буде задовольняти їхні потреби. Для цього можна провести опитування, інтерв'ю та аналіз даних про поведінку користувачів.

- **Аналіз потреб та вимог користувачів** - це найважливіший етап у створенні систем миттєвого обміну повідомленнями. Саме на цьому етапі визначається, які функції та можливості будуть включені в систему.

У процесі аналізу потреб та вимог користувачів можна використовувати різні методи:

- **Опитування** - це найпростіший і найпоширеніший метод. Опитування можна проводити за допомогою онлайн-форм, телефонних дзвінків або особистих інтерв'ю.
- **Інтерв'ю** - це більш детальний метод, який дозволяє краще зрозуміти потреби та вимоги користувачів. Інтерв'ю можна проводити з індивідуальними користувачами або з групами користувачів.
- **Аналіз даних про поведінку користувачів** - це метод, який дозволяє отримати інформацію про те, як користувачі

використовують існуючі системи миттєвого обміну повідомленнями. Цю інформацію можна отримати за допомогою аналізу даних, що зберігаються на серверах систем миттєвого обміну повідомленнями.

Другий етап - проектування системи. На цьому етапі необхідно розробити концепцію системи, написати структуру системи та розробити алгоритм роботи системи.

На цьому етапі необхідно розробити:

- **Концепцію системи**, яка визначає загальні цілі та завдання системи. Концепція системи повинна бути чіткою і лаконічною, щоб її можна було легко зрозуміти.
- **Структуру системи**, яка визначає фізичну та логічну структуру системи. Структура системи повинна бути ефективною та масштабованою.
- **Алгоритм роботи системи**, який визначає послідовність дій користувачів при використанні системи. Алгоритм роботи системи повинен бути простим і зрозумілим.

Третій етап - розробка системи. На цьому етапі необхідно реалізувати систему на основі специфікацій. Цей процес включає в себе написання коду, тестування коду та усунення помилок.

- **Розробка системи** - це найскладніший і найтриваліший етап у створенні систем миттєвого обміну повідомленнями. На цьому етапі необхідно залучити команду досвідчених розробників, які зможуть реалізувати всі функції та можливості системи.

Четвертий етап - розгортання системи. На цьому етапі необхідно встановити систему на серверах і зробити її доступною для користувачів. Цей процес включає в себе настройку системи, забезпечення її безпеки та підтримку її в актуальному стані.

- **Розгортання системи** - це важливий етап, який дозволяє зробити систему доступною для користувачів. На цьому етапі необхідно забезпечити надійну роботу системи та її безпеку.

П'ятий етап - просування системи. На цьому етапі необхідно проінформувати користувачів про систему і її можливості. Для цього можна використовувати різні канали комунікації, такі як веб-сайт, соціальні мережі та електронна пошта.

- **Просування системи** - це важливий етап, який дозволяє залучити користувачів до системи. На цьому етапі необхідно проінформувати користувачів про переваги системи та її можливості.

2.3. Огляд інструментів і технологій, які будуть використовуватися під час розробки

Під час розробки системи Simplify будуть використовуватися такі інструменти та технології:

Інструмент	Опис
HTML	Мова розмітки, яка використовується для створення структури веб-сторінок.
CSS	Мова стилів, яка використовується для форматування веб-сторінок.
JavaScript	Скриптова мова, яка використовується для додавання інтерактивності до веб-сторінок.
PHP	Скриптова мова, яка використовується для створення веб-додатків.
SQL	Мова запитів, яка використовується для доступу до даних у базах даних.
XAMPP	Платформа для розробки та тестування веб-додатків, яка включає в себе сервер Apache, базу даних MySQL, редактор PHP і інші інструменти.
Ajax	Технологія, що використовується в розробці веб-сайтів та веб-додатків для асинхронного взаємодії з веб-сервером без необхідності перезавантаження сторінки. AJAX дозволяє веб-додаткам відправляти та отримувати дані з сервера без переривання роботи користувача на сторінці.

Вибір інструментів і технологій

Вибір інструментів і технологій для розробки системи Simplify був проведений на основі таких критеріїв:

- **Доступність**. Інструменти і технології повинні бути доступними і простими у використанні.
- **Ефективність**. Інструменти і технології повинні бути ефективними і дозволяти розробити надійну і масштабовану систему.

- **Надійність**. Інструменти і технології повинні бути надійними і не допускати збоїв системи.

На основі цих критеріїв було обрано такі інструменти і технології:

- **HTML, CSS, JavaScript** - це стандартні технології для розробки веб-сайтів і додатків, які добре відомі і доступні для більшості розробників.
- **PHP** - це популярна і потужна мова програмування, яка добре підходить для розробки веб-додатків. PHP також є доступною і простою у використанні.
- **SQL** - це стандартна мова запитів, яка використовується для доступу до даних у базах даних. SQL також є доступною і простою у використанні.
- **XAMPP** - це популярна і універсальна платформа для розробки та тестування веб-додатків. XAMPP також є доступною і простою у використанні.

Вибір інструментів і технологій для розробки системи Simplify є важливим кроком, який визначає майбутнє системи. Вибрані інструменти і технології дозволять розробити надійну, масштабовану і ефективну систему.

2.3.1 Переваги використання PHP

Для реалізації таких проектів використовують мову PHP -, це популярна і потужна мова програмування, яка добре підходить для розробки веб-додатків. PHP має ряд переваг, які роблять його хорошим вибором для розробки системи Simplify:

- **Доступність.** PHP - це відкрита мова програмування, яка доступна для безкоштовного використання. Це означає, що будь-хто може використовувати PHP для розробки веб-додатків, не купуючи ліцензію.
- **Простота у використанні**. PHP має простий і інтуїтивно зрозумілий синтаксис, який робить його легкою для вивчення і використання. PHP використовує синтаксис, який схожий на синтаксис мов C і Java, тому його легко вивчити, якщо ви вже знаєте одну з цих мов.
- **Масштабованість**. PHP є масштабованою мовою програмування, яка може підтримувати великі обсяги трафіку. PHP може працювати на різних серверах і платформах, тому його можна легко масштабувати в міру зростання системи.

- **Безпека**. PHP має вбудовані функції безпеки, які допомагають захистити веб-додатки від атак. PHP включає в себе такі функції безпеки, як фільтрація вхідних даних, аутентифікація і авторизація.
- **Мобільність**. PHP є портативною мовою програмування, яка може використовуватися на різних платформах. PHP можна використовувати для розробки веб-додатків, які працюють на веб-браузері, на мобільних пристроях і на серверах.

Розглядаючи вибір мови програмування для кваліфікаційної роботи, PHP вибрано з урахуванням його багатосторонніх переваг. Популярність PHP в розробницькому середовищі забезпечує доступ до широкого спектру ресурсів та знань, що є важливим у вимірюванні успіху проекту. Простота синтаксису дозволяє швидко створювати функціональний код, забезпечуючи швидкість розробки. Підтримка об'єктно-орієнтованого програмування надає структурованість та легкість утримання системи. PHP легко інтегрується з різноманітними технологіями, що дозволяє створювати гнучкі рішення для системи обміну повідомленнями. Вибір PHP також обумовлений його активним розвитком та потужною спільнотою, що гарантує вдосконалення та вчасну підтримку. В цілому, PHP виступає як ідеальний інструмент для створення інноваційної та надійної системи обміну миттєвих повідомлень "Simplify", враховуючи його зручність, ефективність та готовність до майбутніх розширень.

PHP - це гарнийвибір для розробки системи Simplify. PHP має ряд переваг, які роблять його хорошим вибором для розробки масштабованої, надійної і безпечної системи.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Проектування структури бази даних для системи повідомлень

Система повідомлень - це інформаційна система, яка дозволяє користувачам обмінюватися повідомленнями між собою. Система складається з двох основних компонентів:

- База даних для зберігання даних про користувачів, повідомленнях і чатах.
- Веб-інтерфейс для взаємодії користувачів з системою.

В цьому розділі буде розглянуто проектування структури бази даних для системи повідомлень.

Список необхідних таблиць:

Таблиця users

Таблиця users містить інформацію про користувачів системи.

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор користувача
login	VARCHAR(255)	Логін користувача
email	VARCHAR(255)	Адреса електронної пошти користувача
password	VARCHAR(255)	Пароль користувача

Таблиця chats

Таблиця chats містить інформацію про чати.

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор чату
user1_id	INT	Ідентифікатор першого учасника чату
user2_id	INT	Ідентифікатор другого учасника чату
created_at	DATETIME	Дата і час створення чату

Таблиця messages

Таблиця messages містить інформацію про повідомлення.

Поле	Тип	Опис
id	INT	Унікальний ідентифікатор повідомлення
chat_id	INT	Ідентифікатор чату, в якому було відправлено повідомлення
sender_id	INT	Ідентифікатор відправника повідомлення
message	TEXT	Текст повідомлення

created_at	DATETIME	Дата і час відправлення повідомлення

Відносини між таблицями

Таблиці users, chats і messages пов'язані між собою наступними відносинами:

- Один користувач може бути учасником декількох чатів. Це реалізовано за допомогою зовнішнього ключу user1_id та user2_id в таблиці chats, який посилається на поле id в таблиці users.
- Два користувачі можуть бути учасниками одного чату. Це реалізовано за допомогою зовнішнього ключу user1_id та user2_id в таблиці chats, який посилається на поле id в таблиці users.
- Одне повідомлення може бути відправлено в один чат. Це реалізовано за допомогою зовнішнього ключу chat_id в таблиці messages, який посилається на поле id в таблиці chats.
- Повідомлення може бути відправлено одним користувачем. Це реалізовано за допомогою зовнішнього ключу sender_id в таблиці messages, який посилається на поле id в таблиці users.

Проектування структури бази даних для системи повідомлень відповідає сучасним вимогам до проектування баз даних. Структура бази даних є повною і точною, відповідає функціональним вимогам системи.

Використання зовнішніх ключів дозволяє забезпечити цілісність даних в базі даних.

Проектування відповідає принципам проектування баз даних, таким як:

- **Ізоляція даних:** дані в таблицях зберігаються таким чином, щоб їх можна було легко використовувати і обробляти.
- **Ефективність:** структура бази даних оптимізована для швидкого доступу до даних.
- **Здатність до масштабування:** структура бази даних може бути легко розширена для підтримки зростання кількості даних.

В цілому, проектування структури бази даних для системи повідомлень є якісним і відповідає вимогам системи.

База даних у вигляді блок-схеми (рис. 3.1):

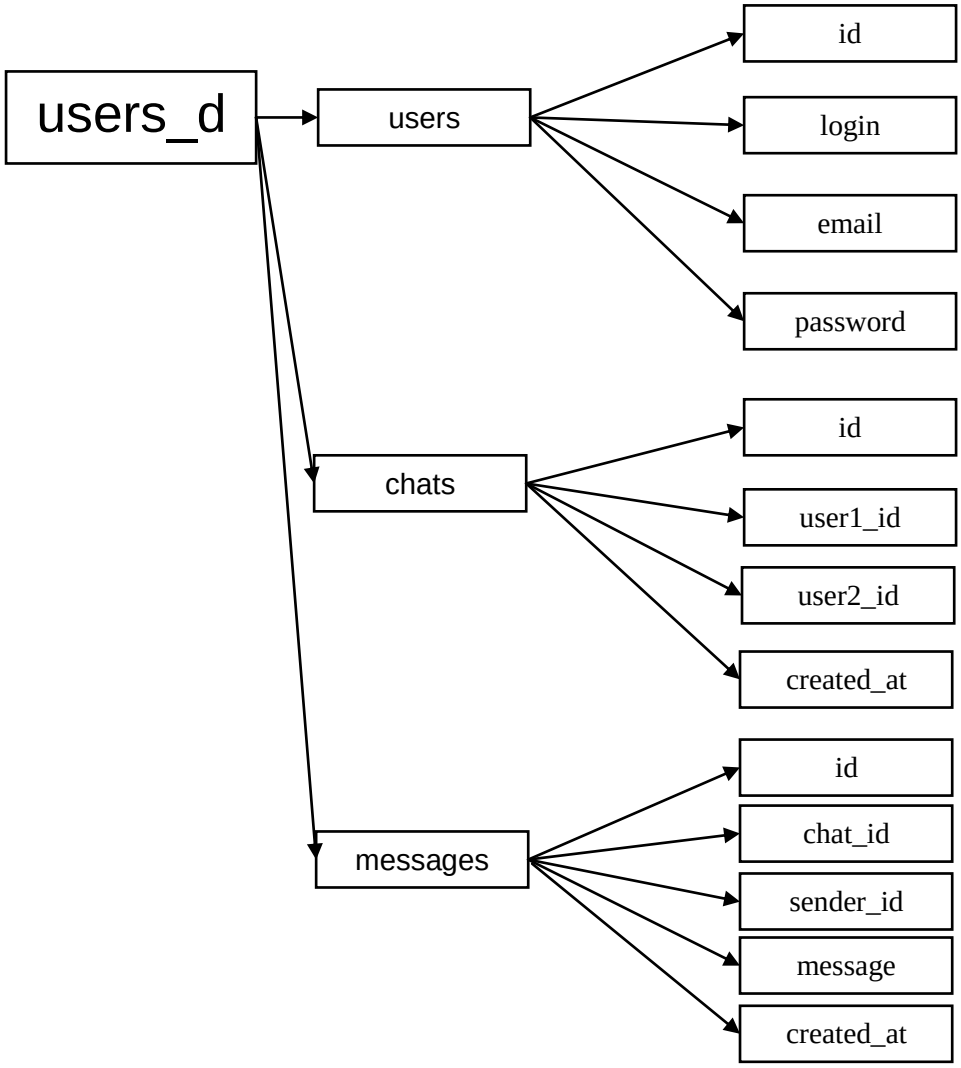

Рис. 3.1

3.2 Опис загальної архітектури системи миттєвих повідомлень Simplify, включаючи фізичну та логічну структури.

Система миттєвих повідомлень Simplify є складною системою, яка складається з кількох компонентів, що взаємодіють між собою. Ці компоненти можуть бути згруповані в три рівні: фізичний, логічний та прикладний.

Фізична структура системи Simplify базується на розподіленому (клієнт-серверному) підході. Вона складається з наступних компонентів:

1. **Клієнтська частина:** Клієнтська частина системи представлена веб-додатком, який доступний користувачам через веб-браузери. Клієнтська сторона відповідає за інтерфейс користувача, взаємодію з користувачем, та відправку та отримання повідомлень до та від серверної частини.

2. **Серверна частина:** Серверна частина системи Simplify централізовано керує всіма процесами. Сюди входять веб-сервер, сервер баз даних та сервери для обробки логіки додатку. Сервери забезпечують обробку запитів від клієнтів, зберігання та витягання даних з бази даних, автентифікацію користувачів та забезпечення безпеки.
3. **База даних:** Важливою складовою фізичної структури є база даних, в якій зберігаються всі дані, пов'язані з користувачами, повідомленнями, налаштуваннями та іншою інформацією, необхідною для роботи системи. База даних забезпечує ефективний доступ до даних та збереження їх у безпечному режимі.

