Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Клименко Богдана Сергійовича*
(ПІБ)

академічної групи *121-20-2*
(шифр)

спеціальності *121 «Інженерія програмного забезпечення»*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка програмного забезпечення месенджера для персональних комп'ютерів*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Лактіонов І.С.*			
розділів:				
спеціальний	*проф. Лактіонов І.С.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Мартиненко А.А.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-20-2* *Клименко Богдана Сергійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка програмного забезпечення месенджера для персональних комп'ютерів*

затверджена наказом ректора НТУ «ДП» від *23.05.2024 р.* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів практик та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***10.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***14.06.2024 р.***

Завдання видав _____ *Проф. Лактіонов І.С.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Клименко Б.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*
Термін подання кваліфікаційної роботи до ЕК: *21.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 80 с., 27 рис., 3 дод., 21 джерел.

Об’єкт розробки: програмне забезпечення месенджера для персональних комп'ютерів.

Мета кваліфікаційної роботи: підвищення ефективності та зручності обміну повідомленнями між користувачами персональних комп'ютерів шляхом створення надійного, функціонального та ергономічного месенджера.

У вступі розглядається сучасний стан проблеми та її аналіз, визначається мета кваліфікаційної роботи, а також галузь, у якій вона може бути застосована, наведено обґрунтування актуальності теми та конкретизується постановка завдання.

У першому розділі проведено аналіз предметної галузі, сформульовано актуальність завдання та призначення розробки, розкрито постановку завдання, зазначено вимоги щодо програмної реалізації, програмних засобів та технологій.

У другому розділі проведено дослідження та порівняння наявних рішень, обґрунтовано вибір платформи для розробки, виконано проектування і розробку програмного забезпечення, описана робота програми, її структура та алгоритми функціонування, а також виклик та завантаження програми, вказані вхідні і вихідні дані, та описано склад параметрів технічних засобів.

В економічному розділі було встановлено трудомісткість розроблення програмного забезпечення, проведено розрахунок вартості робіт зі створення програми та визначено необхідний час для її розробки.

З кожним роком зростає популярність персональних комп'ютерів як інструменту для комунікації та співпраці. Сучасні вимоги до месенджерів на ПК включають надійність, швидкість та зручність у використанні. Розробка ефективного програмного забезпечення для обміну повідомленнями стає ключовим завданням для полегшення комунікації як у персональному, так і у бізнес-середовищі. Ця кваліфікаційна робота спрямована на вдосконалення функціональності та ергономіки месенджерів на персональних комп'ютерах, що робить її актуальною та важливою в сучасних умовах цифрової комунікації.

Практичне завдання полягає у проведенні аналізу наявних рішень для програмного забезпечення месенджерів, створення дизайну та прототипів, спираючись на дослідження, а також розробці месенджера за дизайном.

Список ключових слів: МЕСЕНДЖЕР, ОБМІН ПОВІДОМЛЕННЯМИ, ПЕРСОНАЛЬНИЙ КОМП'ЮТЕР, МЕРЕЖЕВИЙ ПРОТОКОЛ, SQL,Javaicript,HMTL,CSS, JSON.

Abstract

Explanatory Note: 77 pages, 27 figures, 3 appendices, 21 references.

Object of development: messenger software for personal computers.

Purpose of the qualification work: to enhance the efficiency and convenience of message exchange between users of personal computers by creating a reliable, functional, and ergonomic messenger.

The introduction discusses the current state of the problem and its analysis, defines the purpose of the qualification work, as well as the field in which it can be applied, provides justification for the relevance of the topic, and specifies the task statement.

In the first chapter, an analysis of the subject area is conducted, the relevance of the task and the purpose of the development are formulated, the task statement is disclosed, requirements for software implementation, software tools, and technologies are indicated.

In the second chapter, research and comparison of existing solutions are conducted, the choice of platform for development is justified, design and development of software are carried out, the operation of the program, its structure, and algorithms of operation are described, as well as the invocation and loading of the program, input and output data are specified, and a description of the parameters of technical means is provided.

In the economic section, the labor intensity of software development is determined, the cost of work on creating the program is calculated, and the necessary time for its development is determined.

With each passing year, the popularity of personal computers as tools for communication and collaboration continues to rise. Modern requirements for PC messengers include reliability, speed, and user-friendliness. Developing efficient software for message exchange becomes a key task in facilitating communication both in personal and business environments. This qualification work aims to enhance the functionality and ergonomics of PC messengers, making it relevant and significant in today's digital communication landscape

The practical task involves analyzing existing solutions for messenger software, creating designs and prototypes based on research, and developing a messenger according to the design.

List of keywords: MESSENGER, MESSAGE EXCHANGE, PERSONAL COMPUTER, NETWORK PROTOCOL, SQL, Java, JSON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК - Персональний Комп'ютер

ІК - Інтерфейс Користувача

ДО - Досвід Користувача

API - Програмний Інтерфейс Додатків

ОС - Операційна Система

ЦП - Центральний Процесор

ОЗП - Оперативна Запам'ятовувальна Пристрій

ЖД - Жорсткий Диск

ГІ - Графічний Інтерфейс

SDK - Набір Засобів Розробки Програмного Забезпечення

MVC - Модель-Вид-Контролер

MVVM - Модель-Вид-Представлення-Модель

AES - Стандарт Шифрування Даних

RSA - Шифрування Рівеста-Шаміра-Адлемана

HTTP - Протокол Передачі Гіпертексту

JSON - Об'єктна Нотація JavaScript

SQL - Мова Структурованих Запитів

ВСТУП

Тематика даної кваліфікаційної роботи присвячена розробці програмного забезпечення месенджера для персональних комп'ютерів. Метою даної кваліфікаційної роботи є створення ефективного та зручного інструменту для обміну повідомленнями між користувачами, що дозволить забезпечити високий рівень надійності, безпеки та зручності використання.

Останні роки спостерігається стрімке зростання популярності месенджерів, як основного засобу комунікації між людьми. Вони стали незамінними інструментами для особистого та ділового спілкування. Відомі месенджери, такі як WhatsApp, Telegram, Signal, Slack, Microsoft Teams та інші, пропонують широкий спектр можливостей: від текстового обміну повідомленнями до відео- та аудіозв'язку, обміну файлами та документами.

Основна ідея створення месенджера полягає в інтеграції всіх необхідних функцій в одному програмному продукті, який буде працювати на персональних комп'ютерах, забезпечуючи користувачам зручний інтерфейс та високу продуктивність.

Основна мета нашого проекту онлайн чат-додатку полягає в розробці надійної та зручної платформи, яка забезпечує безперешкодну комунікацію та співпрацю між користувачами в реальному часі.[1]

Для досягнення мети роботи передбачається вирішення таких завдань:

Аналіз сучасних рішень у сфері месенджерів та визначення їхніх основних функціональних характеристик;

1. Розробка архітектури програмного забезпечення та вибір відповідних технологій для його реалізації;
2. Впровадження функцій текстового обміну повідомленнями, передачі файлів, аудіо- та відео дзвінків;

3. Забезпечення безпеки даних користувачів шляхом використання сучасних методів шифрування;
4. Проведення тестування розробленого програмного продукту на різних операційних системах та пристроях.

Ринок месенджерів на даний момент дуже насичений, проте кожен з існуючих продуктів має свої переваги та недоліки. Наприклад, WhatsApp відомий своєю простотою у використанні та широким розповсюдженням, проте має обмежені можливості в налаштуваннях приватності. Telegram виділяється високим рівнем безпеки та можливістю створення ботів для автоматизації різних процесів, але вимагає більшого обсягу пам'яті. Signal, зосереджений на безпеці та конфіденційності, проте має менш дружній інтерфейс порівняно з іншими месенджерами.

В результаті виконання даної роботи буде створено програмне забезпечення, яке об'єднає найкращі риси існуючих месенджерів та надасть користувачам зручний інструмент для комунікації з можливістю налаштування під індивідуальні потреби.

Ця робота буде корисною для фахівців у галузі розробки програмного забезпечення, а також для користувачів, які шукають надійні та безпечні засоби для спілкування в Інтернеті.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Месенджери — це програмні продукти або додатки, що забезпечують обмін повідомленнями між користувачами через мережу Інтернет. Вони стали основним засобом комунікації в сучасному світі, дозволяючи миттєво передавати текстові повідомлення, файли, зображення, відео, а також здійснювати аудіо- та відеодзвінки [4].

З розвитком мобільних технологій у 2000-х роках, з'явилися мобільні месенджери, такі як BlackBerry Messenger (BBM) та WhatsApp. Мобільні месенджери значно розширили можливості користувачів для обміну повідомленнями та іншою інформацією, незалежно від місця їх знаходження.

WhatsApp є одним з найпопулярніших месенджерів у світі, належить компанії Meta (раніше Facebook). Він дозволяє обмінюватися текстовими повідомленнями, голосовими повідомленнями, зображеннями, відео та файлами, а також здійснювати аудіо- та відеодзвінки. WhatsApp використовує наскрізне шифрування для забезпечення безпеки даних користувачів.

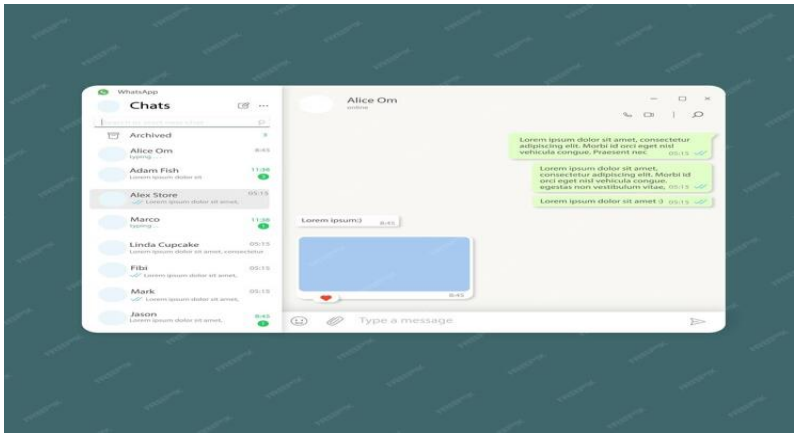

Рис. 1.1. Інтерфейс програми WhatsApp

Telegram, розроблений Павлом Дуровим, відомий своєю безпекою та швидкістю. Він підтримує обмін текстовими повідомленнями, файлами будь-якого типу, а також дозволяє створювати канали для широкої аудиторії та використовувати ботів для автоматизації різних завдань. Telegram також пропонує хмарне зберігання повідомлень та медіафайлів.

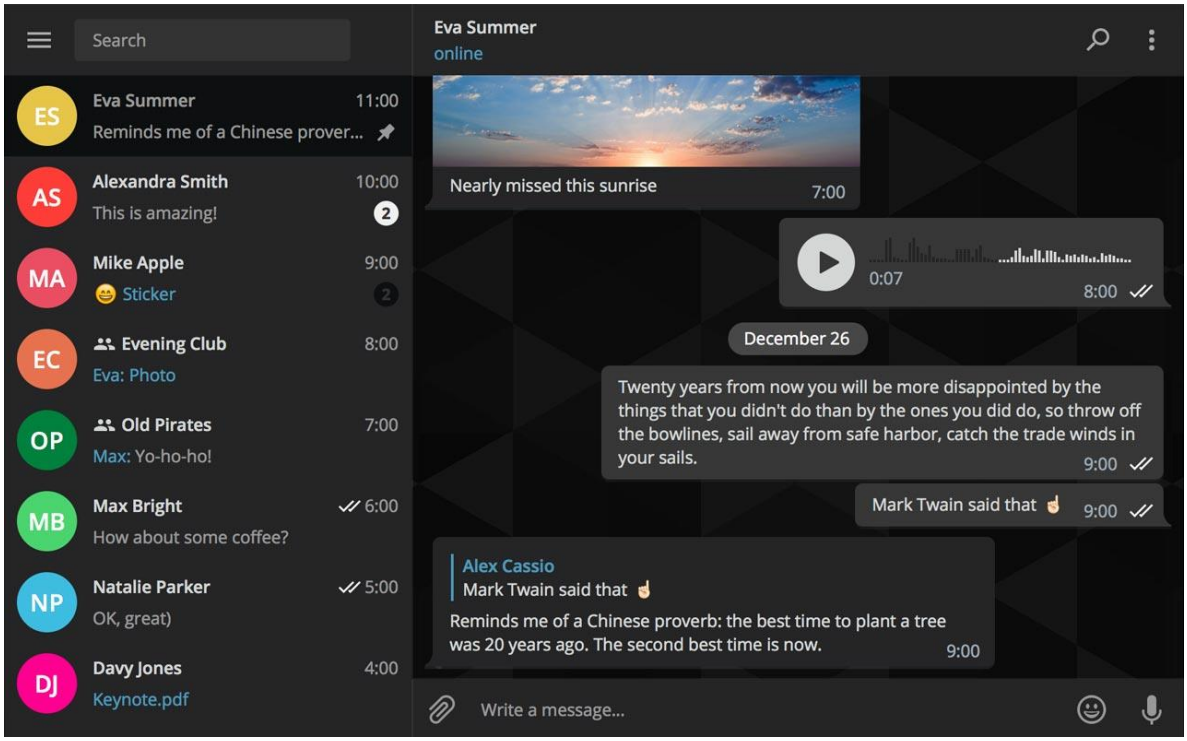

Рис. 1.2. Інтерфейс програми Telegram

Signal є одним з найбезпечніших месенджерів, розроблений організацією Signal Foundation. Він використовує потужні методи шифрування для забезпечення конфіденційності повідомлень та викликів. Signal підтримує обмін текстовими повідомленнями, голосовими та відеовикликами, а також дозволяє надсилати файли.

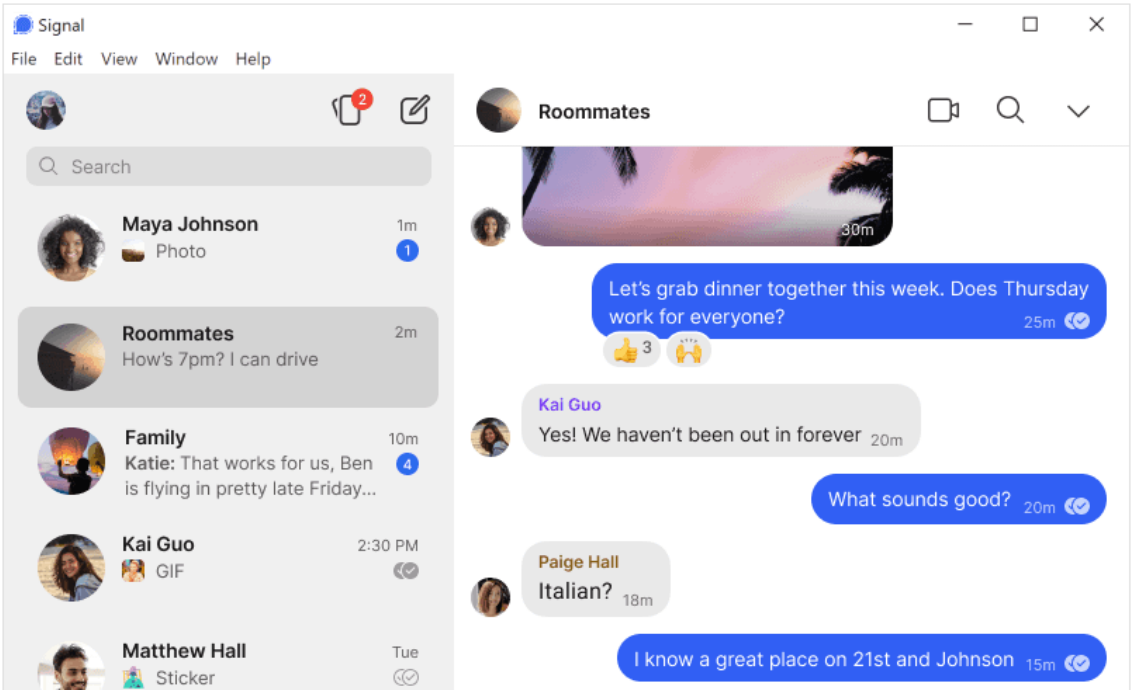

Рис. 1.3. Інтерфейс програми Signal

Microsoft Teams — це корпоративний месенджер, який є частиною пакету Microsoft 365. Він дозволяє командам співпрацювати в режимі реального часу, обмінюватися повідомленнями, файлами, проводити відеоконференції та інтегрувати

Microsoft Teams: Найкращий для корпоративного використання завдяки інтеграції з Office 365 та розширеним функціям для бізнесу але багато функцій можуть бути складними для нових користувачів.

Месенджери грають важливу роль у сучасному світі, забезпечуючи швидку та зручну комунікацію між користувачами. Розробка нового месенджера передбачає врахування багатьох аспектів, включаючи зручність використання, безпеку даних та функціональні можливості, які відповідають потребам сучасних користувачів.

Розробка додатків для Teams включає створення нових додатків або інтеграцію існуючих, використовуючи переваги платформи для потреб бізнесу [3].

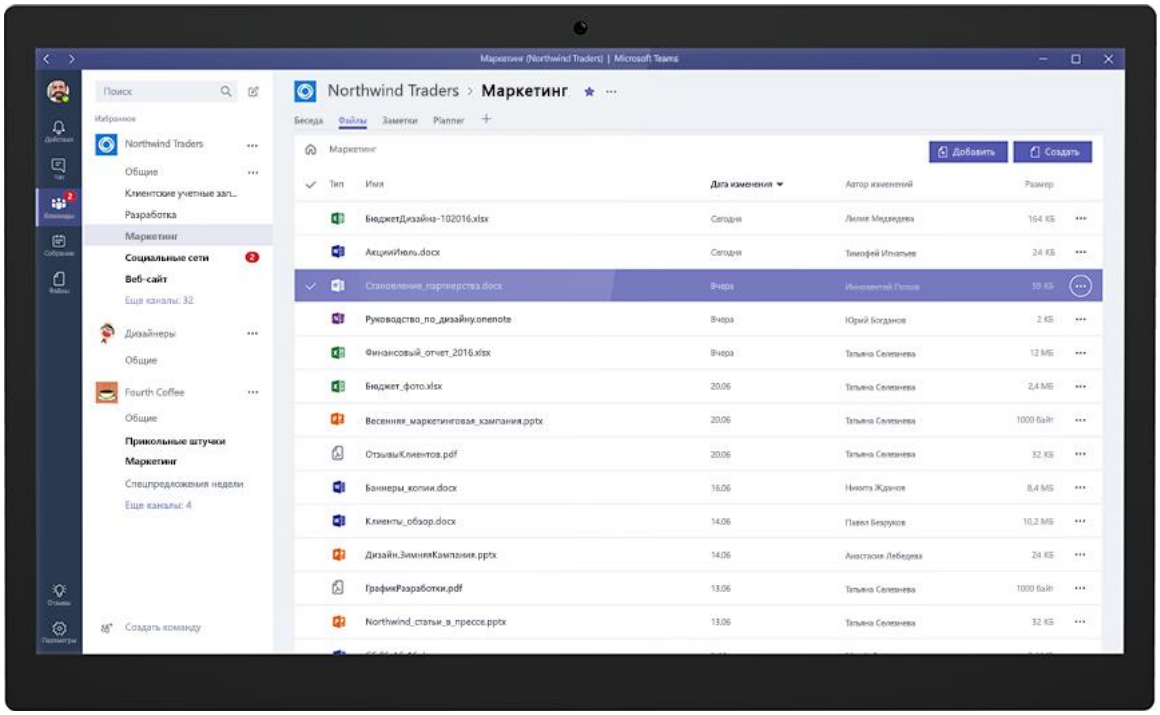

Рис. 1.4. Інтерфейс програми Microsoft Teams

Сучасні месенджери розробляються з використанням різних технологій та мов програмування, таких як JavaScript, Swift, Kotlin, Java, C#, а також фреймворків та бібліотек, таких як React Native, Flutter, Electron та інших.

Кожен з месенджерів має свої переваги та недоліки, і вибір залежить від конкретних потреб користувачів:

WhatsApp: Оптимальний для тих, хто шукає популярну платформу з широким набором функцій і високим рівнем безпеки але Обмеження на розмір файлів: Максимальний розмір відео — 16 МБ, документи — 100 МБ.

Telegram: Підходить для користувачів, які потребують швидкості, великих файлів, та розширених можливостей управління групами але Наскрізне шифрування: Не за замовчуванням для всіх чатів, тільки для секретних чатів.

Signal: Ідеальний вибір для тих, хто ставить безпеку та конфіденційність на перше місце але Функціональність: Менш інтерактивна, ніж у WhatsApp і Telegram, відсутність таких функцій, як статуси

1.2. Призначення розробки та галузь застосування

Розроблюваний месенджер призначений для забезпечення ефективного, безпечного та зручного обміну повідомленнями між користувачами персональних комп'ютерів і мобільних пристроїв. Основні функції месенджера включають:

- Текстовий чат: Можливість обміну миттєвими повідомленнями, з підтримкою різних форматів тексту, включаючи емодзі та інші мультимедійні елементи.

- Обмін файлами: Користувачі можуть надсилати й отримувати файли різних форматів, що полегшує обмін документами, зображеннями, відео та іншими медіа-файлами.

- Групові чати: Створення групових розмов для командної роботи або соціальних груп, що включає можливості керування групами, такі як додавання або видалення учасників, налаштування привілеїв тощо.

- Відео- та аудіо-дзвінки: Підтримка високоякісних відео- та аудіо-дзвінків, що дозволяє користувачам спілкуватися в режимі реального часу незалежно від їхнього місцезнаходження.

- Безпека: Використання сучасних протоколів шифрування для забезпечення приватності та захисту даних користувачів.

- Інтеграція з іншими сервісами: Можливість інтеграції з календарями, планувальниками та іншими інструментами для підвищення продуктивності.

Галузь застосування месенджера охоплює як особисте, так і професійне спілкування. В особистому використанні месенджер допомагає підтримувати зв’язок з родиною та друзями, обмінюватися новинами, організовувати заходи. У професійному середовищі месенджер сприяє підвищенню ефективності внутрішніх комунікацій, полегшує управління проектами, забезпечує зручний і безпечний обмін інформацією між колегами та партнерами.

1.3. Підстава для розробки

В кінці навчання, студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою.

Підставою для розробки кваліфікаційної роботи на тему «Розробка програмного забезпечення месенджера для персональних комп'ютерів» є наказ по Національному технічному університету «Дніпровська політехніка» 469-с від 14.01.2024.

1.4. Постановка завдання

Розробка програмного забезпечення месенджера для персональних комп'ютерів вимагає чіткого визначення завдань, які необхідно вирішити для створення функціонального, безпечного та зручного інструменту для обміну повідомленнями. Постановка завдання включає наступні етапи та підзавдання:

Визначення цільової аудиторії

1. Аналіз потреб користувачів:

- Проведення опитувань та досліджень для визначення основних вимог і побажань користувачів.

- Виявлення ключових сценаріїв використання месенджера у різних сегментах (особисте спілкування, корпоративне середовище, навчальні заклади тощо).

Функціональні можливості:

2. Розробка функціоналу для текстового чату:

- Можливість відправлення та отримання текстових повідомлень у режимі реального часу.

- Підтримка форматування тексту (жирний, курсив, підкреслення).

- Реалізація функції пошуку по історії повідомлень.

3. Обмін мультимедійними даними:

- Підтримка відправлення та отримання файлів (документи, зображення, відео, аудіо).

- Відображення попереднього перегляду файлів (для зображень та відео).

- Забезпечення захисту та зберігання переданих файлів.

4. Групові чати та конференції:

- Створення та управління груповими чатами з можливістю додавання/видалення учасників.

- Підтримка групових аудіо- та відеодзвінків.

- Можливість створення каналів для обговорення певних тем.

5. Система сповіщень:

- Реалізація сповіщень про нові повідомлення, дзвінки та інші події.

- Налаштування користувачем параметрів сповіщень (звук, вібрація, спливаючі вікна).

Безпека та приватність

6. Забезпечення безпеки даних:

- Реалізація наскрізного шифрування для захисту повідомлень від несанкціонованого доступу.

- Використання двофакторної аутентифікації для входу в систему.

- Забезпечення зберігання даних у зашифрованому вигляді на сервері.

7. Приватність користувачів:

- Можливість налаштування приватності (хто може бачити статус онлайн, останнє відвідування тощо).

- Впровадження функції самознищуваних повідомлень.

Інтерфейс користувача

8. Дизайн інтерфейсу:

- Розробка інтуїтивно зрозумілого та зручного інтерфейсу користувача.

- Підтримка адаптивного дизайну для різних розмірів екранів.

- Забезпечення високої продуктивності та швидкого відгуку інтерфейсу.

9. Мультимовність:

- Реалізація підтримки декількох мов інтерфейсу.

- Забезпечення можливості легкого додавання нових мов.

Технічні аспекти

10. Архітектура системи:

- Вибір архітектури клієнт-сервер для забезпечення надійності та масштабованості.

- Використання сучасних технологій та протоколів для передачі даних (WebSocket, HTTP/2).

11. Сумісність:

- Забезпечення сумісності месенджера з основними операційними системами для ПК (Windows, macOS, Linux).

- Підтримка інтеграції з іншими додатками та сервісами (наприклад, календарі, хмарні сховища).

Тестування та впровадження

12. Тестування програмного забезпечення:

- Проведення всебічного тестування (функціональне, навантажувальне, безпекове) для виявлення та усунення помилок.

- Забезпечення бета-тестування за участю реальних користувачів для отримання зворотного зв'язку.

13. Впровадження та підтримка:

- Розгортання серверної інфраструктури для забезпечення стабільної роботи месенджера.

- Налаштування системи моніторингу та підтримки для оперативного реагування на проблеми.

Таким чином, постанова завдання включає всебічний підхід до розробки месенджера, починаючи від аналізу потреб користувачів та закінчуючи

впровадженням і підтримкою готового продукту. Цей комплекс заходів дозволить створити ефективний, безпечний та зручний у використанні месенджер для персональних комп'ютерів, який задовольнить сучасні вимоги користувачів.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

- Підтримка обміну текстовими повідомленнями в режимі реального часу.
- Можливість обміну файлами різних форматів (зображення, документи, відео тощо).
- Функціонал групових чатів та конференцій з можливістю управління доступом.
- Вбудована підтримка відео- та аудіо-дзвінків.
- Реалізація історії чатів з можливістю пошуку та фільтрації повідомлень.
- Наявність системи сповіщень про нові повідомлення та події.