Прикладний рівень відповідає за інтерфейс користувача та взаємодію з користувачами. Він складається з наступних компонентів:

- **Інтерфейс користувача:** Інтерфейс користувача повинен бути інтуїтивно зрозумілим і простим у використанні. Він повинен бути доступним для користувачів з обмеженими можливостями.

Блок-схема прикладного рівня (рис. 3.2):

Рис. 3.2

Запропонована архітектура системи миттєвих повідомлень Simplify забезпечує високу надійність, масштабованість та безпеку. Вона відповідає сучасним вимогам до систем

миттєвих повідомлень і може бути використана для створення ефективних та зручних у використанні систем миттєвих повідомлень.

Логічний рівень(на рис. 3.3) відповідає за логічну організацію даних та процесів. Він складається з наступних компонентів:

- **Система реєстрації:** Система реєстрації використовується для створення облікових записів користувачів. Система реєстрації повинна забезпечувати безпеку та простоту використання.
- **Система авторизації:** Система авторизації використовується для перевірки автентичності користувачів. Система авторизації повинна забезпечувати безпеку та надійність.
- **Система обробки повідомлень:** Система обробки повідомлень відповідає за обробку повідомлень, отриманих від користувачів. Система обробки повідомлень повинна забезпечувати масштабованість та безпеку.
- **Система чат-кімнат:** Система чат-кімнат дозволяє користувачам спілкуватися в чат-кімнатах. Система чат-кімнат повинна забезпечувати безпеку та зручність використання.

Блок-схема логічного рівня (рис. 3.3):

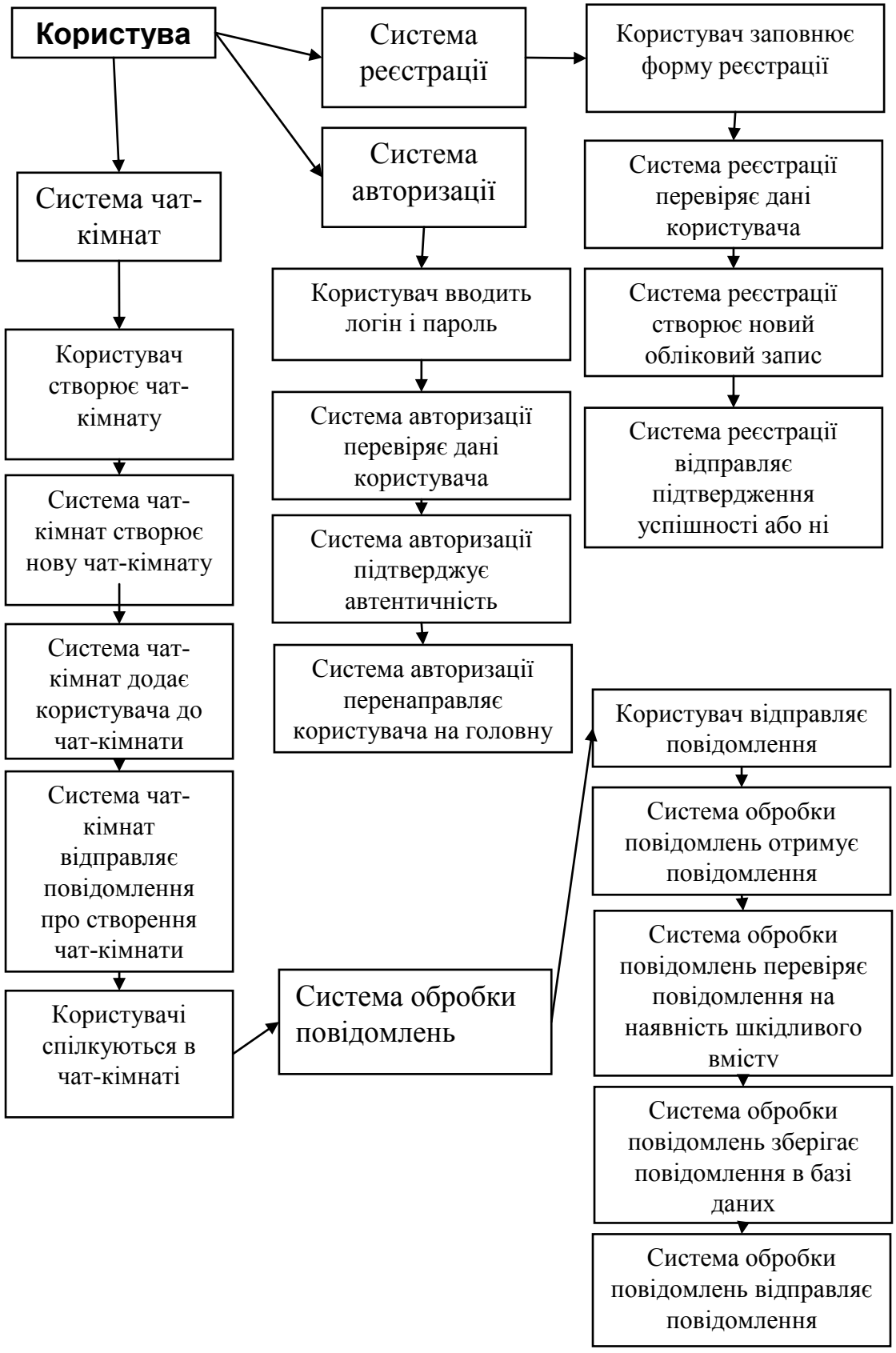

Рис. 3.3

3.3. Алгоритм роботи системи

У цьому розділі ми детально розглянемо послідовність дій, яку здійснюють користувачі під час використання додатку, описуємо алгоритми обробки повідомлень, їх відправлення та отримання, а також надаємо пояснення алгоритмів забезпечення безпеки та аутентифікації користувачів.

У даному розділі ми розкриємо внутрішні процеси, що відбуваються в системі "Simplify", починаючи від моменту реєстрації користувача та входу в систему. Ми розглянемо як система керує інформацією, що надходить від користувачів, обробляє запити на відправлення повідомлень, забезпечує надійну доставку повідомлень до адресата та забезпечує конфіденційність даних під час обміну.

Також, ми розкриємо аспекти безпеки системи, аналізуючи алгоритми шифрування, методи аутентифікації та заходи для запобігання несанкціонованому доступу до даних користувачів. У розділі буде надано докладний огляд та пояснення процесів, що забезпечують цілісність та безпеку інформації в системі "Simplify".

Даний розділ важливий для розуміння та дослідження принципів функціонування системи миттєвих повідомлень, і його зміст буде відзначати ключові аспекти роботи системи та її можливості для забезпечення високоякісного та безпечного обміну повідомленнями між користувачами.

3.3.1 Опис послідовності дій користувачів при використанні системи.

Алгоритм роботи системи Simplify передбачає наступну послідовність дій користувачів під час використання системи. Процес взаємодії користувачів із системою миттєвих повідомлень детально описується нижче:

1. **Авторизація в системі:**
    - Користувач запускає додаток Simplify на своєму пристрої.
    - Система відображає екран авторизації, де користувач повинен ввести свій логін та пароль, які були створені під час реєстрації.
2. **Вибір або створення чату:**
    - Після успішної авторизації користувач має можливість вибрати наявний чат або створити новий.
    - Для вибору чату, користувач переходить до списку доступних чатів та обирає потрібний.
3. **Обмін повідомленнями:**
    - Після вибору чату, користувач може вводити текстові повідомлення у спеціальне поле введення.
    - Після введення тексту користувач натискає кнопку "Відправити," і повідомлення відправляється в чат.
4. **Отримання та відправлення повідомлень:**
    - Повідомлення, надіслані іншими користувачами в цьому чаті, відображаються в реальному часі на екрані користувача.
    - Користувач може надсилати текстові повідомлення одержувачам, і ті можуть відповідати на них.
5. **Завершення сесії та вихід:**

- Після завершення спілкування, користувач може вийти з чату та повернутися до списку доступних чатів або вийти з системи Simplify.

Ця послідовність дій визначає основний процес взаємодії користувачів із системою миттєвих повідомлень Simplify. Вона забезпечує зручну та ефективну комунікацію між користувачами та дозволяє обмінюватися різноманітною інформацією в реальному часі.

3.3.2. Розгляд алгоритмів обробки повідомлень, відправлення та отримання

У даному розділі буде розглянуто ключові алгоритми, що використовуються в системі миттєвих повідомлень Simplify для обробки повідомлень, їх відправлення та отримання. Ці алгоритми визначають ефективну роботу системи та гарантують оперативний обмін інформацією між користувачами.

Алгоритм обробки повідомлень:

1. **Отримання повідомлення:** Коли користувач відправляє повідомлення, система отримує його та ідентифікує відправника та отримувача.
2. **Перевірка безпеки:** Перед збереженням повідомлення в базі даних, система перевіряє наявність шифрування та забезпечення конфіденційності для цього повідомлення.
3. **Збереження повідомлення:** Після перевірки повідомлення зберігається в базі даних системи.
4. **Сповіщення отримувача:** Система сповіщає отримувача про наявність нового повідомлення та передає йому вміст повідомлення.

Алгоритм відправлення повідомлень:

1. **Вибір отримувача:** Користувач обирає отримувача або групу отримувачів для відправлення повідомлення.
2. **Створення повідомлення:** Користувач створює текстове або мультимедійне повідомлення та додає до нього необхідні елементи, такі як зображення чи відео.
3. **Перевірка безпеки:** Перед відправленням система перевіряє, чи включено шифрування та конфіденційність для цього повідомлення.
4. **Відправлення повідомлення:** Повідомлення відправляється отримувачу або групі отримувачів та зберігається в базі даних.

Алгоритми обробки та відправлення повідомлень в системі Simplify дозволяють користувачам надсилати та отримувати повідомлення швидко та ефективно, забезпечуючи при цьому високий рівень безпеки та конфіденційності інформації. Ці алгоритми є важливою частиною функціональності системи та гарантують зручний обмін повідомленнями для користувачів.

3.3.3. Пояснення алгоритмів забезпечення безпеки та аутентифікації користувачів

У цьому розділі будуть розглянуті ключові алгоритми та методи, які використовуються в системі миттєвих повідомлень Simplify для забезпечення безпеки та аутентифікації

користувачів. Забезпечення конфіденційності та безпеки даних є однією з найважливіших складових будь-якої системи з обміну повідомленнями.

Алгоритм забезпечення безпеки повідомлень:

1. **Шифрування даних:** Перед відправленням повідомлення на сервер системи, вміст повідомлення шифрується з використанням сильних криптографічних алгоритмів. Це забезпечує конфіденційність даних під час їх передачі.
2. **Контроль доступу:** Система встановлює правила контролю доступу до повідомлень. Це означає, що користувачі можуть переглядати лише ті повідомлення, до яких у них є доступ.
3. **Захист від атак:** Система використовує заходи безпеки для захисту від різних видів атак, включаючи SQL-ін'єкції, переповнення буфера, атаки на переповнення та інші.

Алгоритм аутентифікації користувачів:

1. **Вхідні дані:** Користувач вводить свій логін та пароль для входу в систему.
2. **Хешування паролю:** Пароль користувача хешується з використанням односторонньої хеш-функції. Цей хеш зберігається в базі даних системи.
3. **Перевірка логіна та паролю:** Введені користувачем дані порівнюються зі збереженими хешами в базі даних. Якщо дані співпадають, користувач отримує доступ до системи.

Алгоритми забезпечення безпеки та аутентифікації користувачів гарантують, що дані користувачів залишаються конфіденційними та безпечними, а доступ до системи обмежується лише для авторизованих користувачів. Ці алгоритми є важливою складовою системи Simplify, спрямованою на забезпечення безпеки та довіри користувачів.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис процесу розробки веб-додатку системи миттєвих повідомлень Simplify.

Для початку, для процесу розробки, треба завантажити XAMPP, запустити його та увімкнути Apache та MySQL за допомогою кнопок Start.

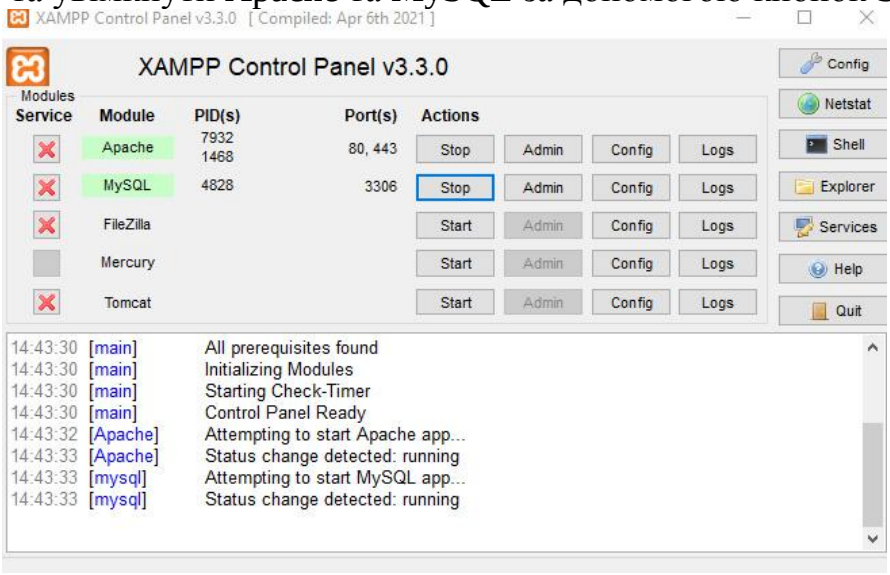

Рис. 4. 1

Після того як це зроблено, треба перейти по посиланню: http://localhost/phpmyadmin/index.php?route=/server/sql&lang=en

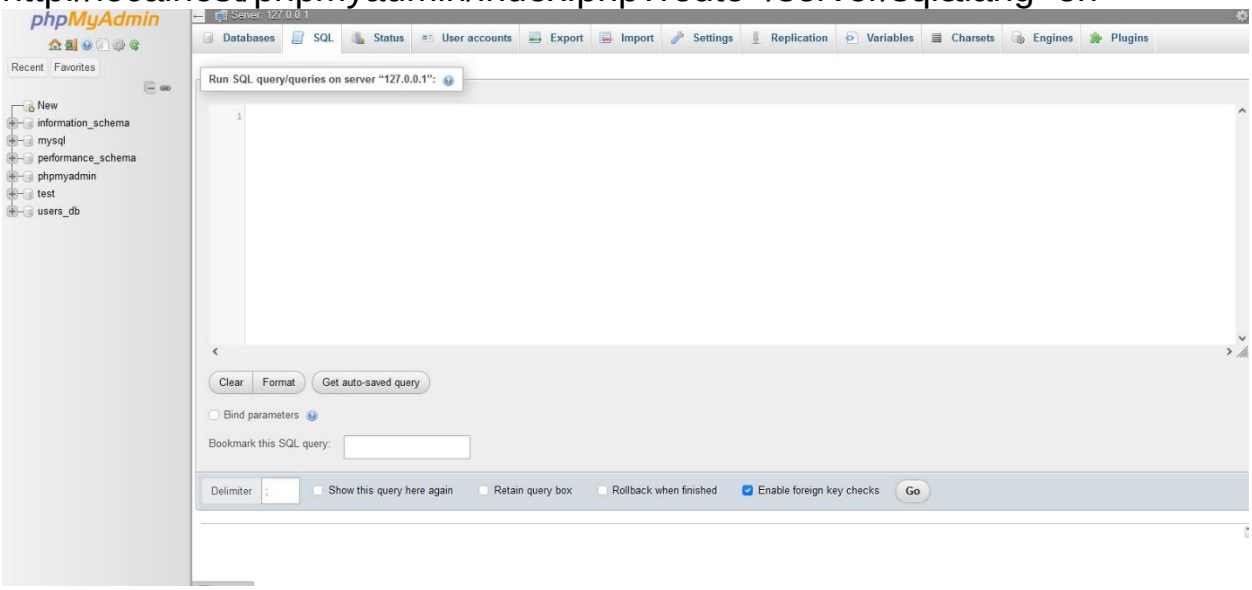

Рис. 4. 2

Це адмін-панель серверу, на який буде завантажено код, та в якому буде міститися база даних SQL.