1.5.2. Вимоги до складу та параметрів технічних засобів

- Клієнтська частина повинна підтримувати роботу на сучасних версіях Windows, macOS та Linux.
- Серверна частина повинна бути сумісною з основними серверними операційними системами (Linux, Windows Server).
- Використання серверів з підтримкою високої продуктивності та відмовостійкості.
- Забезпечення мінімальних затримок при передачі даних через використання оптимізованих мережевих протоколів.

1.5.3. Вимоги до інформаційної та програмної сумісності

- Інтеграція з існуючими системами аутентифікації (LDAP, OAuth2).

- Сумісність з основними браузерами для забезпечення веб-доступу до месенджера.

- Використання відкритих стандартів та протоколів для забезпечення сумісності з іншими системами та сервісами.

- Підтримка API для розширення функціональності та інтеграції з сторонніми додатками.

1.5.4. Вимоги до інформаційної та програмної сумісності

Розроблюваний веб-інтерфейс сумісний зі всіма браузерами, наприклад Google Chrome (починаючи з 29 версії), Microsoft Edge (з 12 версії), Safari (починаючи з 9 версії), Firefox (з 28 версії), Opera (з 12.1 версії).

Ці вимоги та завдання забезпечать створення ефективного, безпечного та зручного у використанні месенджера для персональних комп'ютерів, який відповідатиме сучасним потребам користувачів. Крім того, месенджер підтримує адаптивний дизайн, що робить його зручним для використання на різних пристроях, включаючи персональні комп’ютери, планшети та смартфони. Веб-інтерфейс автоматично підлаштовується під розміри екрану, забезпечуючи зручність користування та доступ до всіх функцій системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Функціональне призначення програми

Мета проекту

Чат-додатки в реальному часі призначені для забезпечення оперативного та інтерактивного досвіду, де повідомлення доставляються та відображаються негайно після їх відправлення. Це означає, що користувачі можуть вести розмови та отримувати оновлення в реальному часі без значних затримок [6].

Основні функції

Основні функції месенджера включають:

1. Обмін повідомленнями

- Текстові повідомлення: Відправка та отримання текстових повідомлень в режимі реального часу.

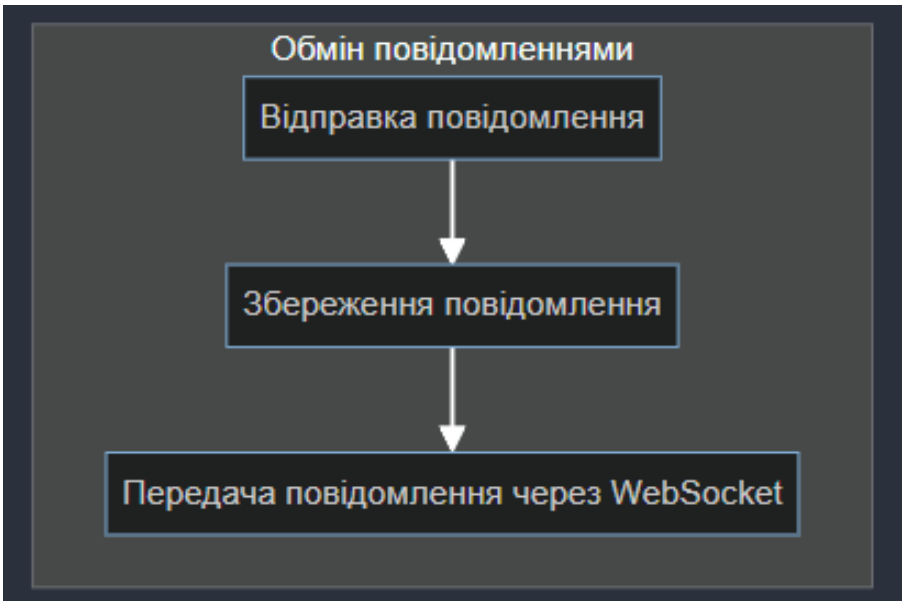

Рис. 2.1. Обмін повідомленнями

2. Групові чати

- Створення груп: Можливість створювати групи для спільного обговорення.

- Управління учасниками: Додавання та видалення учасників групового чату.

- Адміністрування групи: Призначення адміністраторів групи.

3. Система авторизації та реєстрації

- Реєстрація користувачів: Можливість створення нового облікового запису.

- Авторизація: Вхід в систему за допомогою логіна та пароля.

- Відновлення пароля: Можливість відновлення доступу до облікового запису у випадку втрати пароля.

Графічна інформація

Структурна схема системи

1. Клієнтська частина:

- Веб-інтерфейс: Відображення користувацького інтерфейсу.

- Обробка введення користувача: Прийом та обробка введених користувачем даних.

- Відображення повідомлень: Відображення отриманих повідомлень у чатах.

2. Серверна частина:

- Веб-сервер: Обробка запитів від клієнтів (Node.js).

- База даних: Збереження інформації про користувачів, чати та повідомлення (SQLite).

- Обробка запитів: Виконання CRUD операцій з базою даних.

Функціональна схема месенджера

1. Реєстрація та авторизація:

- Форма реєстрації: Заповнення реєстраційної форми користувачем.

- Валідація даних: Перевірка введених даних на сервері.

- Збереження даних: Збереження даних користувача в базу даних.

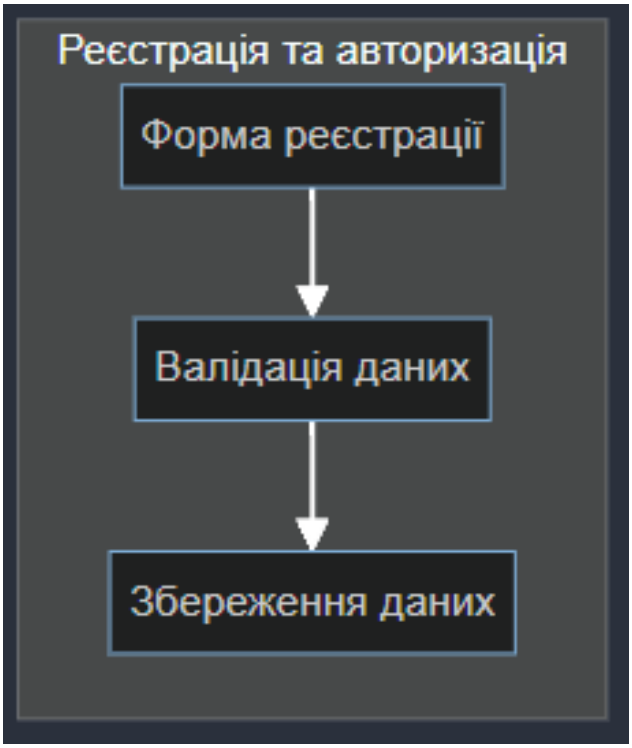

Рис. 2.2. Реєстрація та авторизація

2. Обмін повідомленнями:

- Відправка повідомлення: Введення тексту в полі вводу та натискання кнопки відправки.

- Збереження повідомлення: Збереження повідомлення в базу даних.

- Передача повідомлення: Передача повідомлення іншим користувачам в чаті через WebSocket.

3. Управління чатами:

- Створення чату: Створення нового чату користувачем.

- Додавання учасників: Додавання нових учасників до чату.

- Відображення чатів: Відображення списку доступних чатів та повідомлень в них.

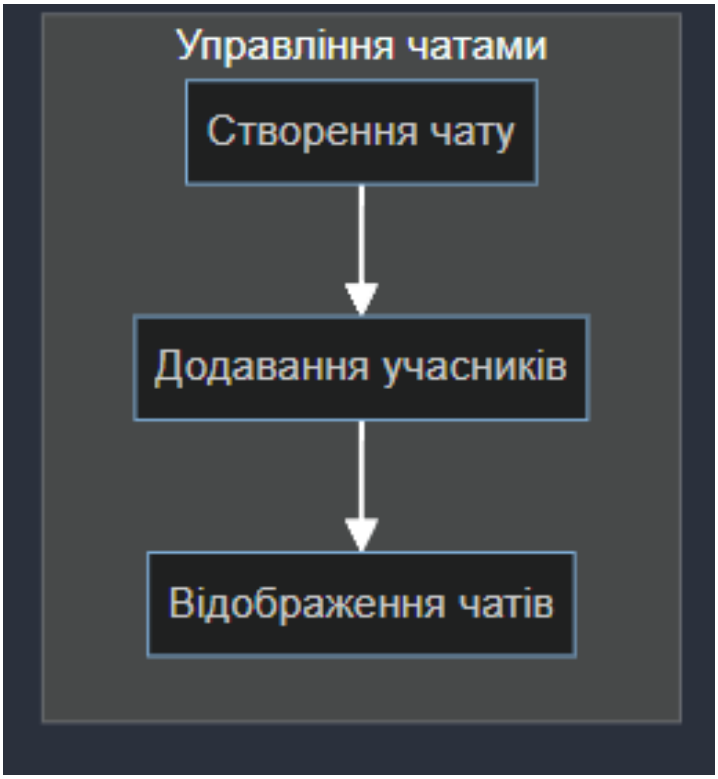

Рис. 2.3. Управління чатами

2.2 Опис застосованих математичних методів

Шифрування та безпека: Симетричне шифрування (AES): використовується для шифрування повідомлень перед їх передачею, що забезпечує конфіденційність даних між користувачами.[11]

Хеш-функції (SHA-256): використовується для хешування паролів користувачів у базі даних, що забезпечує безпеку зберігання паролів та автентифікацію користувачів.[12]

Мережеві протоколи та алгоритми: WebSocket: протокол, який забезпечує двосторонню комунікацію між клієнтом і сервером в реальному часі, використовується для обміну повідомленнями в месенджері.

Управління чергами повідомлень: Черги з пріоритетами: забезпечують обробку повідомлень у порядку їх надходження, що гарантує своєчасну доставку та обробку повідомлень між користувачами.

Обробка даних: Статистичні методи: використовуються для аналізу активності користувачів, вимірювання середнього часу затримки повідомлень та інших метрик, що дозволяє оптимізувати продуктивність системи.

Бази даних: SQLite: використовується для зберігання інформації про користувачів, чати та повідомлення, забезпечуючи надійне та швидке зберігання даних.[5]

2.3 Опис використаної архітектури та шаблонів проектування

Клієнтська частина (Frontend):

JavaScript: JavaScript використовується для динамічного оновлення вмісту веб-сторінок без перезавантаження, обробки подій користувача (наприклад, натискання кнопок, введення тексту), валідації форм, а також для взаємодії з сервером через асинхронні запити (AJAX). Крім того, за допомогою WebSocket JavaScript забезпечує реальне часу спілкування між клієнтом і сервером, що є важливим для чатів та інших інтерактивних застосунків.[20]

HTML

HTML створює основну структуру веб-сторінок, визначаючи такі елементи, як заголовки, абзаци, посилання, зображення, форми та інші компоненти інтерфейсу користувача. Він також використовується для вбудовування мультимедійних елементів, таких як відео та аудіо, і для організації контенту за допомогою таблиць і списків.

Файл HTML містить користувацький інтерфейс для відображення повідомлень чату. В ньому використовуються бібліотеки JavaScript sockjs і stomp.[9]

CSS

CSS відповідає за стилізацію та візуальне оформлення веб-сторінок, включаючи кольори, шрифти, розміри елементів, відступи, вирівнювання та макети. Він дозволяє створювати адаптивний дизайн, що коригує відображення сторінок під різні екрани та пристрої. За допомогою CSS можна додавати анімації та переходи, що підвищують інтерактивність і візуальну привабливість веб-застосунків [7].

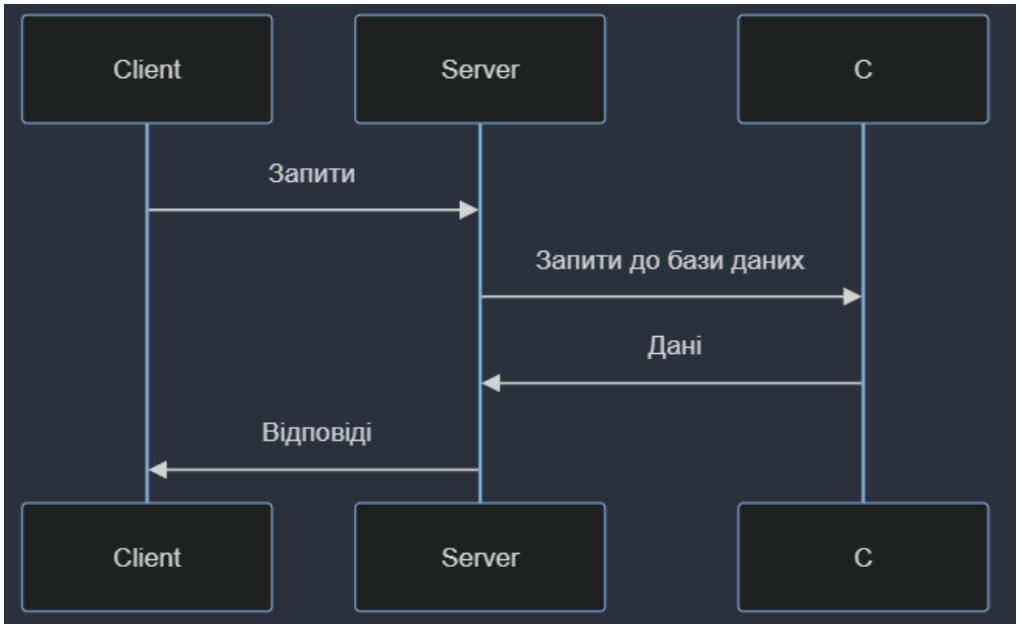

Рис. 2.4. Загальна структура системиня

Серверна частина (Backend):

Node.js

Node.js використовується для створення серверної частини веб-додатків, забезпечуючи асинхронне виконання операцій вводу-виводу, що дозволяє обробляти велику кількість одночасних запитів. Він дозволяє писати серверний код на JavaScript, що полегшує розробку повного стека застосунків.[2]

Якщо ви цікавитесь створенням власного додатку для чату в реальному часі, Node.js є відмінним вибором. Node.js є середовищем виконання JavaScript, яке ідеально підходить для створення масштабованих та ефективних програм в реальному часі.[8]

Express.js

Express.js – це мінімалістичний і гнучкий веб-фреймворк для Node.js, який спрощує створення серверних застосунків і API. Він надає можливості для маршрутизації, обробки запитів та відповідей, управління сесіями та middleware для додаткової обробки даних.[21]

Socket.IO

Socket.IO – це бібліотека, яка дозволяє реалізувати двостороннє реального часу спілкування між клієнтом і сервером. Вона використовується для створення чатів, оновлення даних у реальному часі та інших інтерактивних функцій, що вимагають миттєвого обміну інформацією між користувачами та сервером.