Увесь код треба зберігати у папці: C:\xampp\htdocs\simplifychat

Сам веб-додаток буде доступний по посиланню:
http://localhost/simplifychat/

Перед тим, як починати процес розробки авторизації, реєстрації та іншого, треба створити базу даних:

Можна створити базу даних за допомогою двох способів:

1. За допомогою інтерфейсу адмін-панелі.
2. За допомогою вкладки SQL, де потрібно вручну вводити код.

Розглянемо другий спосіб, для того щоб більш детальніше все описати. Перейдемо у вкладку SQL та будемо все вводити у поле вводу, та після кожної команди треба нажимати кнопку Вперед.

Створимо базу даних:

```
CREATE DATABASE users_db
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_general_ci;
```

- **CREATE DATABASE** — це ключове слово, яке використовується для створення бази даних.
- **users_db** — це ім'я бази даних, яку ми хочемо створити.
- **CHARACTER SET utf8mb4** — це кодування символів, яка буде використовуватися для бази даних.
- **COLLATE utf8mb4_general_ci** — це набір символів, який буде використовуватися для бази даних.

Тепер нам потрібно створити 3 таблиці для зберігання даних користувачів, чатів та повідомлень:

```
USE users_db;
CREATE TABLE users (
  id INT NOT NULL AUTO_INCREMENT,
  login VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE chats (
  id INT NOT NULL AUTO_INCREMENT,
  user1_id INT NOT NULL,
  user2_id INT NOT NULL,
  created_at DATETIME NOT NULL,
  PRIMARY KEY (id)
);

CREATE TABLE messages (
  id INT NOT NULL AUTO_INCREMENT,
  chat_id INT NOT NULL,
```

```
    sender_id INT NOT NULL,
    message VARCHAR(255) NOT NULL,
    created_at DATETIME NOT NULL,
    PRIMARY KEY (id)
);
```

- **CREATE TABLE** — це ключове слово, яке використовується для створення таблиці.
- **users**, **chats** і **messages** — це імена таблиць, які ми хочемо створити.
- **INT** — це тип даних, який буде використовуватися для зберігання ідентифікаторів.
- **VARCHAR(255)** — це тип даних, який буде використовуватися для зберігання рядків довжиною до 255 символів.
- **DATETIME** — це тип даних, який буде використовуватися для зберігання дат і часу.
- **NOT NULL** — це обмеження, яке вказує, що поле не може бути порожнім.
- **AUTO_INCREMENT** — це обмеження, яке вказує, що ідентифікатор буде автоматично збільшуватися на 1 при кожному створенні нового запису.
- **PRIMARY KEY** — це обмеження, яке вказує, що поле є первинним ключем і може використовуватися для однозначної ідентифікації запису.

Також створимо зв’язки між таблицями:

```
USE users_db;
ALTER TABLE chats
    ADD FOREIGN KEY (user1_id) REFERENCES users (id);

ALTER TABLE chats
    ADD FOREIGN KEY (user2_id) REFERENCES users (id);

ALTER TABLE messages
    ADD FOREIGN KEY (chat_id) REFERENCES chats (id);

ALTER TABLE messages
    ADD FOREIGN KEY (sender_id) REFERENCES users (id);
```

Тепер перейдемо до створення самої системи миттєвих повідомлень.

Для початку треба створити config.php, який буде містити дані для того щоб використовувати їх для підключення до бази даних у php скриптах.

```php
<?php
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', '');
define('DB_NAME', 'users_db');
?>

```

Рис. 4. 3

У кожному php скрипті буде міститись строка:

include('config.php');

Яка імпортує змінні для підключення до бази даних.

1. Створення системи реєстрації.

Для початку треба створити звичайну html сторінку, а вже потім необхідні скрипти, та підключити до неї.

Для кожної html сторінки використовуються ті самі початкові теги типу <html>, <head>, <body> і тому подібні, тому немає сенсу їх описувати кожного разу. Створимо registration.html:

```
<!DOCTYPE html>
<html>
<meta charset="UTF-8">
<head>
  <title>Панель реєстрації</title>
  <link rel="stylesheet" type="text/css" href="registration.css">
  <script src="reg.js"></script>
</head>
<body>
  <div class="login-panel">
    <form id="registration-form" method="post">
        <input type="text" name="login" placeholder="Логін" autocomplete="off" required>
        <input type="text" name="email" placeholder="Email" autocomplete="off" required>
        <input type="password" name="password" placeholder="Пароль" autocomplete="off" required>
        <button class="custom-button"></button>
    </form>
  </div>
</body>
</html>
```

Рис. 4. 4

В кожній html сторінці ми будемо посилатися на необхідні css файли, а також js скрипти використовуючи:

<link rel="stylesheet" type="text/css" href="файлстилей.css">

та <script src="скрипт.js"></script>.

Для сторінки реєстрації треба створити контейнер , який буде містити форму, яка містить поля вводу та кнопку для відправки даних з полей вводу у js скрипт, який буде обробляти дані та натискання на кнопку, та відправляти їх у php скрипт, який буде ці дані вносити в базу даних, та повертати інформацію про те, успішно він зареєстрував користувача чи ні.

Створимо js скрипт під назвою reg.js.

Реалізація така:

- Додаємо обробник події DOMContentLoaded до документа. Цей обробник буде виконаний, коли весь HTML-код сторінки завантажується та обробляється браузером.
- Ідентифікуємо форму реєстрації за її ID.
- Додаємо обробник події submit до форми реєстрації. Цей обробник буде виконаний, коли користувач натисне кнопку "Відправити" на формі.
- Відміняємо відправку форми за замовчуванням. Це необхідно для того, щоб мати можливість обробити дані форми перед їх відправкою на сервер.
- Створюємо об'єкт FormData з форми реєстрації. Об'єкт FormData являє собою набір пар ключ-значення, які представляють дані форми.

- Відправляємо дані форми на сервер. Метод fetch() використовується для відправки HTTP-запитів. У даному випадку використовується POST-запит, який відправляє дані форми на сервер.
- Перевіряємо статус відповіді сервера. Якщо статус не OK, викидається помилка.
- Повертаємо JSON-відповідь сервера.
- Обробляємо JSON-відповідь сервера. Якщо реєстрація пройшла успішно, користувачеві виводиться повідомлення про це і він перенаправляється на сторінку входу. Якщо реєстрація не пройшла успішно, користувачеві виводиться повідомлення про помилку.
- Обробляємо помилки, які можуть виникнути під час виконання запиту.

fetch() використовує AJAX для відправки HTTP-запитів. AJAX - це технологія, яка дозволяє веб-сторінкам оновлюватися без перезавантаження.

Ось як буде виглядати код:

```
document.addEventListener("DOMContentLoaded", function () {
  var registrationForm = document.getElementById("registration-form");
  registrationForm.addEventListener("submit", function (event) {
    event.preventDefault();
    var formData = new FormData(registrationForm);
    fetch("process_registration.php", {
        method: "POST",
        body: formData,
    })
        .then(function (response) {
            if (!response.ok) {
                throw new Error("Помилка HTTP: " + response.status);
            }
            return response.json();
        })
        .then(function (data) {
            console.log("Дані від серверу:", data);

            if (data.success) {
                alert("Реєстрація пройшла успішно.");
                window.location.href = "login.html";
            } else {
                alert("Помилка при реєстрації: " + data.message);
            }
        })
        .catch(function (error) {
            console.error("Помилка: " + error);
        });
  });
});
```

Рис. 4. 5

Тепер створимо process_registration.php, до якого буде звертатись js скрипт, щоб він зареєстрував користувача.

Він працює наступним чином:

1. Спочатку код підключається до бази даних, використовуючи функцію mysqli_connect().
2. Потім код отримує дані про користувача з POST-запиту.
3. Код хешує пароль користувача за допомогою функції password_hash().
4. Код формує SQL-запит для вставки нового користувача в базу даних.
5. Код виконує SQL-запит за допомогою функції mysqli_query().
6. Якщо запит виконано успішно, код повертає JSON-об'єкт з повідомленням про успішну реєстрацію.
7. Якщо запит не виконано успішно, код повертає JSON-об'єкт з повідомленням про помилку.
8. Нарешті, код закриває з'єднання з базою даних за допомогою функції mysqli_close().

Ось як буде виглядати код:

```php
<?php
    include('config.php');
    $conn = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);
    if (!$conn) {
        die("Помилка підключення до бази даних: " . mysqli_connect_error());
    }
    $login = $_POST["login"];
    $email = $_POST["email"];
    $password = password_hash($_POST["password"], PASSWORD_BCRYPT);
    $sql = "INSERT INTO users (login, email, password) VALUES ('$login', '$email', '$password')";
    if (mysqli_query($conn, $sql)) {
        echo json_encode(array("success" => true, "message" => "Success."));
    } else {
        echo json_encode(array("success" => false, "message" => "Error: " . mysqli_error($conn)));
    }
    mysqli_close($conn);
?>
```

Рис. 4. 6

Тепер, якщо перейти по посиланню http://localhost/simplifychat/registration.html, подивимось на результат:

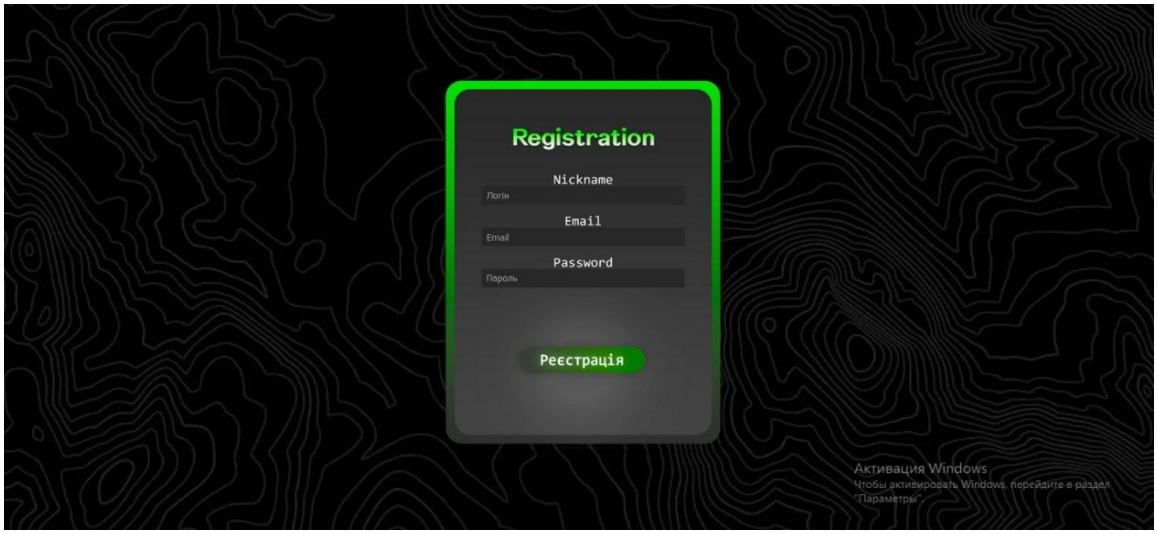

Рис. 4. 7

Для того, щоб сторінка мала такий вигляд, було застосовано необхідні стилі в css, були розроблені необхідні картинки в графічному редакторі, та це було все підключено фоном, також кнопка має фон картинки яка була зроблена в графічному редакторі, до кнопки було застосовано автоматичне збільшення при наведені, а також підсвічення.

registration.css:

```
body {
    margin: 0; padding: 0; background-image: url('photoshop/background.jpg'); background-size: cover;background-repeat: no-repeat;
    background-attachment: fixed; font-family: Arial, sans-serif;
}
.login-panel {
    background-image: url('photoshop/regpanel.png');background-size: cover;position: relative;width: 350px;height: 490px;margin: 100px auto;
    padding: 20px;border-radius: 10px;box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
}
.login-panel input[type="text"]{
    position: absolute;color: #ffffff;top: 153px; left: 55px; width: 68%; padding: 10px; margin-bottom: 10px; border: none;background: transparent; border-radius: 5px;
    outline: none;
}
.login-panel input[name="email"]{
    position: absolute;color: #ffffff;top: 212px; left: 55px; width: 68%; padding: 10px;margin-bottom: 10px;border: none; background: transparent; border-radius: 5px;
    outline: none;
}
.login-panel input[type="password"] {
    position: absolute;color: #ffffff;top: 269px; left: 55px; width: 68%; padding: 10px;  margin-bottom: 10px; border: none; background: transparent;
    border-radius: 5px;outline: none;
}
 .login-panel input[type="text"]:focus,
.login-panel input[type="password"]:focus {

}
.login-panel button.custom-button {
    background-image: url('photoshop/regbutton.png');background-size: cover;background-color: rgba(255, 255, 255, 0);position: absolute;
    top: 350px; left: 97px; width: 200px; height: 80px; border: none; color: #ffffff; font-size: 16px; cursor: pointer; filter: brightness(1);
    transition: transform 0.3s, filter 0.3s;
}
.login-panel button.custom-button:hover {
    transform: scale(1.1);filter: brightness(1.1);
}
```

Рис. 4. 8

2. Створення системи авторизації.