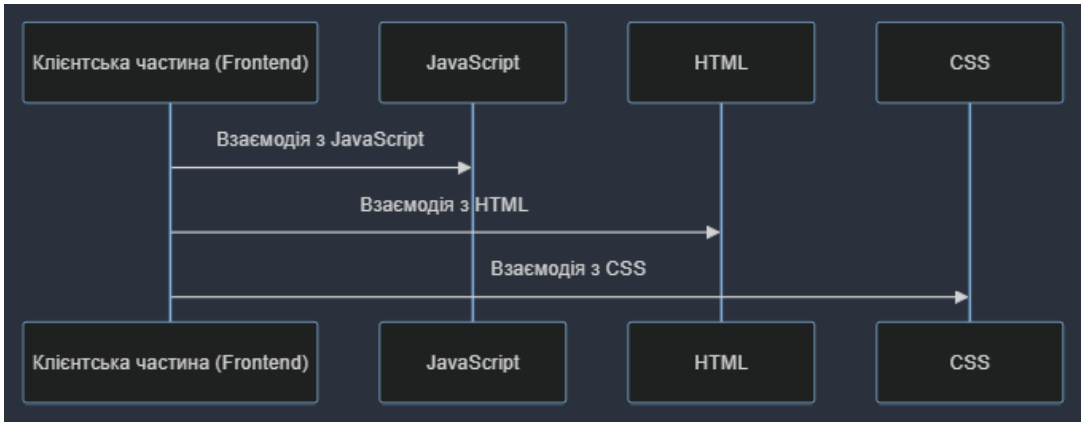

Рис. 2.5. Компоненти клієнтської частини

MongoDB

MongoDB – це NoSQL база даних, яка зберігає дані у форматі документів BSON (Binary JSON). Вона забезпечує гнучкість у зберіганні та обробці

різнорідних і динамічних даних, що підходить для сучасних веб-застосунків з мінливими вимогами до даних.

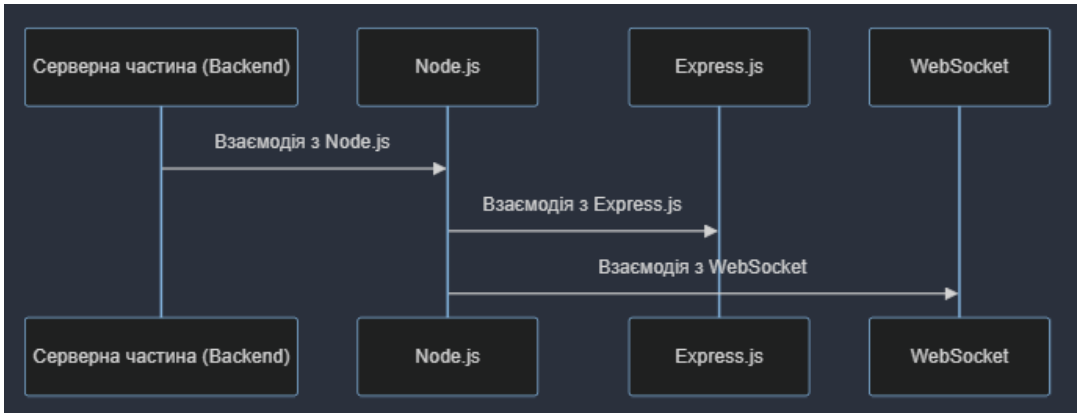

Рис. 2.6. Компоненти серверної частини

Архітектурні підходи: RESTful API: Структурований підхід для обміну даними між клієнтом та сервером через HTTP запити, що забезпечує простоту та ефективність.

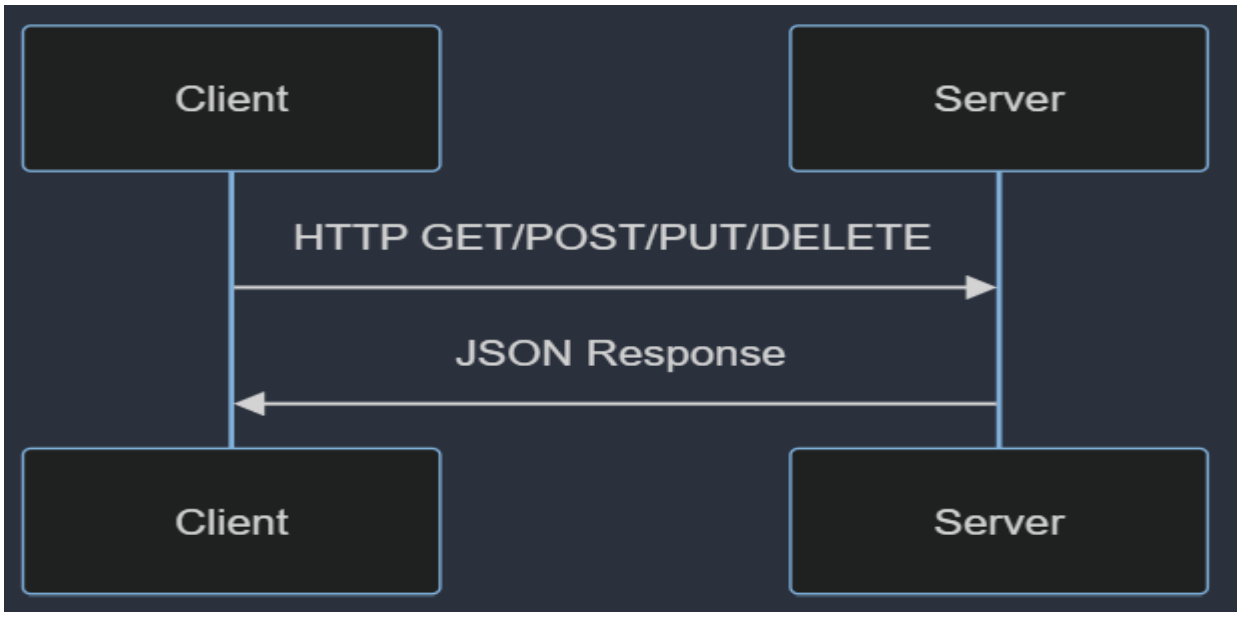

Рис. 2.7. RESTful API

WebSocket: Протокол для встановлення двостороннього зв'язку між клієнтом і сервером у реальному часі, що забезпечує швидкий обмін повідомленнями без затримок.

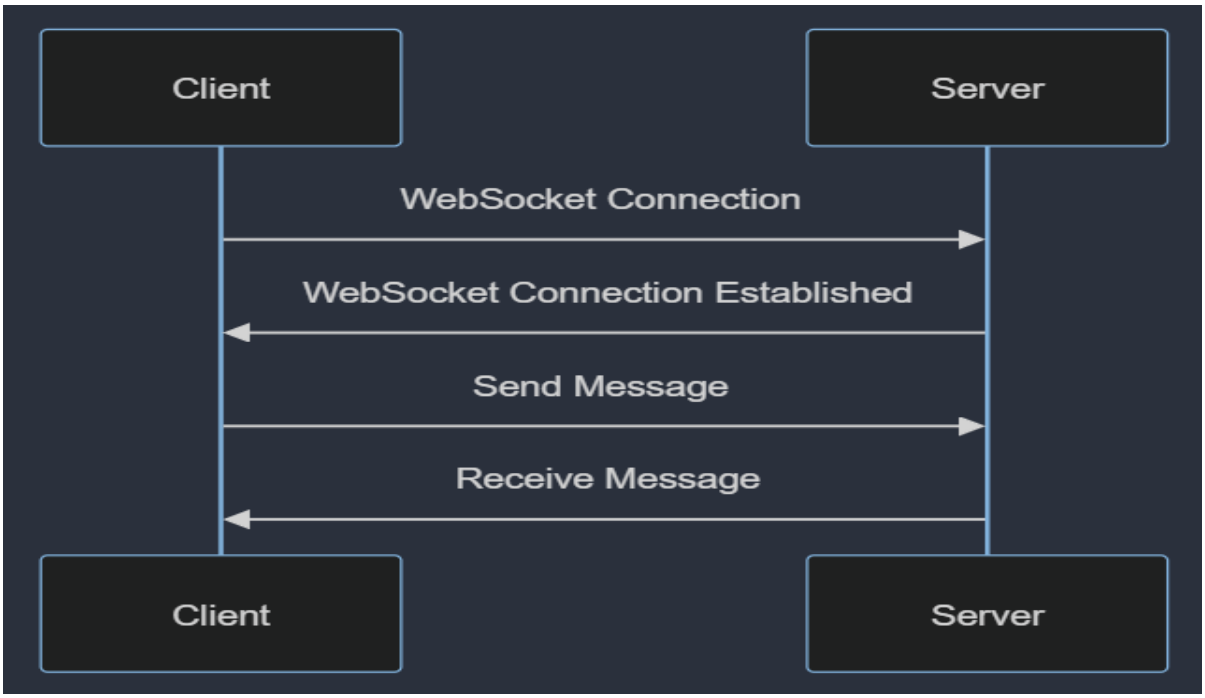

Рис. 2.8. WebSocket

WebSocket працює шляхом спочатку встановлення звичайного HTTP з'єднання з сервером, а потім оновлює його до двонаправленого з'єднання WebSocket, відправляючи заголовок Upgrade.[10]

Шаблони проектування: Model-View-Controller (MVC): Шаблон проектування, що розділяє додаток на три основні компоненти: Model (модель даних), View (представлення інтерфейсу) та Controller (контролер логіки). Це дозволяє спрощувати підтримку та розширення коду.[13]

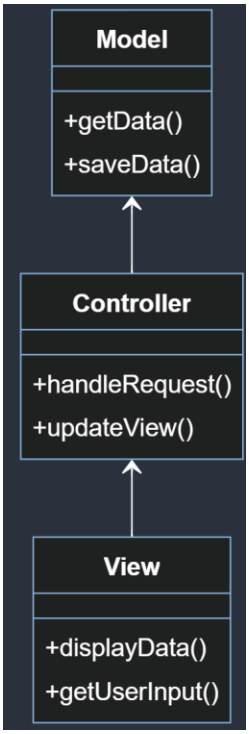

Рис. 2.9. Шаблони проектування

2.4 Опис використаних технологій та мов програмування

Під час розробки месенджера для кваліфікаційної роботи були використані наступні технології та мови програмування:

1. HTML, CSS, JavaScript:
    - HTML використовувався для створення основних структурних компонентів месенджера, таких як форми реєстрації, вхідні форми, чати та вікна повідомлень. Наприклад, створення форми реєстрації:

```
<form id="register-form">
    <input type="text" id="register-username" placeholder="Username">
    <input type="password" id="register-password" placeholder="Password">
    <button id="register-btn">Register</button>
</form>
```

- CSS забезпечував стиль і зовнішній вигляд елементів, що полегшувало користувачам взаємодію з інтерфейсом. Наприклад, стилізація кнопок:

```
button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    cursor: pointer;
    border-radius: 5px;
}
```

- JavaScript використовувався для динамічної інтерактивності сторінок, таких як переключення між формами реєстрації та входу, а також для валідації введених даних. Наприклад, обробник події для переключення на форму входу:

```
const loginLink = document.getElementById('login-link');

loginLink.addEventListener('click', function(event) {
    event.preventDefault();
    registerContainer.style.display = 'none';
    loginContainer.style.display = 'block';
});
```

2. SQLite:
    - SQLite використовувався для створення і керування локальною базою даних, яка зберігала інформацію про користувачів, їх паролі,

повідомлення та інші дані. Наприклад, створення таблиці для користувачів:

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT NOT NULL,
  password TEXT NOT NULL
);
```

3. Розробка та інструменти:
    - Microsoft Visual Studio Code 2022 використовувався для розробки коду, забезпечуючи зручність у редагуванні, налагодженні та відстеженні змін. Наприклад, встановлення розширень для автоматичної перевірки коду.
    - Локальний веб-сервер (XAMPP) забезпечував середовище для запуску месенджера на локальному комп'ютері, включаючи Apache для обробки HTTP запитів і SQLite для зберігання даних. Наприклад, налаштування веб-сервера Apache для розгортання месенджера.

Ці технології і інструменти спільно використовувалися для створення повноцінного месенджера зі зручним веб-інтерфейсом, надійним зберіганням даних і високою ступеню взаємодії з користувачами.

2.5 Опис структури програми та алгоритмів її функціонування

Структура програми

Програма вашого месенджера складається з двох основних частин: клієнтської і серверної.

Клієнтська частина представлена веб-інтерфейсом, який користувач бачить у своєму веб-браузері. Цей інтерфейс відповідає за відображення чату, введення повідомлень та взаємодію з користувачем.

Серверна частина виконується на веб-сервері і відповідає за обробку запитів від клієнтської частини. Вона здійснює обробку повідомлень, зберігає та відправляє їх між користувачами, керує автентифікацією та авторизацією користувачів.

Схематично, взаємодію інтерфейсу з сервером можна показати наступним чином(див. рис. 2.10):

Рис. 2.10. Схема взаємодії інтерфейсу з сервером

Клієнтська частина (Front-end):

- HTML/CSS/JavaScript: Веб-інтерфейс, що відповідає за візуальне представлення месенджера, включаючи відправлення повідомлень, чатів та інші інтерактивні елементи.

Серверна частина (Back-end):

- Web-сервер: Обробляє HTTP-запити від клієнтської частини та відправляє відповіді.

- SQLite: Реляційна база даних для зберігання інформації про користувачів, повідомлення та інші дані.

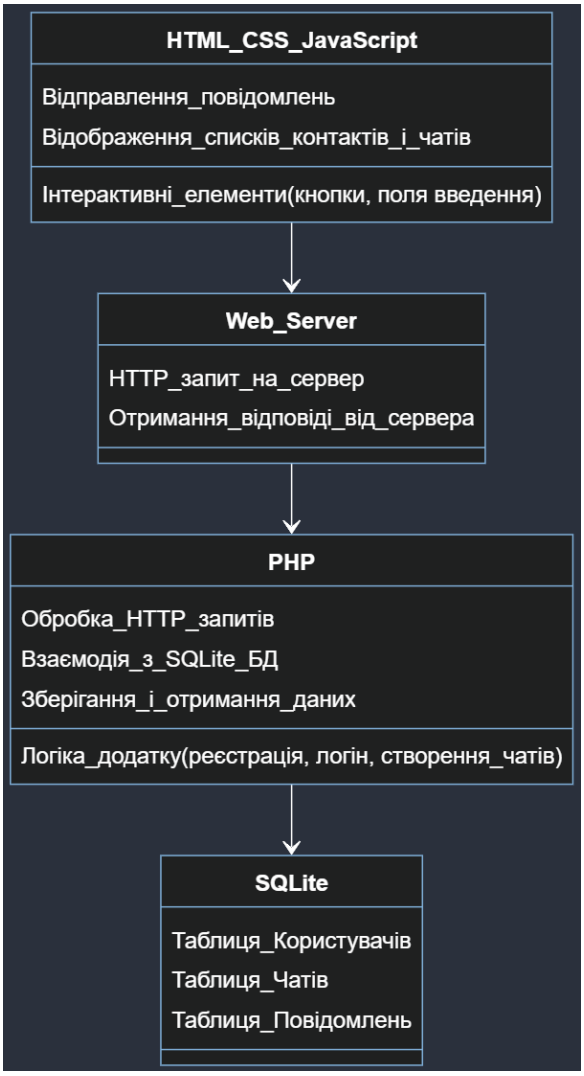

Рис. 2.11. Схема взаємодії між різними компонентами

Алгоритми функціонування

1. Авторизація та реєстрація користувачів:

- Користувач вводить свої дані для входу або реєструється в системі.

- На сторінці авторизації/реєстрації виконується перевірка введених даних.

- При успішній авторизації користувача в системі йому надається доступ до особистого кабінету.

. Нижче наведений приклад:

```
registerButton.addEventListener('click', () => {
    username = registerUsername.value;
    const password = registerPassword.value;
    fetch('/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
    })
    .then(response => response.text())
    .then(message => {
        alert(message);
        registerForm.style.display = 'none';
        loginForm.style.display = 'block';
    })
    .catch(error => {
        console.error('Error registering user:', error);
        alert('Error registering user');
    });    });
```

2. Відправлення та отримання повідомлень:

- Користувач обирає співрозмовника та вводить текст повідомлення.

- JavaScript ініціює асинхронний запит до сервера через AJAX для відправлення повідомлення.

- Серверна сторона обробляє запит, записує повідомлення в базу даних і відсилає підтвердження.

- При наявності нових повідомлень сервер відсилає їх до клієнтської сторони, яка відображає їх у веб-інтерфейсі чата.

3. Обробка помилок та винятків:

- При виникненні помилок під час реєстрації, авторизації, відправлення повідомлення сервер повертає відповідну помилкову інформацію.

- Клієнтська сторона обробляє ці відповіді та відображає користувачеві відповідне повідомлення про помилку.

4. Захист інформації:

- Дані, що передаються між клієнтом і сервером, шифруються за допомогою HTTPS для забезпечення конфіденційності та безпеки.

- Безпека серверної частини забезпечується за допомогою обробки введених даних, перевірки на валідність, використання підготовлених SQL-запитів для запобігання SQL-ін'єкціям та інших технік захисту.

. Нижче наведений приклад:

1. Таблиця `users`:

- `user_id` (INT, AUTO_INCREMENT, PRIMARY KEY) - унікальний ідентифікатор користувача

- `username` (VARCHAR) - ім'я користувача

- `password` (VARCHAR) - хеш пароля користувача

2. Таблиця `messages`:

- `message_id` (INT, AUTO_INCREMENT, PRIMARY KEY) - унікальний ідентифікатор повідомлення

- `sender_id` (INT, FOREIGN KEY) - зовнішній ключ на `user_id` з таблиці `users`, хто відправив повідомлення

- `receiver_id` (INT, FOREIGN KEY) - зовнішній ключ на `user_id` з таблиці `users`, кому призначене повідомлення

- `message_content` (TEXT) - текст повідомлення

- `sent_at` (DATETIME) - час відправлення повідомлення

3. Таблиця `conversations`:

- `conversation_id` (INT, AUTO_INCREMENT, PRIMARY KEY) - унікальний ідентифікатор розмови

- `user1_id` (INT, FOREIGN KEY) - зовнішній ключ на `user_id` з таблиці `users`, перший учасник розмови

- `user2_id` (INT, FOREIGN KEY) - зовнішній ключ на `user_id` з таблиці `users`, другий учасник розмови

2.6. Обґрунтування та організація вхідних та вихідних даних програми

А авторизація — це надання підтвердження особі права на певні дії, доступ до інформації тощо.[14]

1. Авторизація та реєстрація користувачів:

- Вхідні дані: Вхідні дані включають ідентифікатори та конфіденційні дані, необхідні для ідентифікації користувача і встановлення його сесії у системі.

- Вихідні дані: Вихідні дані надають інформацію про статус авторизації та особисті дані користувача, які можуть використовуватися для подальшої персоналізації і взаємодії з додатком..

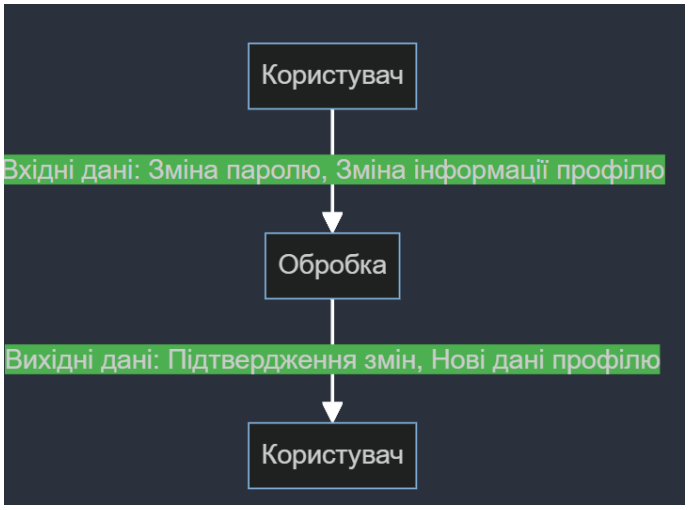

Рис. 2.12. Структурна схема

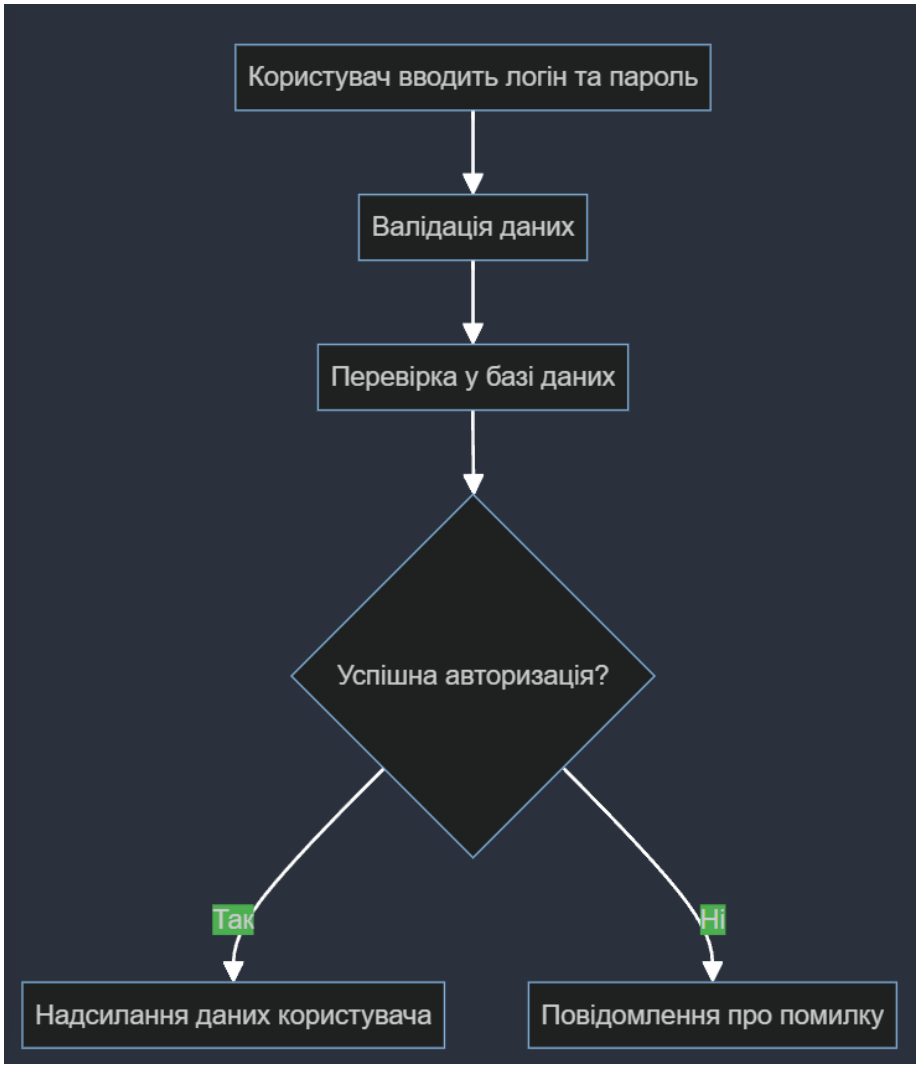

Рис. 2.13. Функціональна схема

2. Обмін повідомленнями:

Рис. 2.14. Структурна схема

- Вхідні дані: ID отримувача, текст повідомлення.
- Вихідні дані: Доставлене повідомлення (дата, час), статус доставки.

Рис. 2.15. Функціональна схема

5. Налаштування профілю:

Рис. 2.16. Структурна схема

- Вхідні дані: Зміна паролю, зміна інформації профілю (наприклад, ім'я, фото профілю).

- Вихідні дані: Підтвердження змін, нові дані профілю.

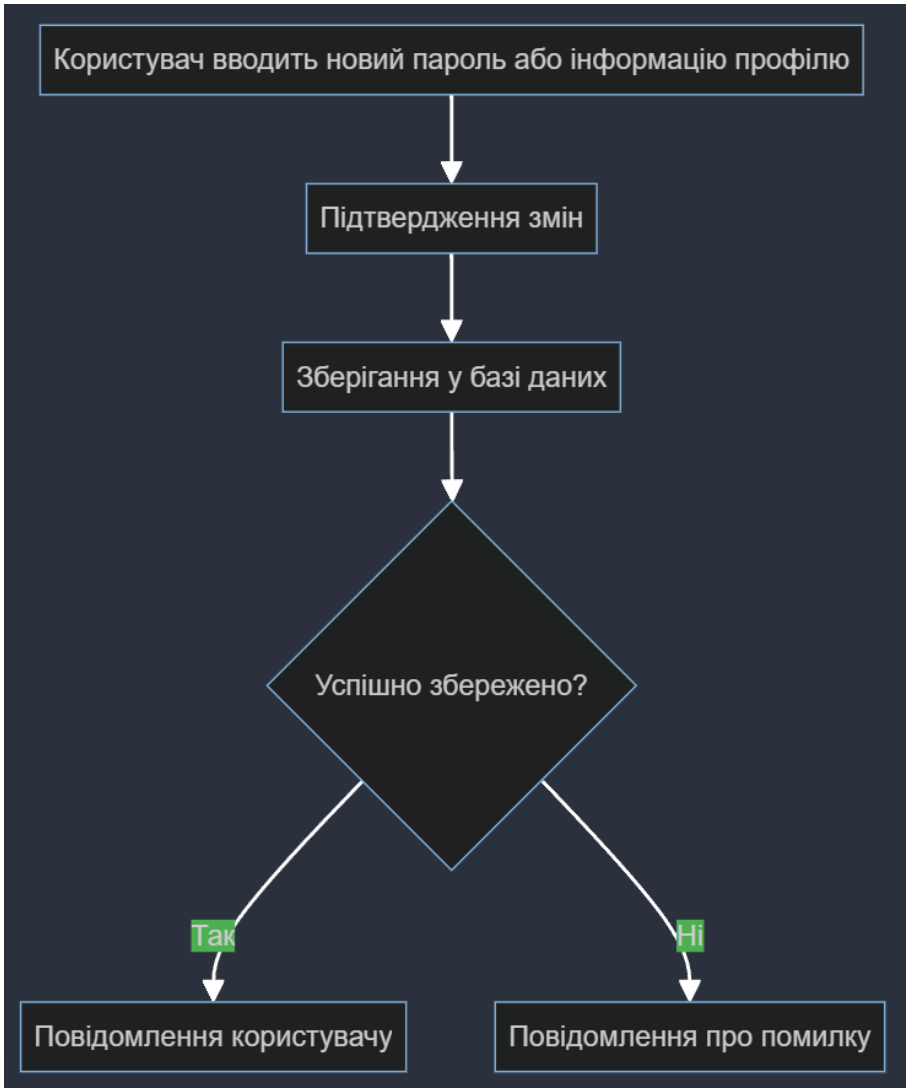

Рис. 2.17. Функціональна схема

Організація вхідних та вихідних даних

1. Обробка вхідних даних:

- Валідація та перевірка на відповідність формату і правилам введення (наприклад, валідація електронної пошти, перевірка на унікальність імені користувача).

- Захист від SQL-ін'єкцій та інших видів атак.

2. Обробка вихідних даних:

- Підготовка відповідей у форматі JSON або іншому, залежно від потреби.

- Відображення інформації користувачам у зручному форматі (наприклад, списки повідомлень або контактів).

3. Використання бази даних:

- Зберігання даних про користувачів, повідомлення, контакти тощо.

- Виконання операцій з базою даних через запити (SELECT, INSERT, UPDATE, DELETE) для отримання та зберігання інформації.

4. Захист даних:

- Використання шифрування для зберігання паролів користувачів.

- Захист особистої інформації користувачів від несанкціонованого доступу.

2.7 Опис розробленого програмного продукту

2.7.1 Використані технічні засоби

Дане програмне забезпечення є веб-інтерфейсом, тому переважно виконується у веб-браузері на локальному сервері, і для його функціонування вистачить мінімальних ресурсів ПК. Основні технічні засоби для роботи ПЗ:

– Персональний комп’ютер або мобільний пристрій;

– Будь-який веб-браузер із списку: Google Chrome, Microsoft Edge, Safari

– Стабільне підключення до мережі Internet;

2.7.2 Використані програмні засоби

HTML, CSS, JavaScript:

- HTML (HyperText Markup Language) використовується для створення структури веб-сторінок.
- CSS (Cascading Style Sheets) використовується для стилізації і форматування веб-сторінок, зокрема для надання їм вигляду і відображення.
- JavaScript використовується для динамічної інтерактивності на сторінках, маніпуляції DOM, взаємодії з користувачем та інших функцій, що покращують веб-додаток.

Node.js:

- Node.js використовується для реалізації серверної частини додатка. Він використовує JavaScript-движок V8 від Google та надає можливості для створення високопродуктивних і масштабованих мережевих застосунків.

Express.js:

- Express.js використовується як фреймворк для Node.js для зручного створення веб-додатків та API. Він допомагає управляти маршрутизацією, обробкою запитів та відповідей, аутентифікацією та іншими аспектами серверної логіки.

MongoDB:

- MongoDB використовується як NoSQL база даних для зберігання і управління даними, пов'язаними з користувачами, повідомленнями, контактами та іншою інформацією, необхідною для функціонування месенджера.[15]

Socket.IO:

- Socket.IO використовується для реалізації багаторівневого зв'язку між клієнтською і серверною частинами додатка за допомогою WebSocket. Це дозволяє надсилати та отримувати повідомлення в реальному часі без необхідності перезавантаження сторінки.[16]

- У моїй роботі я використовував HTML, CSS, та JavaScript для створення інтерактивного користувацького інтерфейсу на стороні клієнта. HTML використовувався для створення структури веб-сторінок, CSS — для їх стилізації, а JavaScript — для динамічної інтерактивності, такої як валідація форм, обробка подій введення, та взаємодія з сервером через AJAX запити.
- На стороні сервера я використовував Node.js з Express.js. Node.js дозволяв створювати серверну частину додатка, обробляти HTTP-запити від клієнтів, та взаємодіяти з базою даних MongoDB, що була обрана для зберігання інформації про користувачів, повідомлення, та інші дані, необхідні для роботи месенджера.
- Socket.IO використовувався для забезпечення реального часу обміну повідомленнями між клієнтами та сервером через WebSocket, що забезпечувало миттєву доставку повідомлень без необхідності перезавантаження сторінки.
- Ці технології були інтегровані для створення функціонального та ефективного месенджера зі зручним інтерфейсом та можливістю обміну повідомленнями в реальному часі.

2.7.3 Виклик та завантаження програми

Для виклику та завантаження програми необхідно:

– Завантажити та помістити папку з ПЗ

– Запустити консоль

– За допомогою команди cd перейти до розташування файлу server.js

– Запустити сервер командою node server.js
– Відкрити браузер
– Ввести у строці для URL http://localhost:3000
– Запустити початкову сторінку Log In

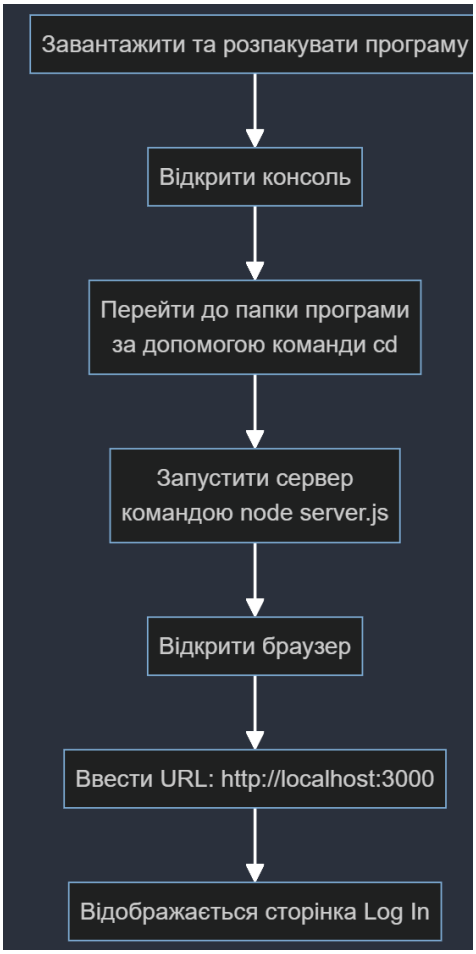

Рис. 2.18. Функціональна схема

2.7.4 Опис інтерфейсу користувача

Після запуску інтерфейсу відкривається сторінка авторизації (див. рис. 2.3). Якщо користувач вже зареєстрований, він може ввести свій логін та пароль, і натиснути кнопку «Login» (див. рис. 2.19)

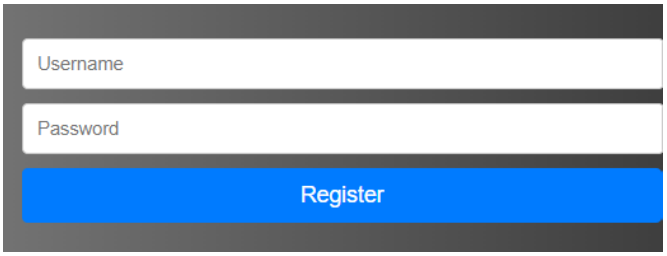

Рис. 2.19. Сторінка реєстрації

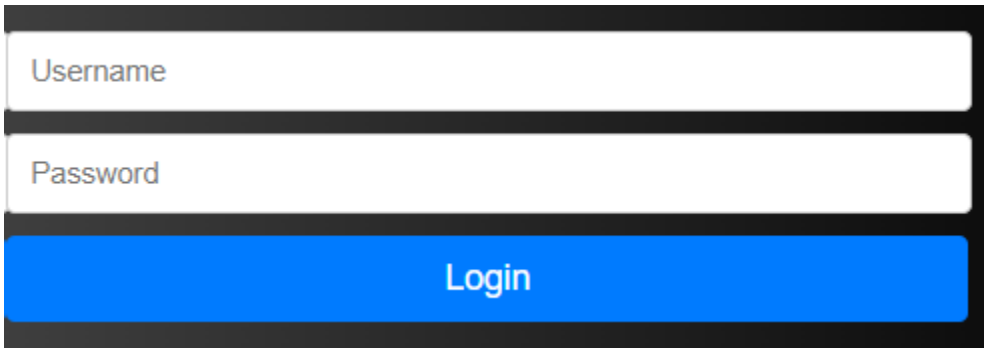

Рис. 2.20. Сторінка авторизації

При неправильно введеному логіні або паролі у формі виводиться певний надпис з помилкою, яку допустив користувач (див. рис. 2.16).

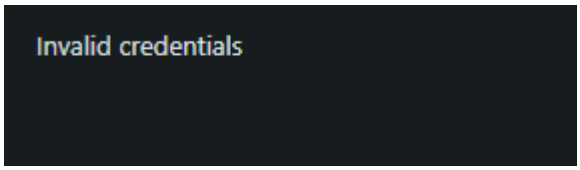

Рис. 2.21. Помилка при введенні неправильного логіну або паролю

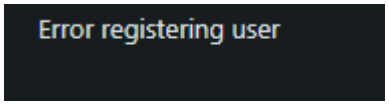

Рис. 2.22. Помилка при існуванні користувача з таким данними

Після того, як користувач успішно авторизується, він буде автоматично перенаправлений на сторінку «Chat», яка є центральним пунктом для обміну повідомленнями в системі (див. рис. 2.18).

Навігаційна панель

Зліва розташована навігаційна панель, яка дозволяє користувачеві додавати нові чати, а також кнопка для видалення існуючих.

Головний блок чату

У верхній частині головного блоку відображається поточний чат. Тут користувач може бачити всі повідомлення, що надіслані та отримані в обраному чаті. Повідомлення відображаються у хронологічному порядку з індикатором часу для кожного повідомлення.

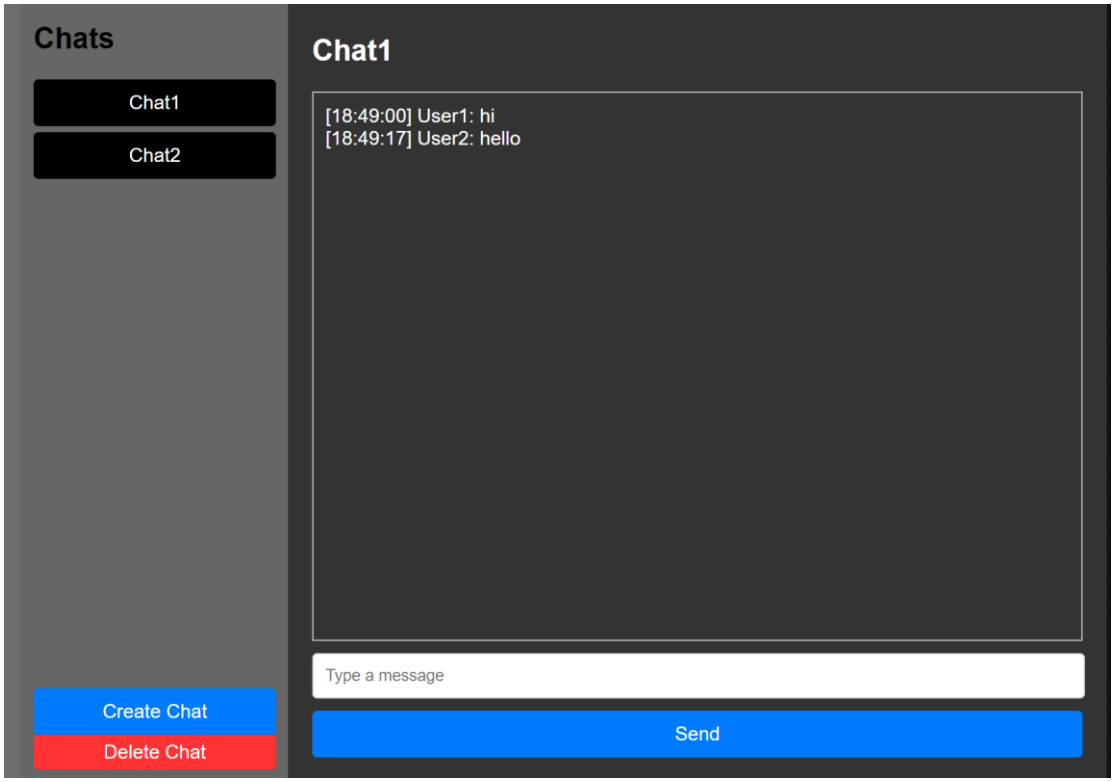

Рис. 2.23. Сторінка «Chat»

Введення повідомлення

У нижній частині головного блоку знаходиться поле для введення повідомлення та кнопка відправки. Користувач може ввести текст у поле для введення повідомлення і натиснути кнопку відправки. Всі надіслані повідомлення миттєво відображаються в чаті.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. Передбачуване число операторів програми – 3000.
2. Коефіцієнт складності програми – 1,3.
3. Коефіцієнт корекції програми в ході її розробки – 0,05.Годинна заробітна плата програміста – 250 грн/год (сучасні дані, ураховуючи інфляцію і ринкові зміни).
4. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2.
5. Коефіцієнт кваліфікації програміста, обумовлений від стажу– 1,4.
6. Вартість машино-години ЕОМ – 0,8 грн/год.

Примітка:

За статистикою на середину 2023 року, середня зарплата Junior Front End розробника в Україні коливається в діапазоні від 700$ до 1400$ в місяць. Знаючи ці дані, можна розрахувати середню місячну заробітну плату, яка становить приблизно 1050$ у місяць. Ураховуючи офіційний курс валют

Національного банку України на початок травня 2023 року, де один американський долар еквівалентний 36,56 гривням, середня місячна зарплата Junior Front End розробника в Україні складає 38388 грн. Якщо припустити стандартний робочий графік з 176 годинами на місяць, то зарплата за годину становитиме приблизно 218 гривень.

Проте, враховуючи поточну інфляцію і ринкові зміни, для розрахунків ми використовуватимемо середню погодинну зарплату програміста, яка становить 250 грн/год.

Для розробки цього проекту використовувався потужний ПК та монітор 27 дюймів. Це потребує врахування витрат на електроенергію та інтернет від провайдера «Київстар».