Сторінка авторизації буде виглядати також само як і реєстрації, але буде мати 2 поля вводу, тому сенсу розглядати html код сторінки немає, так як він такий самий.

Єдина відмінність буде полягати в тому, що js скрипт не буде застосовуватись, а замість нього буде тільки php скрипт, який ми підключемо до html сторінки таким чином:

<form action="login.php" method="post">

Просто замість id, зробили для форми action, при вводі даних у поля вводу та натисканні кнопки Авторизація, буде виконуватись цей php код.
Тепер створимо сам login.php.

Він працюватиме наступним чином:

1. Спочатку код перевіряє, чи використовував користувач метод POST для відправки даних. Якщо ні, код виводить повідомлення про помилку і завершує свою роботу.
2. Потім код отримує логін і пароль користувача з POST-запиту.
3. Код формує SQL-запит для вибірки користувача з бази даних за логіном.
4. Код виконує SQL-запит і отримує результат.
5. Якщо в результаті вибірки було знайдено одного користувача, код перевіряє пароль користувача.
6. Якщо пароль користувача вірний, код створює сесію для користувача і перенаправляє його на головну сторінку веб-сайту.

7. Якщо пароль користувача невірний, код виводить повідомлення про помилку і пропонує користувачеві спробувати ще раз.
8. Якщо в результаті вибірки не було знайдено жодного користувача, код виводить повідомлення про помилку і пропонує користувачеві спробувати ще раз.
9. Нарешті, код закриває з'єднання з базою даних.

Код , який реалізує цей процес:

```php
<?php
session_start();
include('config.php');
$conn = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);
if (!$conn) {
    die("Помилка підключення до бази даних: " . mysqli_connect_error());
}
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST["login"];
    $password = $_POST["password"];
    $query = "SELECT * FROM users WHERE login = '$username'";
    $result = mysqli_query($conn, $query);

    if (mysqli_num_rows($result) == 1) {
        $row = mysqli_fetch_assoc($result);
        if (password_verify($password, $row["password"])) {
            $_SESSION["username"] = $username;
            header("Location: main.php");
            exit();
        } else {
            echo "Невірний пароль. <a href='login.html'>Спробувати ще раз</a>";
        }
    } else {
        echo "Користувач з таким логіном не знайдений. <a href='login.html'>Спробувати ще раз</a>";
    }
    mysqli_close($conn);
}
?>
```

Рис. 4. 9

Після успішної авторизації, користувача перекидує на головну сторінку, де вже будуть всі його особисті переписки та дані.

Також під цю сторінку трохи було перероблено картинку реєстрації, щоб було тільки 2 поля вводу та замість напису реєстрації, був напис авторизації.

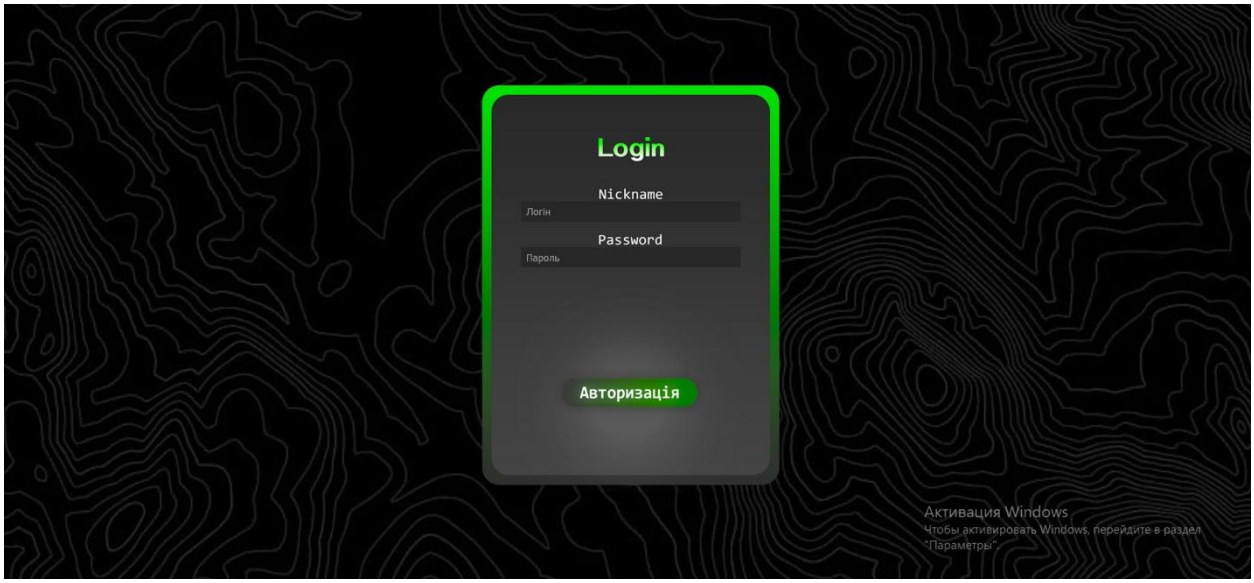

Рис. 4. 10

3. Створимо головну сторінку.

Тепер, треба розробити головну сторінку, ця сторінка буде містити весь основний функціонал системи обміну миттєвих повідомлень.

Сторінка буде мати назву main.php, розглянемо код:

```
<!DOCTYPE html>
<html>
<head>
    <title>Головна сторінка</title>
    <link rel="stylesheet" type="text/css" href="main.css">
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">

</head>
<body>

    <?php
    include('checkuser.php');
    include('vars.php');
    $username2 = $username;
    ?>
    <!-- <a href="logout.php">Выход</a> -->
    <div class="container">
        <div class="left-sidebar">
            <div class="newchatbutton" id="openChatModal">New chat</div>
            <div class="menubutton"></div>
            <div id="chatModal" class="modal">
              <div class="modal-content">
                <span class="close" id="closeChatModal">&times;</span>
                <h2>Створити новий чат</h2>
                <input type="text" id="searchInput" placeholder="Пошук по login...">
                <button id="createChatButton">Створити новий чат</button>
                <div id="suggestions"></div>
                <div id="successMessage" class="modal2" style="display: none;">
                <div class="modal-content2">
                    <span class="close" id="closeSuccessMessage">&times;</span>
                    <h2>Чат успішно створено</h2>
                </div>
                </div>
              </div>
            </div>
            <div id="chatList">
                <!-- Тут будуть відображатись чати користувача -->
            </div>
        </div>
        <div class="right-content">
            <div class="top-sidebar">
                <div id="chat-info">

                </div>
```

Рис. 4. 11

Для того щоб сторінка мала нормальний шрифт, ми застосуємо шрифт під назвою Roboto.

Ось як додамо його в html сторінку:

<link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">

А також треба вказати цей фон в файлі css.

Головна сторінка буде мати container, який буде мати 2 основних контейнера: right-content та left-sidebar.

В right-content буде top-sidebar.

Left-sidebar буде з лівої сторони, та займати десь 30% від екрану,

Right-content буде розташовано по правій частині екрану та займати все інше місце.

Left-sidebar буде мати кнопку “New chat”, розташовану в правій верхній частині цього контейнера, а також по лівій верхній стороні контейнера буде кнопка “Меню”, при натисканні якої буде виводитись вікно в якому буде кнопка виходу з системи.

Все інше що йде під цими двома кнопками, це буде виводитись список чатів при відкритті цієї сторінки.

```
            <div id="chatList">
                <!-- Тут будуть відображатись чати користувача -->
            </div>
        </div>
        <div class="right-content">
            <div class="top-sidebar">
                <div id="chat-info">

                </div>
            </div>
            <!-- Чат -->
            <div class="chat-bar">
                <div id="chat">
                <!-- Повідомлення чату будуть завантажуватися з використанням JavaScript -->
                </div>
                <div id="contextMenu" style="display: none;">
                  <button id="deleteMessage">Видалити</button>
                </div>
                <div id="message-input-container" style="display: none;">
                <input type="text" id="messageInput" placeholder="Введіть повідомлення...">
                <button id="sendMessageButton"></button>
                </div>
            </div>
        </div>
    <script>
    const currentUser = <?php echo $currentUser; ?>;
    </script>
    <script src="createnewchat.js"></script>
    <script src="main.js"></script>
    <script src="loadmessages.js"></script>
    <script src="updatemessages.js"></script>
    <script src="sendmessage.js"></script>
    <script src="createchatmenu.js"></script>
    <script src="suggestusers.js"></script>
    <!--<script src="rightclickonmessage.js"></script>-->
    <!--<script src="deletemessage.js"></script>-->
    <!--<script src="test.js"></script>-->
</body>
</html>
```

Рис. 4. 12

Top-sidebar буде вкладений в right-content, буде зверху чата та буде відображати ім’я співбесідника.

Для виводу списку чатів, а також виведення повідомлень при кліку на чат, ми створимо контейнери , але про це пізніше.

Тепер, як сторінка створена, нам треба реалізувати важливу технічну частину, доступу до неї.

Нам потрібно, щоб при заході користувача на головну сторінку, система перевіряла, чи авторизований користувач, чи ні.

Для того щоб це зробити, створимо checkuser.php та одразу додамо його в наш main.php файл таким чином:

```
<?php

    include('checkuser.php');

    $username2 = $username;

    ?>
```

Він також буде повертати значення змінної нинішнього користувача.

Тепер створимо сам checkuser.php:

1. Спочатку код перевіряє, чи є у користувача сесія. Якщо ні, код перенаправляє користувача на сторінку входу.
2. Потім код отримує логін користувача з сесії.
3. Код формує SQL-запит для вибірки email користувача з бази даних за логіном.
4. Код виконує SQL-запит і отримує результат.
5. Якщо результат запиту не порожній, код отримує email користувача з результату запиту.
6. Код формує SQL-запит для вибірки id користувача з бази даних за логіном.
7. Код виконує SQL-запит і отримує результат.
8. Якщо результат запиту не порожній, код отримує id користувача з результату запиту.
9. Нарешті, код закриває з'єднання з базою даних.

Код:

```php
<?php
include('config.php');
session_start();
if (!isset($_SESSION["username"])) {
    header("Location: login.html");
    exit();
}
$username = $_SESSION["username"];
$conn = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);

if (!$conn) {
    die("Помилка підключення до бази даних: " . mysqli_connect_error());
}
$query = "SELECT email FROM users WHERE login = '$username'";
$result = mysqli_query($conn, $query);
if ($result) {
    $row = mysqli_fetch_assoc($result);
    $useremail = $row['email'];
    mysqli_free_result($result);
} else {
    echo "Помилка запиту: " . mysqli_error($conn);
}
$query2 = "SELECT id FROM users WHERE login = '$username'";
$result2 = mysqli_query($conn, $query2);
if ($result2) {
    $row2 = mysqli_fetch_assoc($result2);
    $currentUser = $row2['id'];
    mysqli_free_result($result2);
} else {
    echo "Помилка запиту: " . mysqli_error($conn);
}
?>
```

Рис. 4. 13

Тепер, після того як головна сторінка була створена, час переходити до створення функціоналу серверної сторони, яка буде відповідати за створення чатів, вивід чатів, вивід повідомлень, відправка повідомлень, та їх оновлення у режимі реального часу.

4. Розробка системи створення чатів.

Для того щоб створити чат, для початку нам треба оформити це візуально, а потім створити та підключити необхідні скрипти.

Нам необхідно, щоб при натисканні кнопки New chat, з'являлося модальне вікно, де користувач може шукати інших користувачів по їх імені, щоб була кнопка Створити новий чат і щоб при пошуку користувача з'являлися похожі або сам користувач, максимум до трьох, щоб не навантажувати систему та самого користувача, також щоб при натисканні на знайденого користувача, він виділявся, а вже потім натискати на кнопку Створити новий чат.

Для початку додамо html розмітку до нашого main,php у left-sidebar:

```
<div class="newchatbutton" id="openChatModal">New chat</div>
<div class="menubutton"></div>
<div id="chatModal" class="modal">
  <div class="modal-content">
    <span class="close" id="closeChatModal">&times;</span>
    <h2>Створити новий чат</h2>
    <input type="text" id="searchInput" placeholder="Пошук по login...">
    <button id="createChatButton">Створити новий чат</button>
    <div id="suggestions"></div>
    <div id="successMessage" class="modal2" style="display: none;">
    <div class="modal-content2">
        <span class="close" id="closeSuccessMessage">&times;</span>
        <h2>Чат успішно створено</h2>
    </div>
    </div>
  </div>
</div>
```

Рис. 4. 14

Та задамо модальному вікну параметри в css для того щоб воно розміщувалось по центру екрану та мало вигляд такий який нам треба.

Тепер розробимо необхідні js скрипти createnewchat.js, createchatmenu.js, suggestusers.js та php скрипти search_user.php, create_chat.php, search.php.

Розглянемо їх:

Створюємо новий чат - createnewchat.js:

```
document.addEventListener("DOMContentLoaded", function() {
    const createChatButton = document.getElementById("createChatButton");
    createChatButton.addEventListener("click", () => {
        const selectedUser = document.querySelector(".searchresult-selected");
        const userLogin = selectedUser.id;
        console.log("User Login :", userLogin);
        fetch('search_user.php', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ login: userLogin }),
        })
        .then(response => response.json())
        .then(data => {
            if (data && data.id) {
                const userId = data.id;
                fetch('create_chat.php', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({ user1_id: currentUser, user2_id: userId }),
                })
```

Рис. 4. 15

Спочатку він створює запит у php скрипт для пошуку користувача, а потім вже робить запит для створення самого чата, якщо користувача знайдено.

```
                .then(response => response.json())
                .then(result => {
                    if (result.success) {
                        console.log('Чат успішно створено.');
                        const successMessage = document.getElementById("successMessage");
                        successMessage.style.display = "block";
                        const closeSuccessMessage = document.getElementById("closeSuccessMessage");
                        closeSuccessMessage.addEventListener("click", () => {
                          successMessage.style.display = "none";
                        });
                        window.addEventListener("click", (event) => {
                          if (event.target === successMessage) {
                            successMessage.style.display = "none";
                          }
                        });
                    } else {
                        console.error('Помилка при створенні чату.');
                    }
                })
                .catch(error => console.error('Помилка при відправці запросу на створення чату: ', error));
            } else {
                console.error('Користувач з таким логіном не знайдений.');
            }
        })
        .catch(error => console.error('Помилка при відправці запросу на пошук: ', error));
    });
});
```

Рис. 4. 16

Відкриття модального вікна - createchatmenu.js:

```
const openChatModal = document.getElementById("openChatModal");
const chatModal = document.getElementById("chatModal");
openChatModal.addEventListener("click", () => {
  chatModal.style.display = "block";
});
const closeChatModal = document.getElementById("closeChatModal");
closeChatModal.addEventListener("click", () => {
  chatModal.style.display = "none";
});
window.addEventListener("click", (event) => {
  if (event.target === chatModal) {
    chatModal.style.display = "none";
  }
});
```

Рис. 4. 17

Пошук користувачів – suggestusers.js:

```
const searchInput = document.getElementById("searchInput");
const suggestions = document.getElementById("suggestions");
let selectedDivId = null;
function selectDiv(id) {
  const divs = suggestions.getElementsByClassName("searchresult");
  for (const div of divs) {
    div.classList.remove("searchresult-selected");
  }
  const selectedDiv = document.getElementById(id);
  if (selectedDiv) {
    selectedDiv.classList.add("searchresult-selected");
    selectedDivId = id;
  }
}
searchInput.addEventListener("input", () => {
  const inputValue = searchInput.value.trim();
  suggestions.innerHTML = "";
```

Рис. 4. 18

suggetusers.js потрібен для того щоб при вводі у пошукове поле підказувати схожі імена користувачів. Буде виведено максимум 3 користувача для вибору.

```
18
19    if (inputValue !== "") {
20      fetch("search.php?query=" + inputValue)
21        .then((response) => response.json())
22        .then((data) => {
23          suggestions.innerHTML = "";
24          data.forEach((account) => {
25            const div = document.createElement("div");
26            div.classList.add("searchresult");
27            div.textContent = account.login;
28            div.setAttribute("id", account.login);
29            div.addEventListener("click", () => {
30              selectDiv(account.login);
31            });
32
33            suggestions.appendChild(div);
34          });
35        })
36        .catch((error) => {
37          console.error("Помилка при запиті: ", error);
38        });
39    }
40  });
41  const createChatButton = document.getElementById("createChatButton");
42  createChatButton.addEventListener("click", () => {
43    if (selectedDivId) {
44      console.log("Обраний ID: " + selectedDivId);
45    } else {
46      console.log("Жодного користувача не обрано");
47    }
48  });
```

Рис. 4. 19

Розглядати search_user.php та create_chat.php скрипти сенсу немає, так как там код буде той самий що й в минулих php скриптах, єдине що відрізняється це команда sql.

Для пошуку користувачів у search_user.php застосовується SQL команда:

SELECT id FROM users WHERE login = '$userLogin'

Для створення чату:

INSERT INTO chats (user1_id, user2_id) VALUES ('$user1Id', '$user2Id')

Єдиний php скрипт що розглянемо це search.php, а саме те як він видає тільки трьох користувачів:

```php
$query = $_GET["query"];
$sql = "SELECT login FROM users WHERE login LIKE ?";
$stmt = $conn->prepare($sql);
$query = '%' . $query . '%';
$stmt->bind_param("s", $query);
$stmt->execute();
$result = $stmt->get_result();
$maxResults = 3;
$exactMatch = null;
$suggestedResults = [];
while ($row = $result->fetch_assoc()) {
    if ($row["login"] === $_GET["query"]) {
        $exactMatch = $row;
    } else {
        $suggestedResults[] = $row;
    }
    if (count($suggestedResults) >= $maxResults) {
        break;
    }
}
if ($exactMatch) {
    echo json_encode([$exactMatch]);
} else {
    echo json_encode($suggestedResults);
}
```

Рис. 4. 20

Він робить запит SQL, а потім сортує так щоб було максимум 3 користувача у відповіді до js скрипту.

Ось як це буде візуально виглядати:

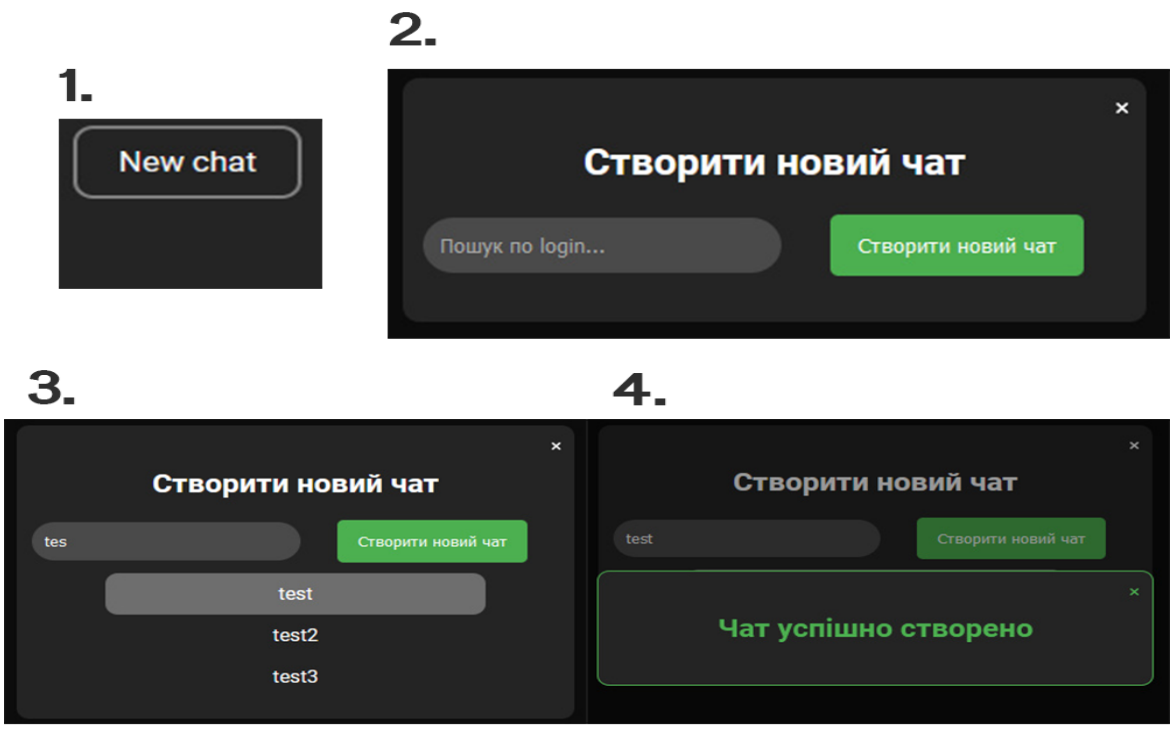

Рис. 4. 21

5. Вивід чатів користувача.

Тепер нам треба зробити, щоб при заході чи оновленні головної сторінки, з’являлися чати користувача.

Для початку додамо у left-sidebar новий контейнер:

У нього будуть додаватися чати в яких є користувач.

Тепер створимо js та php скрипти.

Створимо main.js:

```
document.addEventListener("DOMContentLoaded", function() {
    function parseMessageTime(messageTime) {
      const [datePart, timePart] = messageTime.split(' ');
      const [year, month, day] = datePart.split('-');
      const [hour, minute, second] = timePart.split(':');
      return new Date(year, month - 1, day, hour, minute, second);
    }
    function displayChats(chats) {
      const chatListElement = document.getElementById("chatList");
      chatListElement.innerHTML = '';
      const chatsWithMessages = [];
      const chatsWithoutMessages = [];
      async function fetchLastMessageAndProcess(chat, chatElement) {
        const currentUserIsUser1 = chat.user1_id === currentUser;
        const otherUserId = currentUserIsUser1 ? chat.user2_id : chat.user1_id;
        const otherUserName = currentUserIsUser1 ? chat.user2_name : chat.user1_name;
        chatElement.setAttribute("chat-user-name", otherUserName);

        const data = {
          chatId: chat.id,
        };
        return fetch('lastmessage.php', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(data),
        })
        .then(response => {
          if (!response.ok) {
            throw new Error('Помилка HTTP: ' + response.status);
          }
          return response.text();
        })
        .then(data => {
          try {
            console.log('Відповідь сервера:', data);
            const jsonData = JSON.parse(data);
            const lastMessageTime = jsonData.created_at.slice(11, 16);
            const lastMessageTime2 = jsonData.created_at;
            const lastMessage = jsonData.message;
```

Рис. 4. 22

Цей скрипт буде додавати контейнери chat-item(чати користувача) у контейнер chatList.

Він буде брати інформацію о чатах користувача за допомогою php скрипта listchats.php.

Відображатись у контейнері будуть ім’я співбесідника, останнє повідомлення та час останнього повідомлення.

```
            chatElement.innerHTML = `<div class="username-text">${otherUserName}</div><div class="lastmessage">${lastMessage}</div><div class="lastmessagetime">${lastMe
            chatsWithMessages.push(chatElement);
          } catch (error) {
            chatElement.innerHTML = `<div class="username-text">${otherUserName}</div><div class="newchat">New chat</div>`;
            chatsWithoutMessages.push(chatElement);
          }
        })
        .catch(error => {
          chatElement.innerHTML = `<div class="username-text">${otherUserName}</div><div class="newchat">New chat</div>`;
          chatsWithoutMessages.push(chatElement);
        });
      }
      const fetchPromises = chats.map(chat => {
        const chatElement = document.createElement("div");
        chatElement.classList.add("chat-item");
        chatElement.setAttribute("chat-id", chat.id);
        return fetchLastMessageAndProcess(chat, chatElement);
      });
      Promise.all(fetchPromises)
        .then(() => {
          chatsWithMessages.sort((chatA, chatB) => {
            const timeA = parseMessageTime(chatA.querySelector(".lastmessagetime2").getAttribute("title"));
            const timeB = parseMessageTime(chatB.querySelector(".lastmessagetime2").getAttribute("title"));
            return timeB - timeA;
          });
          chatsWithMessages.forEach(chatElement => {
            chatListElement.appendChild(chatElement);
          });

          chatsWithoutMessages.forEach(chatElement => {
            chatListElement.appendChild(chatElement);
          });
        });
    }
    function listChat() {
        fetch('listchats.php')
            .then(response => {
                if (!response.ok) {
                    throw new Error('Помилка HTTP: ' + response.status);
                }
```


Рис. 4. 23

```
                    return response.text();
                })
                .then(data => {
                    try {
                        const jsonData = JSON.parse(data);
                        displayChats(jsonData);
                    } catch (error) {
                        console.error('Помилка при разборі JSON:', error);
                    }
                })
            .catch(error => console.error('Пмилка при завантаженні даних: ', error));
        }
        //function displayChatPeriodically() {
            //listChat();
            // Задаем интервал повторного вызова функции (например, каждые 5 секунд)
            //setTimeout(displayChatPeriodically, 5000); // 5000 миллисекунд = 5 секунд
        //}
        listChat();
});
```

Рис. 4. 24

```
    chatElement.innerHTML = `<div class="username-text">${otherUserName}</div><div class="lastmessage">${lastMessage}</div><div class="lastmessagetime">${lastMessageTime
    chatsWithMessages.push(chatElement);
  } catch (error) {
    chatElement.innerHTML = `<div class="username-text">${otherUserName}</div><div class="newchat">New chat</div>`;
    chatsWithoutMessages.push(chatElement);
  }
})
.catch(error => {
  chatElement.innerHTML = `<div class="username-text">${otherUserName}</div><div class="newchat">New chat</div>`;
  chatsWithoutMessages.push(chatElement);
});
```

Рис. 4. 25

```
e}</div><div class="lastmessagetime">${lastMessageTime}</div><div class="lastmessagetime2" title="${lastMessageTime2}"></div>`
```

Рис. 4. 26

А також створимо listchats.php:

Він буде виконувати запит до бази даних, та брати звідти всю інформацію про чати користувача та видавати їх у main.js.

```php
<?php
include('config.php');
$conn = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);
if (!$conn) {
    die("Помилка підключення до бази даних: " . mysqli_connect_error());
}
session_start();
if (!isset($_SESSION["username"])) {
    header("Location: login.html");
    exit();
}
$username = $_SESSION["username"];
$query = "SELECT id FROM users WHERE login = '$username'";
$result = mysqli_query($conn, $query);
if ($result) {
    $row = mysqli_fetch_assoc($result);
    $currentUserId = $row['id'];
    mysqli_free_result($result);
} else {
    echo "Помилка запиту: " . mysqli_error($conn);
}
$sql = "SELECT c.id, c.user1_id, c.user2_id, u1.login AS user1_name, u2.login AS user2_name
        FROM chats c
        LEFT JOIN users u1 ON c.user1_id = u1.id
        LEFT JOIN users u2 ON c.user2_id = u2.id
        WHERE c.user1_id = ? OR c.user2_id = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("ii", $currentUserId, $currentUserId);
$stmt->execute();
$result = $stmt->get_result();
$chats = array();
while ($row = $result->fetch_assoc()) {
    $chats[] = $row;
}
$stmt->close();
$conn->close();
header('Content-Type: application/json');
echo json_encode($chats);
?>
```

Рис. 4. 27

У lastmessage.php буде виконуватися SQL запит:

"SELECT m.*, u.login AS sender_name FROM messages m

INNER JOIN users u ON m.sender_id = u.id

WHERE m.chat_id = ?

ORDER BY m.created_at DESC

LIMIT 1"

Тепер подивимось на візуальний результат виводу чатів користувача.

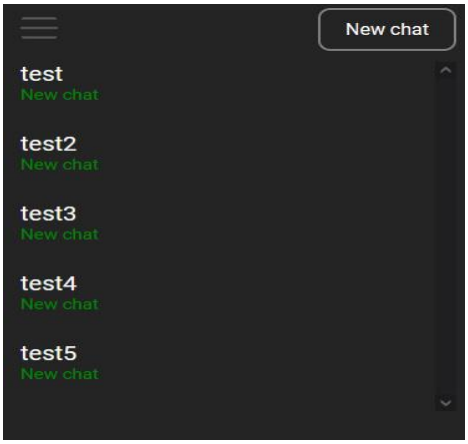

Рис. 4. 28

Ось, як бачимо, якщо повідомлень в чаті немає, то є напис зеленого кольору “New chat”.

6. Вивід, відправлення та оновлення повідомлень.

Тепер додамо контейнери для виводу повідомлень та контейнер у top-sidebar для відображення імені співбесідника.

```
<div class="top-sidebar">
    <div id="chat-info">

    </div>
</div>
<!-- Чат -->
<div class="chat-bar">
    <div id="chat">
    <!-- Повідомлення чату будуть завантажуватися з використанням JavaScript -->
    </div>
    <div id="contextMenu" style="display: none;">
      <button id="deleteMessage">Видалити</button>
    </div>
    <div id="message-input-container" style="display: none;">
    <input type="text" id="messageInput" placeholder="Введіть повідомлення...">
    <button id="sendMessageButton"></button>
    </div>
</div>
```

Рис. 4. 29

Треба створити три js скрипта loadmessages.js, updatemessages.js та sendmessage.js, а також php скрипти messages.php, getchatupdates.php та sendmessage.php.