Типовий персональний комп'ютер споживає 65-250 Вт за годину, монітор – 20-80 Вт. Враховуючи ці характеристики, споживання електроенергії за годину становить приблизно 280 Вт*год. За даними постачальника електроенергії Yasno, тариф для населення на кінець травня становить* 1,44 грн/кВт*год*. Отже, вартість електроенергії для ПК та монітору становить 0,28 * 1,44 = 0,4 грн/год.

Домашнє інтернет-підключення від «Київстар» оплачується за тарифом «Все разом Домашній», вартість якого становить 300 грн на місяць, що дорівнює приблизно 0,4 грн за годину користування. Таким чином, загальна вартість машино-години ЕОМ становить 0,8 грн/год.

Джерела даних:

- Платформа найму "Djinni" для зарплат програмістів.
- Національний банк України для курсу валют.
- Постачальник електроенергії Yasno для тарифів на електроенергію.
- Провайдер "Київстар" для тарифів на інтернет.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі;

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Розрахунок умовного числа операторів (Q):

$$Q = q \cdot C \cdot (1 + p) \quad (3.2)$$

де q - передбачуване число операторів

C - коефіцієнт складності програми

p - коефіцієнт корекції програми в ході її розробки

Підставляємо значення**:**

$$Q=C\cdot q\cdot(1+p)=1,3\cdot 3000\cdot(1+0,05)=1,3\cdot 3000\cdot 1,05=4095$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин} \quad (3.3)$$

Де:

B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста

Підставляємо значення:

$$t_u = 4914/119 = 41,31 \text{ людино-годин}$$

Витрати на розробку блок-схеми алгоритму (ta):

За формулою (3.4):

$$t_a = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин} \qquad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставляємо значення:

$$t_a = 4095/35 = 117 \text{ людино-годин}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин} \qquad (3.5)$$

$$t_n = 4095/33{,}6 = 121{,}88 \text{ людино-годин}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин} \qquad (3.6)$$

$$t_{отл} = 4095/7 = 585 \text{ людино-годин}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}\ \text{, людино-годин} \qquad (3.7)$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин} \quad (3.8)$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др} \text{ , людино-годин.} \quad (3.9)$$

Підставляючи відповідні значення, отримаємо:

$t_{др}$ = 4095/28=146,25 людино-годи

$t_{до}$ =0,75· $t_{др}$ =0,75·146,25=109,69 людино-годин

$t_{д}$ = $t_{др}$ + $t_{до}$ =146,25+109,69=255,94 людино-годин

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t=50+41,31+117+121,88+585+255,94=1171,13 людино-годин

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн} \quad (3.10)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн} \quad (3.11)$$

де: t - загальна трудомісткість, людино-годин;

$C_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 250 грн / год, отримуємо:

$$З_{ЗП} = t \cdot C_{ПР} = 1171{,}13 \cdot 250 = 292782{,}5 \text{ грн}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн}, \quad (3.12)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год.

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = = 585 \cdot 0{,}8 = 468 \text{ грн}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 292782{,}5 + 468 = 293250{,}5 \text{ грн}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \quad (3.13)$$

де B_k - число виконавців;

F_p - місячний фонд робочого часу.

Очікуваний період створення ПЗ:

$$T = 1171{,}13/1 \cdot 176 = 6{,}65 \text{ місяців}$$

Висновок: Розроблений інтерфейс є прототипом мессенджера для ефективного спілкування користувачів. Його головна мета – забезпечити зручний та швидкий обмін повідомленнями між користувачами. Вартість розробки цього програмного забезпечення становить 293250,5 грн. Очікується, що процес розробки триватиме приблизно 6,65 місяців, з огляду на складність поставлених задач та значний обсяг роботи.

ВИСНОВКИ

У рамках даної кваліфікаційної роботи був розроблений веб-мессенджер згідно з поставленим завданням. Основна мета цього мессенджера полягає у забезпеченні зручної та ефективної комунікації між користувачами в реальному часі.

Цей мессенджер має практичне застосування, оскільки надає користувачам можливість реєструватися, авторизуватися та обмінюватися повідомленнями у реальному часі. Інтерфейс мессенджера дозволяє легко відправляти та отримувати повідомлення, переглядати історію чатів та управляти контактами.

Структура розробленого програмного забезпечення базується на клієнт-серверній архітектурі, де веб-інтерфейс виступає як клієнт, а локальний веб-сервер XAMPP виконує роль сервера. Взаємодія між клієнтом і сервером здійснюється за допомогою WebSocket для обміну повідомленнями в реальному часі. Була розроблена база даних SQLite, яка забезпечує зберігання інформації про користувачів та їхні повідомлення. Крім того, на етапі розробки було створено маршрут користувача, що охоплює процеси реєстрації, авторизації та використання мессенджера.

Для розробки даного мессенджера використовувалися мови розмітки HTML, мова стилів CSS, мова програмування JavaScript для фронтенду, а також Java для бекенду. Інструментом для розробки слугувала робоча середа Microsoft Visual Studio Code 2022. Веб-інтерфейс запускається на базі локального веб-серверу, який надає програмне забезпечення XAMPP.

У процесі виконання кваліфікаційної роботи було визначено трудомісткість розробленої системи, а з використанням середньої зарплати розробника була розрахована вартість роботи, яка склала 293250,5гривень. Також було оцінено час, необхідний для розробки веб-мессенджера, який склав 1369,64 людино-годин, що приблизно відповідає 6,65 місяцям роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GeeksforGeeks. (n.d.). How to create a chat application. Retrieved July 1, 2024, from https://www.geeksforgeeks.org/online-chat-application-project-in-software-development/

2. dan-it. (n.d.). Що це таке Node JS простими словами. Retrieved July 1, 2024, from https://dan-it.com.ua/blog/chto-jeto-takoe-node-js-prostymi-slovami/

3. Microsoft. (n.d.). Microsoft Teams app development. Retrieved July 1, 2024, from https://learn.microsoft.com/en-us/microsoftteams/platform/overview

4. vseosvita. (n.d.). Що таке месенджери, які у них можливості . Retrieved July 1, 2024, from https://vseosvita.ua/library/so-take-mesendzeri-aki-u-nih-mozlivosti-top-popularnih-mesendzeriv-281047.html

5. freehost. (n.d.). що таке sqlite. Retrieved July 1, 2024, from https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/

6. TutsMake. (n.d.). Build a simple chat app using JavaScript. Retrieved July 1, 2024, from https://www.geeksforgeeks.org/real-time-chat-application-in-javascript/

7. css.in. (n.d.). Що таке CSS. Retrieved July 1, 2024, from https://css.in.ua/article/shcho-take-html_10

8. FreeCodeCamp. (n.d.). Building a real-time chat app with Node.js, Express, and Socket.io. Retrieved July 1, 2024, from https://sstechsolutions.medium.com/how-to-build-a-real-time-chat-app-with-node-js-sstech-system-8e65ec40e6fd

9. CalliCoder. (n.d.). Creating a chat app using Spring Boot and WebSocket. Retrieved July 1, 2024, from https://www.callicoder.com/spring-boot-websocket-chat-example/

10. Scaledrone. (n.d.). Android chat tutorial: Building a realtime messaging app. Retrieved July 1, 2024, from https://www.scaledrone.com/blog/android-chat-tutorial/

11. iitd. (n.d.). Шифрування та захист баз дани. Retrieved July 1, 2024, from https://iitd.com.ua/shifruvannja-ta-zahist-baz-danih/

12. blog.whitebit. (n.d.). Що таке хеш у блокчейні?. Retrieved July 1, 2024, from https://blog.whitebit.com/uk/what-is-a-hash-in-blockchain/

13. javarush. (n.d.). Ознайомлення з патерном MVC . Retrieved July 1, 2024, from https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller

14. cityhost. (n.d.). Як верифікувати користувача на сайті. Retrieved July 1, 2024, from https://cityhost.ua/uk/blog/yak-verifikuvati-koristuvacha-na-sayti-dzvinki-sms-elektronna-poshta.html

15. mongodb. (n.d.). MongoDB Architecture Guide. Retrieved July 1, 2024, from https://www.mongodb.com/de-de/lp/resources/products/fundamentals/mongodb-architecture-guide?utm_content=rlsapostreg&utm_source=google&utm_campaign=search_gs_pl_evergreen_atlas_general_retarget-brand-postreg_gic-null_emea-all_ps-all_desktop_eng_lead&utm_term=&utm_medium=cpc_paid_search&utm_ad=&utm_ad_campaign_id=14412646473&adgroup=131761130532&cq_cmp=14412646473&gad_source=1&gclid=CjwKCAjwyo60BhBiEiwAHmVLJW_zeWb0fdviAMXHVq4IoDYjLBxkYf92iVam6Rnxtg1s119qGOc8JhoCG0IQAvD_BwE

16. TutorialsPoint. (n.d.). Socket.IO chat application. Retrieved July 1, 2024, from https://www.tutorialspoint.com/socket.io/socket.io_chat_application.htm

17. Code Wall. (n.d.). Real-time chat application using Node.js and Socket.IO. Retrieved July 1, 2024, from https://socket.io/get-started/chat/

18. FreeCodeCamp. (n.d.). Build a realtime chat app with React, Express, Socket.io, and HarperDB. Retrieved July 1, 2024, from https://www.freecodecamp.org/news/build-a-realtime-chat-app-with-react-express-socketio-and-harperdb/

19. GeeksforGeeks. (n.d.). Real-time chat application with Firebase. Retrieved July 1, 2024, from https://www.geeksforgeeks.org/

20. goit.global. (n.d.). Що таке JavaScript. Retrieved July 1, 2024, from https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/

21. expressjs. (n.d.). Expres. Retrieved July 1, 2024, from https://expressjs.com/uk/

Додаток А

ЛІСТИНГ ПРОГРАМИ

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Messenger</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="auto">
        <div id="register-form">
            <input type="text" id="register-username" placeholder="Username">
            <input type="password" id="register-password" placeholder="Password">
            <button id="register-btn">Register</button>
        </div>
        <div id="login-form">
            <input type="text" id="login-username" placeholder="Username">
            <input type="password" id="login-password" placeholder="Password">
            <button id="login-btn">Login</button>
        </div>
    </div>
    <div id="chat-container" style="display: none;">
        <div id="sidebar">
            <h2>Chats</h2>
            <div id="chat-list"></div>
            <button id="create-chat-btn">Create Chat</button>
            <button id="delete-chat-btn">Delete Chat</button>
            <div id="create-chat-form" style="display: none;">
                <input type="text" id="new-chat-name" placeholder="Chat Name">
                <button id="create-chat-submit">Create</button>
            </div>
        </div>
        <div id="main">
```

```
            <h2 id="current-chat"></h2>
            <div id="messages"></div>
            <input type="text" id="message-input" placeholder="Type a message">
            <button id="send-btn">Send</button>
        </div>
    </div>
    <script src="/socket.io/socket.io.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

style.css