Почнемо з loadmessages.js, він буде завантажувати всі повідомлення чату при натисканні на чат, а також відобразить поле вводу, кнопку Відправити та ім’я співбесідника

```
function displayMessages(messages) {
  const chatElement = document.getElementById("chat");
  chatElement.innerHTML = '';
  messages.forEach(message => {
    if (message.sender_name === currentUser2) {
      const messageElement = document.createElement("div");
      messageElement.classList.add("message-item2");
      messageElement.id = message.id;
      messageElement.innerHTML = `<span>${message.message}</span><span class="time">${message.created_at.slice(11, 16)}</span>`;
      chatElement.appendChild(messageElement);
    } else {
      const messageElement = document.createElement("div");
      messageElement.classList.add("message-item");
      messageElement.id = message.id;
      messageElement.innerHTML = `<span>${message.message}</span><span class="time">${message.created_at.slice(11, 16)}</span>`;
      chatElement.appendChild(messageElement);
    }
  });
}
function loadMessages(chatId) {
    const chatInfoElement = document.getElementById("chat-info");
    const chatIdValue = chatInfoElement.getAttribute("chat-id");
    const data = {
        chatId: chatIdValue,
    };
    fetch('messages.php', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify(data),
    })
    .then(response => {
        if (!response.ok) {
            throw new Error('Помилка HTTP: ' + response.status);
        }
        return response.text();
    })
    .then(data => {
        try {
            console.log('Відповідь від сервера:', data);
```


Рис. 4. 30

У кожному повідомленні буде відображатись ім’я відправника, саме повідомлення та час відправки. У messages.php буде виконуватись такий запит у базу даних:

"SELECT m.*, u.login AS sender_name FROM messages m

INNER JOIN users u ON m.sender_id = u.id

WHERE m.chat_id = ? ORDER BY m.created_at ASC"

```
                const jsonData = JSON.parse(data);
                displayMessages(jsonData);
            } catch (error) {
                console.error('Помилка при розборі JSON:', error);
            }
        })
        .catch(error => console.error('Помилка при завантаженні даних: ', error));
}
function displayChatInfo(chatId, chatName) {
    const chatInfoElement = document.getElementById("chat-info");
    chatInfoElement.setAttribute("chat-id", chatId);
    chatInfoElement.setAttribute("chat-name", chatName);
    chatInfoElement.innerHTML = `<p>${chatName}</p>`;
}
function displayMessageInput() {
    const messageInputContainer = document.getElementById("message-input-container");
    messageInputContainer.style.display = "block";
}
let isClickActive = true;
document.addEventListener("click", function (e) {
    if (!isClickActive) {
        return;
    }
    isClickActive = false;
    if (e.target && e.target.classList.contains("chat-item")) {
        e.target.focus();
        const chatId = e.target.getAttribute("chat-id");
        const chatName = e.target.getAttribute("chat-user-name");
        displayChatInfo(chatId, chatName);
        displayMessageInput();
        loadMessages(chatId);
        const chatBar = document.querySelector(".chat-bar");
        chatBar.scrollTop = chatBar.scrollHeight;
        const messageInput = document.getElementById("messageInput");
        messageInput.focus();
    }
    setTimeout(function () {
        isClickActive = true;
    }, 10);
});
```

Рис. 4. 31

Тепер розглянемо відправку повідомлень sendmessage.js, він при натисканні кнопки буде відправляти повідомлення.

```
document.addEventListener("DOMContentLoaded", function() {
    const sendMessageButton = document.getElementById("sendMessageButton");
    if (sendMessageButton !== null) {
        sendMessageButton.addEventListener("click", () => {
            const messageInput = document.getElementById("messageInput");
            const message = messageInput.value;
            const chatInfoElement = document.getElementById("chat-info");
            const chatId = chatInfoElement.getAttribute("chat-id");
            const data = {
                chatId: chatId,
                message: message
            };
            fetch('sendmessage.php', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(data),
            })
            .then(response => {
                if (!response.ok) {
                    throw new Error('Ошибка HTTP: ' + response.status);
                }
                return response.text();
            })
            .then(result => {
                console.log(result);
                messageInput.value = '';
            })
            .catch(error => {
                console.error('Помилка при відправці повідомлення: ', error
            });
        });
    }
});
```

Рис. 4. 32

Та розглянемо як sendmessage.php відправляє повідомлення.

```php
$row = mysqli_fetch_assoc($result);
$senderId = $row['id'];
$chatId = $data->chatId;
$message = $data->message;
$sql = "INSERT INTO messages (chat_id, sender_id, message) VALUES (?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("iis", $chatId, $senderId, $message);
if ($stmt->execute()) {
    echo "Повідомлення успішно відправлено.";
} else {
    echo "Помилка при відправці повідомлення: " . mysqli_error($conn);
}
```

Рис. 4. 33

Тепер нам потрібно щоб після кожної відправки повідомлень система оновлювала наш чат в режимі реального часу.

Для цього створимо updatemessages.js. Він буде перевіряти зміни чата у базі даних, так якщо додалося нове повідомлення, то він їх виводить, але не повністю оновлюючи чат, а тільки додає, це потрібно для оптимізації процесу та невеликого навантаження.

```javascript
document.addEventListener("DOMContentLoaded", function() {
    let currentUser2;
    let lastUpdate = "";
    const displayedMessageIds = new Set();
    fetch('getcurrentuser.php')
      .then(response => response.json())
      .then(data => {
        if (data.username) {
          currentUser2 = data.username;
          console.log("Нинішній користувач:", currentUser2);
          console.log(currentUser2);
        } else {
          console.error("Помилка: " + data.error);
        }
      })
      .catch(error => console.error('Помилка при отриманні даних: ', error));
    function displayNewMessages(newMessages) {
        const chatElement = document.getElementById("chat");
        newMessages.forEach(message => {
            if (!displayedMessageIds.has(message.id)) {
                if (message.sender_name === currentUser2) {
                    const messageElement = document.createElement("div");
                    messageElement.classList.add("message-item2");
                    messageElement.id = message.id;
                    messageElement.innerHTML = `<span>${message.message}</span><span class="time">${message.created_at.slice(11, 16)}</span>`;
                    chatElement.appendChild(messageElement);
                } else {
                    const messageElement = document.createElement("div");
                    messageElement.classList.add("message-item");
                    messageElement.id = message.id;
                    messageElement.innerHTML = `<span>${message.message}</span><span class="time">${message.created_at.slice(11, 16)}</span>`;
                    chatElement.appendChild(messageElement);
                }
                displayedMessageIds.add(message.id);
            }
        });
    }
function loadAndRefreshMessages(chatId) {
    const chatInfoElement = document.getElementById("chat-info");
    const chatIdValue = chatInfoElement.getAttribute("chat-id");
    const data = {
```


Рис. 4. 34

Цей скрипт виконується у циклі, щоб кожну секунду(1000 мілісекунд) виконувалася функція для перевірки нових повідомлень.

```
41      const data = {
42        chatId: chatIdValue,
43        lastUpdate: lastUpdate,
44      };
45      fetch('getchatupdates.php', {
46        method: 'POST',
47        headers: {
48          'Content-Type': 'application/json',
49        },
50        body: JSON.stringify(data),
51      })
52      .then(response => {
53        if (!response.ok) {
54          throw new Error('Помилка HTTP: ' + response.status);
55        }
56        return response.text();
57      })
58      .then(data => {
59        try {
60          console.log('Відповідь від сервера:', data);
61          const jsonData = JSON.parse(data);
62          displayNewMessages(jsonData.newMessages);
63          lastUpdate = jsonData.serverTime;
64        } catch (error) {
65          console.error('Помилка при розборі JSON:', error);
66        }
67      })
68      .finally(() => {
69        setTimeout(() => loadAndRefreshMessages(chatId), 1000);
70      });
71    }
72    loadAndRefreshMessages();
73  });
```

Рис. 4. 35

Кожне повідомлення виводиться у контейнер message-item або message-item2.

message-item – це контейнер повідомлень співбесідника.

message-item2 – це контейнер повідомлень нинішнього користувача.

Код перевіряє, якщо нинішній користувач є відправником повідомлення, то тоді додається message-item2, якщо ні, то message-item.

Тепер розглянемо getchatupdates.php.

Він робить запит до бази даних, щоб отримати нові повідомлення, тобто у яких час створення перевищує час у змінній lastUpdate.

З кожним оновленням система змінює останній час оновлення у змінній lastUpdate.

```php
<?php
include('config.php');
$data = json_decode(file_get_contents('php://input'), true);
if (!$data || !isset($data['chatId'])) {
    http_response_code(400);
    echo json_encode(array('error' => 'Невірний запит.'));
    exit();
}
$chatId = $data['chatId'];
$lastUpdate = $data['lastUpdate'];
$conn = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);
if (!$conn) {
    http_response_code(500);
    echo json_encode(array('error' => 'Помилка підключення до бази даних.'));
    exit();
}
$sql = "SELECT m.*, u.login AS sender_name
        FROM messages m
        INNER JOIN users u
        ON m.sender_id = u.id
        WHERE m.chat_id = ?
        AND m.created_at > ?
        ORDER BY m.created_at ASC;";
$stmt = $conn->prepare($sql);
$stmt->bind_param('is', $chatId, $lastUpdate);
$stmt->execute();
$result = $stmt->get_result();
$newMessages = array();
while ($row = $result->fetch_assoc()) {
    $newMessages[] = $row;
}
$serverTime = date('Y-m-d H:i:s.u');
http_response_code(200);
header('Content-Type: application/json');
echo json_encode(array('newMessages' => $newMessages, 'serverTime' => $serverTime));
?>
```

Рис. 4. 36

Тепер подивимось на візуальний результат:

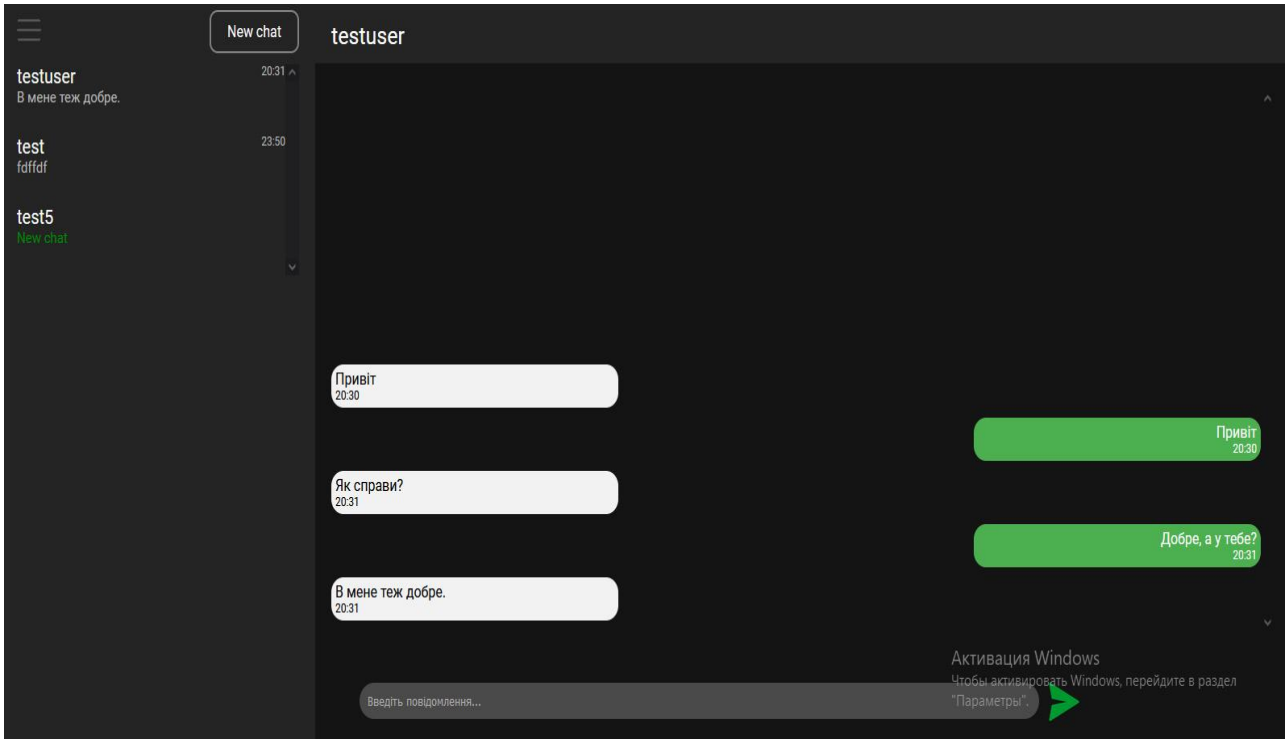

Рис. 4. 37

Файл стилей для main.php - main.css:

```
body, html {
        overflow-y: hidden;height: 100%; margin: 0; padding: 0;  color: white; font-family: 'Roboto', sans-serif; max-width: 100%;
    }

    .container {
        display: flex;height: 100%; max-width: 100%;
    }

    .left-sidebar {
        width: 350px;background-color: #242424; padding: 20px; height: 100%;  position: relative; z-index: 1;max-width: 50%;width: clamp(350px, 50%, 350px);

    }

    .right-content {
        flex: 1; background-color: #141414;  padding: 20px; height: 100%;   display: flex; flex-direction: column;  position: relative;  max-width: 99%;
        width: clamp(200px, 50%, 800px);
    }
    .top-sidebar {
        width: 100%; background-color: #242424;  padding: 20px; height: 8%;   position: absolute;   top: 0;  left: 0;  right: 0;box-sizing: border-box; max-width: 100%;
    }
#searchInput {
    position: relative;  z-index: 2;
}
.chat-item {
    position: relative;  z-index: 1; border-radius: 10px;  transition: background-color 0.3s; max-width: 100%; height: 70px; display: flex;  flex-direction: column;
    align-items: flex-start;
}
.chat-item:hover {
    background-color: #2b2b2b;  cursor: pointer;  filter: brightness(1);
}
#messageInput {
    position: absolute; top: 680px;left: 55px; width: 65%; padding: 10px; border: none;  border-radius: 30px;  z-index: 2;transition: border 0.1s ease-in-out;
    color: white;  background-color: #4a4a4a; max-width: 70%;
}

#messageInput:hover {
    border: 2px solid #888;
}
#sendMessageButton {
    position: absolute; top: 670px; left: 1300px; padding: 0;  width: 38px;   height: 38px;  position: fixed;z-index: 2;margin-top: 10px;
    background: url('photoshop/send-message_button.png') no-repeat center center; background-size: contain;  background-color: rgba(0, 0, 255, 0);
    border: none; cursor: pointer;  outline: none;
}
```

Рис. 4. 38

```
#sendMessageButton:hover {
    background-image: url('photoshop/send-message_button_hover.png');
}
.chat-bar {
    display: flex; flex-direction: column-reverse;  height: 100%; max-width: 100%;max-height: calc(80vh - 8%);  overflow-y: scroll; margin-top: 6%;
    width: clamp(200px, 100%, 1300px);
}
.message-item {
    width: 30%; margin-bottom: 10px;  display: flex;flex-direction: column; align-items: flex-start; padding: 5px; max-width: 70%; background-color: #f2f2f2;
    color: black;border-radius: 15px; word-wrap: break-word; height: auto;
}

.message-item span {
    display: inline-block; max-width: 100%;  word-wrap: break-word;
}

.message-item2 {
    width: 30%; margin-bottom: 10px; display: flex; flex-direction: column; align-items: flex-end; padding: 5px; width: clamp(200px, 50%, 800px);
    background-color: #4CAF50; color: white; border-radius: 15px; word-wrap: break-word; margin-left: auto;  max-width: 30%;
}

.message-item2 span {
    display: inline-block;  max-width: 100%; word-wrap: break-word;
}

.message-input-container {
    margin-top: 10px;
}
#chatList {
    overflow-y: scroll;  max-height: 90vh;  margin-top: 44px;
}
#chat-info {
    margin-top: -25px;font-size: 25px;
}
#chat {
    width: clamp(200px, 100%, 1300px);
}
#chat .time {
    font-size: 12px;
}
.lastmessagetime {
    position: absolute; top: 0;  right: 0;font-size: 12px;color: #bababa;
}
```


Рис. 4. 39

```
}
.username-text {
    font-size: 20px;
}
.lastmessage {
    font-size: 15px;color: #bababa;
}
.newchat {
    font-size: 15px;color: #018706;
}

.newchatbutton {
    display: inline-block; padding: 10px 20px; border: 2px solid gray; border-radius: 10px; text-align: center; cursor: pointer; transition: border-color 0.3s ease;
    float: right; position: relative;top: -10px;
}

.newchatbutton:hover {
    border-color: lightgray;
}
.menubutton {
    width: 30px;  height: 30px;  background-image: url('photoshop/menu.png'); background-size: cover;  cursor: pointer; float: left; position: relative; top: -5px;
    border-radius: 10px;
}
.menubutton:hover {
    background-image: url('photoshop/menuhover.png');
}
.modal {
    display: none; position: fixed;z-index: 1; left: 0;top: 0; width: 100%; height: 100%; overflow: auto; background-color: rgba(0, 0, 0, 0.4); display: flex;
    align-items: center;justify-content: center; text-align: center;
}

.modal-content {
    background-color: #242424;  margin: 10% auto; padding: 20px; border: 1px;
    width: 80%; max-width: 400px; position: relative; border-radius: 10px;
}

.close {
    position: absolute; top: 0;right: 0;padding: 10px; cursor: pointer;
}
#chatModal {
    display: none;
}
```


Рис. 4. 40

```
#searchInput {
    position: relative;right: 25px; width: 48%; padding: 10px;border: none; border-radius: 30px; z-index: 2; transition: border 0.1s ease-in-out; color: white;
    background-color: #4a4a4a; max-width: 70%;
}

#searchInput:hover {
    border: 2px solid #888;
}

#createChatButton {
    z-index: 1;
}
.searchresult {
    width: 80%;  max-width: 300px;  margin: 0 auto;  cursor: pointer; padding: 10px;  transition: background-color 0.2s ease-in-out; border-radius: 10px;
    box-sizing: border-box;
}
.searchresult:hover {
    background-color: #6e6e6e;
}
#suggestions {
    margin-top: 10px;
}
.searchresult-selected {
    width: 80%;  max-width: 300px;  margin: 0 auto;  cursor: pointer; padding: 10px; background-color: #6e6e6e; transition: background-color 0.2s ease-in-out;
    border-radius: 10px; box-sizing: border-box;
}
#createChatButton {
    background-color: #4CAF50; color: white;border: none; padding: 10px 15px; border-radius: 5px; cursor: pointer; transition: background-color 0.3s ease;
    width: 150px;  height: 40px;  box-sizing: border-box;
}
#createChatButton:hover {
    background-color: #3a873d;
}
#successMessage {
    display: none; position: fixed; z-index: 1; left: 0; top: 0; width: 100%; height: 100%; overflow: auto; background-color: rgba(0, 0, 0, 0.4); display: flex;
    align-items: center; justify-content: center;text-align: center; border-radius: 10px;z-index: 1000;
}
```

Рис. 4. 41

```
#createChatButton {
  background-color: #4CAF50; color: white;border: none; padding: 10px 15px; border-radius: 5px; cursor: pointer; transition: background-color 0.3s ease;
  width: 150px;  height: 40px;  box-sizing: border-box;
}
#createChatButton:hover {
  background-color: #3a873d;
}
#successMessage {
  display: none; position: fixed; z-index: 1; left: 0; top: 0; width: 100%; height: 100%; overflow: auto; background-color: rgba(0, 0, 0, 0.4); display: flex;
  align-items: center; justify-content: center;text-align: center; border-radius: 10px;z-index: 1000;
}

#successMessage .modal-content2 {
  background-color: #242424; margin: 10% auto; padding: 20px; border: 1px solid #4CAF50; width: 80%;max-width: 400px; position: relative;
  border-radius: 10px; justify-content: center; margin-top: 300px; color: #4CAF50;
}

#successMessage .close {
  position: absolute;top: 0; right: 0; padding: 10px; cursor: pointer;
}
.deleteMessage {
    position: absolute; top: 0; right: 0; background-color: #f00; color: #fff; padding: 5px 10px; border: none; border-radius: 0 5px 0 0; display: none;cursor: pointer;
    z-index: 1001;
}
.contextMenu {
    z-index: 1001;
}

```


Рис. 4. 42

4.2. Тестування додатку.

Тестування - це процес систематичного перевірки програми на виявлення помилок, аномалій та недоліків. Під час тестування використовуються різні тестові сценарії, які включають в себе різні види вхідних даних та умови використання програми. Мета тестування - забезпечити коректну та надійну роботу програми під час різних ситуацій.

Тестування програмного продукту включає в себе різноманітні види тестів, кожен з яких спрямований на перевірку певних аспектів функціонування програми. Ось декілька основних видів тестування та їх опис:

1. **Модульне (unit testing) тестування:**
   - **Опис:** В цьому виді тестування окремі компоненти (модулі) програми перевіряються на коректність роботи. Зазвичай це перша стадія тестування, де кожна функція або метод перевіряється окремо.
   - **Мета:** Виявлення помилок та недоліків в роботі окремих функцій чи методів програми.
2. **Інтеграційне (integration testing) тестування:**
   - **Опис:** На цій стадії тестуються взаємодії між різними модулями чи компонентами програми. Вимагається встановлення, що всі частини програми взаємодіють коректно.
   - **Мета:** Виявлення помилок, що можуть виникнути під час взаємодії між компонентами програми.
3. **Функціональне (functional testing) тестування:**
   - **Опис:** Перевірка програми з точки зору функціональних вимог. Тестується, чи виконує програма очікувані функції та реагує на вхідні дані відповідно до специфікації.

- **Мета:** Перевірка, чи програма виконує очікувані функції коректно та повністю.

4. **Автоматизоване (automated testing) тестування:**
   - **Опис:** Використання автоматизованих інструментів та скриптів для виконання тестів. Дозволяє швидше та повніше проводити тестування.
   - **Мета:** Забезпечення швидкості та ефективності тестування, особливо в проектах з великим обсягом коду.
5. **Прийняття (acceptance testing) тестування:**
   - **Опис:** Перевірка того, як програма відповідає потребам та очікуванням кінцевих користувачів. Часто виконується в реальному середовищі користувачів.
   - **Мета:** Забезпечення того, що програма задовольняє вимоги та очікування кінцевих користувачів.
6. **Стрес-тестування (stress testing):**
   - **Опис:** Випробування програми при максимальних навантаженнях для визначення, як вона реагує на перевищення ресурсів чи надмірне навантаження.
   - **Мета:** Виявлення слабких місць у програмі та забезпечення її стабільності під час інтенсивного використання.
7. **Безпекове (security testing) тестування:**
   - **Опис:** Перевірка програми на вразливості та можливості вторгнення. Включає в себе тестування на вразливості веб-додатків.
   - **Мета:** Забезпечення безпеки програмного продукту та запобігання можливим атакам.
8. **Сумісність (compatibility testing) тестування:**
   - **Опис:** Перевірка того, як програма працює на різних платформах, операційних системах та браузерах.
   - **Мета:** Забезпечення того, щоб програма була сумісною з різними середовищами користувачів.

Ці види тестування можуть поєднуватися та використовуватися разом, залежно від потреб проекту та програмного продукту. Комбінація різних видів тестування допомагає забезпечити високу якість та надійність програми перед її випуском на ринок.

Проведемо тестування на працездатність та в цілому перевіримо функціонал.

Зайдемо на сторінку реєстрації та зареєструємо користувача.

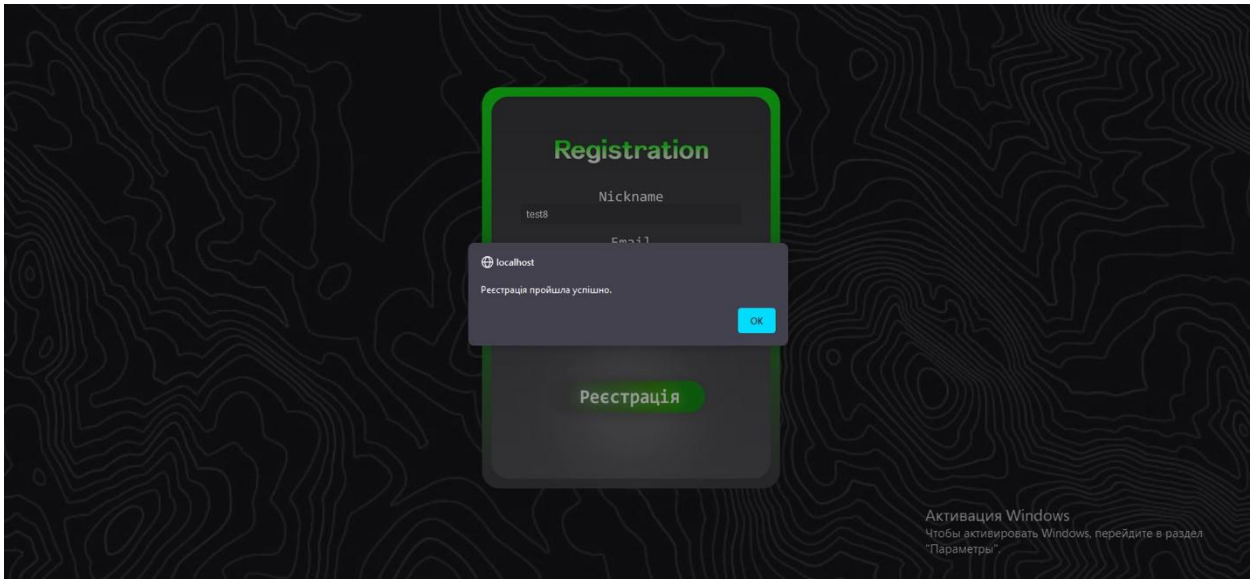

Рис. 4. 43

Нас перекидує на сторінку авторизації.

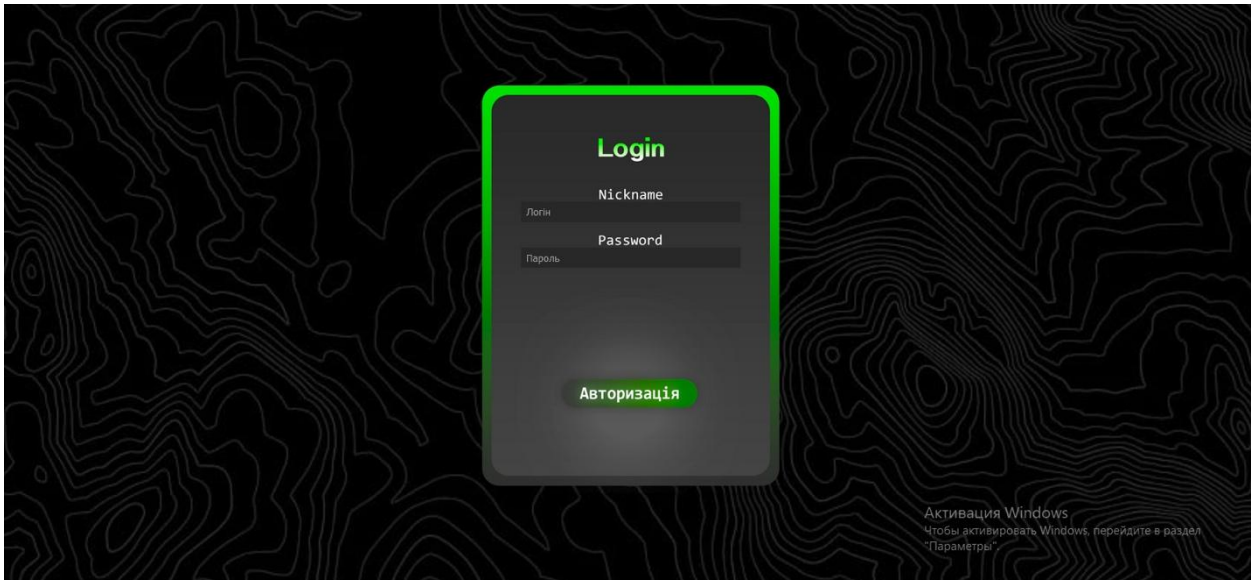

Рис. 4. 44

Після цього входимо в систему за допомогою тих даних яких ми зареєстрували.
Нас перекидує на головну сторінку, все працює успішно.

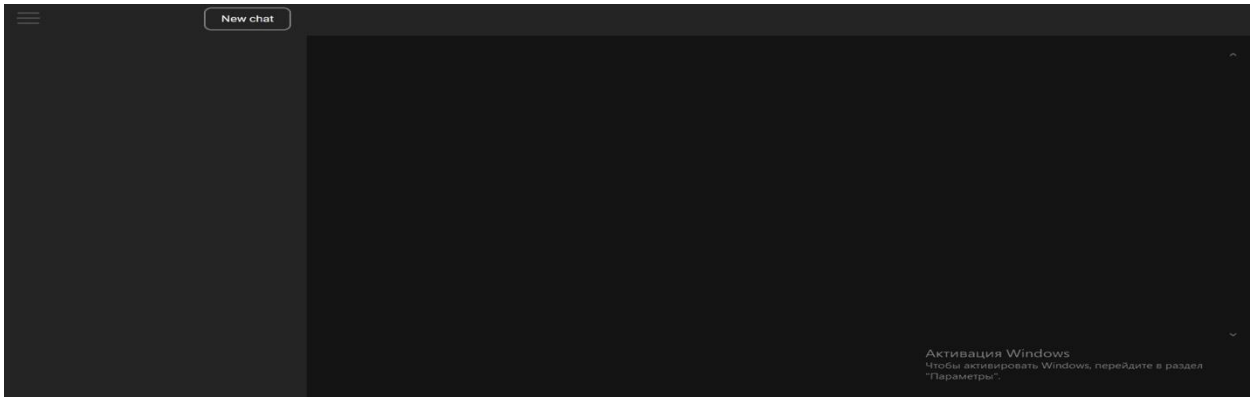

Рис. 4. 45

Створимо новий чат с користувачем testuser.

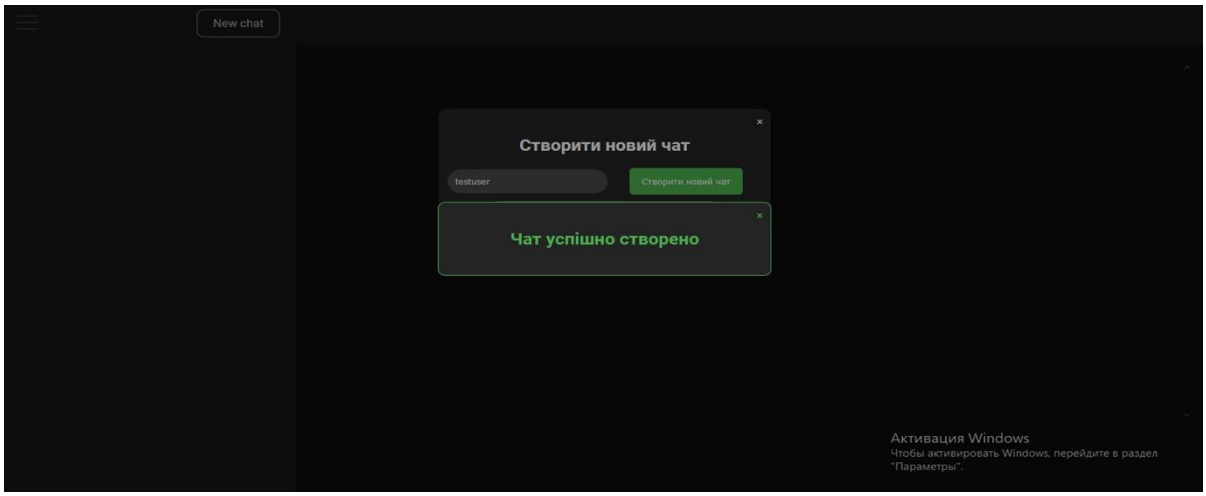

Рис. 4. 46

Як бачимо все пройшло успішно, тепер зайдемо в чат та відправимо повідомлення.

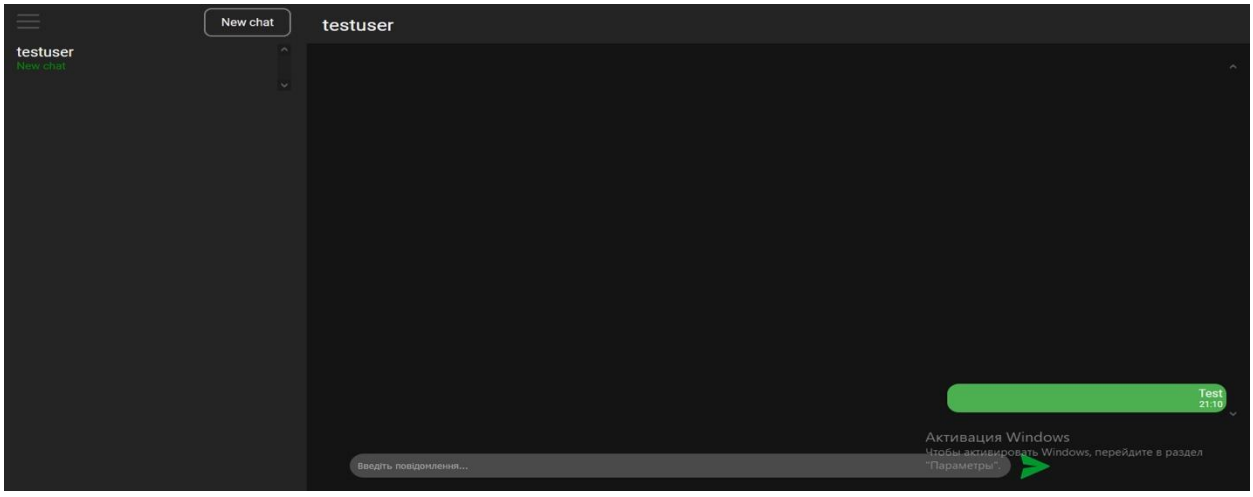

Рис. 4. 47

Повідомлення успішно відправлено. Тестування додатку пройшло успішно, сайт відповідає вимогам поставлених задач та повністю функціонує.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи, була розроблена система обміну миттєвими повідомленнями "Simplify". Розробка даної системи було важливим та актуальним завданням у сучасному інформаційному суспільстві, оскільки системи миттєвих повідомлень стали необхідним інструментом для оперативного обміну інформацією та спілкування в режимі реального часу.

Під час розробки "Simplify" було виконано ряд завдань та досягнуто основних цілей проекту:

1. **Аналіз і вивчення існуючих систем миттєвих повідомлень:** Було проведено детальний аналіз функціональності та особливостей інших систем миттєвих повідомлень для кращого розуміння їх переваг та недоліків.
2. **Проектування структури бази даних:** Була розроблена структура бази даних, яка дозволяє зберігати повідомлення, профілі користувачів та інші необхідні дані.
3. **Розробка інтерфейсу користувача:** Був створений інтуїтивний та зручний інтерфейс для взаємодії користувачів з системою, включаючи можливість обміну текстовими та мультимедійними повідомленнями.
4. **Використання сучасних технологій:** Для розробки системи використовувалися сучасні технології програмування та інструменти, які забезпечили надійність та ефективність роботи системи.
5. **Тестування:** Було проведено тестування додатку "Simplify" для перевірки його відповідності вимогам та очікуванням користувачів. Тестування дозволило виявити і усунути різноманітні помилки та недоліки.

Для успішної реалізації проекту "Simplify" було використано інтегроване середовище розробки XAMPP, що включає в себе веб-сервер Apache, базу даних SQL та мову програмування PHP. Це спростило процес розгортання та забезпечило високу стабільність системи. Створення інтерфейсу користувача виконано за допомогою веб-технологій, таких як HTML для структури, CSS для стилізації та JavaScript для реалізації динамічних елементів. Використання цих технологій дозволило досягти сучасного та зручного дизайну, а також забезпечило високу взаємодію з користувачем. Крім того, мова запитань SQL використовувалася для оптимізованого доступу та управління базою даних, підвищуючи продуктивність та ефективність системи обміну повідомленнями "Simplify".

Загалом, розробка системи обміну миттєвих повідомлень "Simplify" була успішною та відповідала поставленим цілям та завданням проекту.

Результати виконання кваліфікаційної роботи були представлені у вигляді тез на дев’ятнадцятої науково-практичної конференції студентів закладів освіти споживчої кооперації України.

СПИСОК ЛІТЕРАТУРИ

1. Черненко Оксана Олексіївна. "Методичні рекомендації до виконання кваліфікаційної роботи". / Черненко Оксана Олексіївна - Вищий навчальний заклад Укоопспілки «Полтавський університет економіки і торгівлі», 2023 – 68 с.
2. Джон Дакетт. "HTML та CSS: Проектування та створення веб-сайтів". / Джон Дакетт - Wiley, 2011 – 490 с.
3. Девід Фланаган. "JavaScript: Вичерпний посібник". / Девід Фланаган - O'Reilly Media, 2021 – 746 с.
4. Ларрі Уллман. "PHP та MySQL для динамічних веб-сайтів". / Ларрі Уллман - Peachpit Press, 2018 – 704 с.
5. Ерік Мейєр та Естель Вейл. "CSS: Вичерпний посібник". / Ерік Мейєр, Естель Вейл - O'Reilly Media, 2021 – 1088 с.
6. Дженніфер Нідерст Роббінс. "Вивчаємо веб-дизайн: Посібник для початківців з HTML, CSS та веб-графіки". / Дженніфер Нідерст Роббінс - O'Reilly Media, 2018 – 810 с.
7. Робін Ніксон. "Вивчаємо PHP, MySQL та JavaScript". / Робін Ніксон - O'Reilly Media, 2018 – 806 с.
8. Стів Круг. "Не змушуй мене думати, переглянуте: Підхід до юзабіліті на основі здорового глузду". / Стів Круг - New Riders, 2014 – 216 с.
9. Ітан Браун. "Веб-розробка за допомогою HTML, CSS та JavaScript". / Ітан Браун - O'Reilly Media, 2020 – 408 с.
10. Джон Олсопп. "Окрема книга: Адаптивний веб-дизайн". / Джон Олсопп - A Book Apart, 2011 – 183 с.
11. Ніколя Бевакуа. "Розробка додатків на JavaScript: A Build-First Approach". / Ніколя Бевакуа - O'Reilly Media, 2015 – 182 с.
12. Бред Фрост. "Атомарний дизайн". / Бред Фрост - BradFrost, 2016 – 256 с.
13. Шей Хоу. "Вчимося програмувати HTML та CSS: Розробка та стилізація веб-сайтів". / Шей Хоу - Pearson, 2014 – 256 с.
14. Дженніфер Нідерст Роббінс. "Веб-дизайн у двох словах". / Дженніфер Нідерст Роббінс - O'Reilly Media, 2006 – 812 с.
15. Джон Дакетт. "JavaScript та JQuery: Інтерактивна інтерфейсна веб-розробка". / Джон Дакетт - Wiley, 2014 – 640 с.
16. Джессі Джеймс Гарретт. "Елементи користувацького досвіду: Орієнтований на користувача дизайн для Інтернету і не тільки". / Джессі Джеймс Гарретт - New Riders, 2010 – 192 с.
17. Девід Сойєр МакФарланд. "PHP та MySQL: Вичерпний посібник к". / Девід Сойєр МакФарланд - O'Reilly Media, 2012 – 526 с.
18. Адам Фріман. "Pro HTML5 з CSS, JavaScript та мультимедіа". / Адам Фріман - Apress, 2017 – 849 с.
19. Естель Вейл. "CSS: Вичерпний посібник". / Естель Вейл - O'Reilly Media, 2020 – 898 с.

ДОДАТОК А.

СКРИПТ

```
<?php
include('config.php');
$data = json_decode(file_get_contents('php://input'), true);
if (!$data || !isset($data['chatId'])) {
    http_response_code(400);
    echo json_encode(array('error' => 'Невірний запит.'));
    exit();
}
$chatId = $data['chatId'];
$lastUpdate = $data['lastUpdate'];
$conn = mysqli_connect(DB_HOST, DB_USER, DB_PASS, DB_NAME);
if (!$conn) {
    http_response_code(500);
    echo json_encode(array('error' => 'Помилка підключення до бази даних.'));
    exit();
}
$sql = "SELECT m.*, u.login AS sender_name
                FROM messages m
                INNER JOIN users u
                ON m.sender_id = u.id
                WHERE m.chat_id = ?
                AND m.created_at > ?
                ORDER BY m.created_at ASC;";
$stmt = $conn->prepare($sql);
```

```
$stmt->bind_param('is', $chatId, $lastUpdate);
$stmt->execute();
$result = $stmt->get_result();
$newMessages = array();
while ($row = $result->fetch_assoc()) {
    $newMessages[] = $row;
}
$serverTime = date('Y-m-d H:i:s.u');
http_response_code(200);
header('Content-Type: application/json');
echo json_encode(array('newMessages' => $newMessages, 'serverTime' => $serverTime));
?>
document.addEventListener("DOMContentLoaded", function() {
            let currentUser2;
            fetch('getcurrentuser.php')
              .then(response => response.json())
              .then(data => {
                        if (data.username) {
                          currentUser2 = data.username;
                          console.log("Нинішній користувач:", currentUser2);
                          console.log(currentUser2);
                        } else {
                          console.error("Ошибка: " + data.error);
                        }
              })
              .catch(error => console.error('Помилка при отриманні даних: ', error));
            function displayMessages(messages) {
```

```
            const chatElement = document.getElementById("chat");
            chatElement.innerHTML = '';
            messages.forEach(message => {
                if (message.sender_name === currentUser2) {
                  const messageElement = document.createElement("div");
                  messageElement.classList.add("message-item2");
                  messageElement.id = message.id;
                  messageElement.innerHTML = `<span>${message.message}</span><span
class="time">${message.created_at.slice(11, 16)}</span>`;
                  chatElement.appendChild(messageElement);
                } else {
                  const messageElement = document.createElement("div");
                  messageElement.classList.add("message-item");
                  messageElement.id = message.id;
                  messageElement.innerHTML = `<span>${message.message}</span><span
class="time">${message.created_at.slice(11, 16)}</span>`;
                  chatElement.appendChild(messageElement);
                }
             });
          }
          function loadMessages(chatId) {
                const chatInfoElement = document.getElementById("chat-info");
                const chatIdValue = chatInfoElement.getAttribute("chat-id");
                const data = {
                      chatId: chatIdValue,
                };
                fetch('messages.php', {
                      method: 'POST',
                      headers: {
```

```
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(data),
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Помилка HTTP: ' + response.status);
            }
            return response.text();
        })
        .then(data => {
            try {
                console.log('Відповідь від сервера:', data);
                const jsonData = JSON.parse(data);
                displayMessages(jsonData);
            } catch (error) {
                console.error('Помилка при разборі JSON:', error);
            }
        })
        .catch(error => console.error('Помилка при завантаженні даних: ', error));
}
function displayChatInfo(chatId, chatName) {
    const chatInfoElement = document.getElementById("chat-info");
    chatInfoElement.setAttribute("chat-id", chatId);
    chatInfoElement.setAttribute("chat-name", chatName);
    chatInfoElement.innerHTML = `<p>${chatName}</p>`;
}
```

```
    function displayMessageInput() {
        const messageInputContainer = document.getElementById("message-input-container");
        messageInputContainer.style.display = "block";
    }
    let isClickActive = true;
    document.addEventListener("click", function (e) {
        if (!isClickActive) {
            return;
        }
        isClickActive = false;
        if (e.target && e.target.classList.contains("chat-item")) {
            e.target.focus();
            const chatId = e.target.getAttribute("chat-id");
            const chatName = e.target.getAttribute("chat-user-name");
            displayChatInfo(chatId, chatName);
            displayMessageInput();
            loadMessages(chatId);
            const chatBar = document.querySelector(".chat-bar");
            chatBar.scrollTop = chatBar.scrollHeight;
            const messageInput = document.getElementById("messageInput");
            messageInput.focus();
        }
        setTimeout(function () {
            isClickActive = true;
        }, 10);
    });
});
```

```
const searchInput = document.getElementById("searchInput");

const suggestions = document.getElementById("suggestions");

let selectedDivId = null;

function selectDiv(id) {

  const divs = suggestions.getElementsByClassName("searchresult");

  for (const div of divs) {
```