```
body {
    display: flex;
    justify-content: center;
    height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
    background: linear-gradient(270deg, black, gray);
}
#auto{
    display: flex;
    justify-content: center;
    height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
    align-items: center;
}
#register-container, #login-container {
    background-color: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}
#register-form h2, #login-form h2 {
    margin-bottom: 20px;
```

```
}

input[type="text"], input[type="password"] {
    width: calc(100% - 20px);
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    width: 100%;
    padding: 10px;
    background-color: #007bff;
    border: none;
    color: white;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}

button:hover {
    background-color: #0056b3;
}

p {
    margin-top: 10px;
}

a {
    color: #007bff;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}
```

```
#chat-container {
    padding: 0 15%;
    display: flex;
    flex: 1;
}

#sidebar {
    width: 200px;
    background-color: #666666;
    padding: 10px;
    box-shadow: 2px 0px 5px rgba(0,0,0,0.1);
    display: flex;
    flex-direction: column;
}

#main {
    flex: 1;
    padding: 20px;
    display: flex;
    flex-direction: column;
    background-color:#333333;
    color: white;
}

#messages {
    flex: 1;
    overflow-y: auto;
    border: 1px solid #ccc;
    padding: 10px;
    margin-bottom: 10px;
}

#message-input {
    margin-bottom: 10px;
}
```

```
#chat-list button {
    display: block;
    width: 100%;
    margin-bottom: 5px;
    background-color: black;
}

#create-chat-form {
    margin-top: 10px;
}

#chat-list {
    flex: 1;
    overflow-y: auto;
}

#delete-chat-btn {
    margin-top: auto;
    background-color: #ff3333;
    color: white;
    border: none;
    padding: 5px 10px;
    cursor: pointer;
    width: 100%;
}

#delete-chat-btn:hover {
    background-color: #cc0000;
}
```

server.js

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const bcrypt = require('bcryptjs');
const db = require('./database');
```

```
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
const users = {};

app.use(express.static('public'));
app.use(express.json());

const saltRounds = 10;
app.post('/register', (req, res) => {
    const { username, password } = req.body;
    bcrypt.hash(password, saltRounds, (err, hash) => {
        if (err) return res.status(500).send('Error hashing password');
        db.run('INSERT INTO users (username, password) VALUES (?, ?)', [username, hash], (err) => {
            if (err) return res.status(500).send('Error registering user');
            res.status(201).send('User registered');
        });
    });
});

app.post('/login', (req, res) => {
    const { username, password } = req.body;
    db.get('SELECT * FROM users WHERE username = ?', [username], (err, user) => {
        if (err || !user) return res.status(400).send('Invalid credentials');
        bcrypt.compare(password, user.password, (err, result) => {
            if (err || !result) return res.status(400).send('Invalid credentials');
            res.status(200).send('Login successful');
        });
    });
});

app.post('/create_chat', (req, res) => {
    const { chatName } = req.body;
    db.run('INSERT INTO chats (chat_name) VALUES (?)', [chatName], (err) => {
        if (err) return res.status(500).send('Error creating chat');
        res.status(201).send('Chat created');
    });
```

```
});

app.delete('/delete_chat/:chatName', (req, res) => {
    const { chatName } = req.params;
    db.get('SELECT id FROM chats WHERE chat_name = ?', [chatName], (err, chatData) => {
        if (err || !chatData) return res.status(500).send('Error deleting chat');
        db.run('DELETE FROM chats WHERE id = ?', [chatData.id], (err) => {
            if (err) return res.status(500).send('Error deleting chat');
            db.run('DELETE FROM messages WHERE chat_id = ?', [chatData.id], (err) => {
                if (err) return res.status(500).send('Error deleting messages');
                io.emit('chatDeleted', chatName);
                res.status(200).send('Chat deleted');
            });
        });
    });
});

io.on('connection', (socket) => {
    console.log('User connected');

    socket.on('joinChat', ({ chat, username }) => {
        socket.join(chat);
        if (!users[socket.id]) {
            users[socket.id] = { username, currentChat: chat };
        } else {
            users[socket.id].currentChat = chat;
        }
        sendMessagesToClient(chat);
        sendChatListToClient();
    });

    socket.on('leaveChat', (chat) => {
        socket.leave(chat);
    });

    socket.on('sendMessage', ({ chat, sender, content }) => {
        const timestamp = new Date().toLocaleTimeString();
```

```
            db.run('INSERT INTO messages (chat_id, sender, content, timestamp) VALUES ((SELECT id FROM chats WHERE chat_name = ?), ?, ?, ?)', [chat, sender, content, timestamp], (err) => {
                if (err) console.error(err);
                sendMessagesToClient(chat);
            });
        });

        function sendMessagesToClient(chat) {
            db.all('SELECT sender, content, timestamp FROM messages JOIN chats ON messages.chat_id = chats.id WHERE chats.chat_name = ? ORDER BY timestamp', [chat], (err, rows) => {
                if (err) console.error(err);
                io.to(chat).emit('messages', rows);
            });
        }

        function sendChatListToClient() {
            db.all('SELECT chat_name FROM chats', (err, rows) => {
                if (err) console.error(err);
                io.emit('chatList', rows);
            });
        }

        socket.on('disconnect', () => {
            console.log('User disconnected');
            delete users[socket.id];
        });
    });

    const PORT = process.env.PORT || 3000;
    server.listen(PORT, () => {
        console.log(`Server running on port ${PORT}`);
    });
```

database.js

```
const sqlite3 = require('sqlite3').verbose();
const db = new sqlite3.Database('./data.txt');
```

```
db.serialize(() => {

  db.run(`
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      username TEXT UNIQUE,
      password TEXT
    )
  `);


  db.run(`
    CREATE TABLE IF NOT EXISTS chats (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      chat_name TEXT UNIQUE
    )
  `);

  db.run(`
    CREATE TABLE IF NOT EXISTS messages (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      chat_id INTEGER,
      sender TEXT,
      content TEXT,
      timestamp TEXT,
      FOREIGN KEY(chat_id) REFERENCES chats(id)
    )
  `);
  db.run(`
    CREATE TABLE IF NOT EXISTS user_chats (
      user_id INTEGER,
      chat_id INTEGER,
      FOREIGN KEY(user_id) REFERENCES users(id),
      FOREIGN KEY(chat_id) REFERENCES chats(id),
      PRIMARY KEY (user_id, chat_id)
    )
  `);
```

```
});

module.exports = db;
```

script.js

```
const socket = io();

document.addEventListener('DOMContentLoaded', () => {
    const registerForm = document.getElementById('register-form');
    const registerButton = document.getElementById('register-btn');
    const registerUsername = document.getElementById('register-username');
    const registerPassword = document.getElementById('register-password');

    const loginForm = document.getElementById('login-form');
    const loginButton = document.getElementById('login-btn');
    const loginUsername = document.getElementById('login-username');
    const loginPassword = document.getElementById('login-password');

    const chatContainer = document.getElementById('chat-container');
    const messageInput = document.getElementById('message-input');
    const sendButton = document.getElementById('send-btn');
    const messagesContainer = document.getElementById('messages');
    const chatListContainer = document.getElementById('chat-list');
    const createChatButton = document.getElementById('create-chat-btn');
    const createChatForm = document.getElementById('create-chat-form');
    const createChatSubmit = document.getElementById('create-chat-submit');
    const newChatNameInput = document.getElementById('new-chat-name');
    const currentChatTitle = document.getElementById('current-chat');
    const deleteChatButton = document.getElementById('delete-chat-btn');

    let currentChat = '';
    let username = '';

    registerButton.addEventListener('click', () => {
        username = registerUsername.value;
        const password = registerPassword.value;
        fetch('/register', {
```

```
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
    })
    .then(response => response.text())
    .then(message => {
        alert(message);
        loginForm.style.display = 'none';
        registerForm.style.display = 'none';
        chatContainer.style.display = 'flex';
    })
    .catch(error => {
        console.error('Error registering user:', error);
        alert('Error registering user');
    });
});

loginButton.addEventListener('click', () => {
    username = loginUsername.value;
    const password = loginPassword.value;
    fetch('/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password })
    })
    .then(response => response.text())
    .then(message => {
        if (message === 'Login successful') {
            loginForm.style.display = 'none';
            registerForm.style.display = 'none';
            chatContainer.style.display = 'flex';

            socket.emit('joinChat', { chat: '', username });
        } else {
            alert(message);
        }
    })
```

```
    .catch(error => {
        console.error('Error logging in:', error);
        alert('Error logging in');
    });
});

sendButton.addEventListener('click', () => {
    const content = messageInput.value;
    if (content) {
        socket.emit('sendMessage', { chat: currentChat, sender: username, content });
        messageInput.value = '';
    }
});



socket.on('messages', messages => {
    messagesContainer.innerHTML = '';
    messages.forEach(msg => {
        const messageElement = document.createElement('div');
        messageElement.textContent = `[${msg.timestamp}] ${msg.sender}: ${msg.content}`;
        messagesContainer.appendChild(messageElement);
    });
});

socket.on('chatList', chats => {
    chatListContainer.innerHTML = '';
    chats.forEach(chat => {
        const chatButton = document.createElement('button');
        chatButton.textContent = chat.chat_name;
        chatButton.addEventListener('click', () => switchChat(chat.chat_name));
        chatListContainer.appendChild(chatButton);
    });
});

createChatButton.addEventListener('click', () => {
    createChatForm.style.display = 'block';
});
```

```
createChatSubmit.addEventListener('click', () => {
    const chatName = newChatNameInput.value;
    if (chatName) {
        fetch('/create_chat', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ chatName })
        })
        .then(response => response.text())
        .then(message => {
            if (message === 'Chat created') {
                const chatButton = document.createElement('button');
                chatButton.textContent = chatName;
                chatButton.addEventListener('click', () => switchChat(chatName));
                chatListContainer.appendChild(chatButton);
                newChatNameInput.value = '';
                createChatForm.style.display = 'none';
            }
        })
        .catch(error => {
            console.error('Error creating chat:', error);
            alert('Error creating chat');
        });
    }
});

socket.on('chatDeleted', chatName => {
    const chatButtons = Array.from(chatListContainer.getElementsByTagName('button'));
    const buttonToDelete = chatButtons.find(btn => btn.textContent === chatName);
    if (buttonToDelete) {
        buttonToDelete.remove();
    }
});

deleteChatButton.addEventListener('click', () => {
    if (confirm('Are you sure you want to delete this chat?')) {
```

```
            fetch(`/delete_chat/${currentChat}`, {
                method: 'DELETE'
            })
            .then(response => response.text())
            .then(message => {
                alert(message);
                switchChat('');
            })
            .catch(error => {
                console.error('Error deleting chat:', error);
                alert('Error deleting chat');
            });
        }
    });

    function switchChat(chatName) {
        messagesContainer.innerHTML = '';
        socket.emit('leaveChat', currentChat);
        socket.emit('joinChat', { chat: chatName, username });
        currentChat = chatName;
        currentChatTitle.textContent = chatName;
    }
});
```

app.js

```
const socket = io();
let currentChat = 'general';

socket.on('messages', (messages) => {
    const messagesDiv = document.getElementById('messages');
    messagesDiv.innerHTML = '';
    messages.forEach(message => {
        const messageElement = document.createElement('div');
        messageElement.textContent = `${message.timestamp}: ${message.sender}: ${message.content}`;
        messagesDiv.appendChild(messageElement);
    });
});
```

```
        socket.on('messages', (messages) => {
            const messagesDiv = document.getElementById('messages');
            messagesDiv.innerHTML = '';
            messages.forEach(message => {
                const messageElement = document.createElement('div');
                messageElement.textContent = `${message.timestamp}: ${message.sender}: ${message.content}`;
                messagesDiv.appendChild(messageElement);
            });
        });

        socket.on('chatList', (chatList) => {
            const chatListDiv = document.getElementById('chat-list');
            chatListDiv.innerHTML = '';

            const generalChatButton = document.createElement('div');
            generalChatButton.textContent = 'General';
            generalChatButton.classList.add('chat-tab');
            generalChatButton.onclick = () => switchChat('general');
            chatListDiv.appendChild(generalChatButton);

            chatList.forEach(chat => {
                const chatElement = document.createElement('div');
                chatElement.textContent = chat.chat_name;
                chatElement.classList.add('chat-tab');
                chatElement.onclick = () => switchChat(chat.chat_name);
                chatListDiv.appendChild(chatElement);

                const chatButton = document.createElement('button');
                chatButton.textContent = chat.chat_name;
                chatButton.onclick = () => openChat(chat.chat_name);
                    document.getElementById('app').insertBefore(chatButton,   document.getElementById('register-
login'));
            });
        });

        async function register() {
```

```
    const username = document.getElementById('register-username').value;
    const password = document.getElementById('register-password').value;
    const response = await fetch('/register', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ username, password })
    });
    if (response.status === 201) {
        alert('User registered');
    } else {
        alert('Registration failed');
    }
}

async function login() {
    const username = document.getElementById('login-username').value;
    const password = document.getElementById('login-password').value;
    const response = await fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ username, password })
    });
    if (response.ok) {
        document.getElementById('register-login').style.display = 'none';
        document.getElementById('chat').style.display = 'block';
        socket.emit('joinChat', currentChat);
    } else {
        alert('Invalid credentials');
    }
}

function sendMessage() {
    const sender = document.getElementById('login-username').value;
```

```
    const content = document.getElementById('message').value;
    socket.emit('sendMessage', { chat: currentChat, sender, content });
    document.getElementById('message').value = '';
}

async function createPrivateChat() {
    const chat_name = document.getElementById('private-chat-name').value;
    const user1 = document.getElementById('login-username').value;
    const user2 = document.getElementById('private-chat-user2').value;
    const response = await fetch('/create_private_chat', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ chat_name, user1, user2 })
    });
    if (response.status === 201) {
        alert('Private chat created');
        currentChat = chat_name;
        socket.emit('joinChat', currentChat);
        closeModal();
    } else {
        alert('Creating private chat failed');
    }
}

async function createChat() {
    const chatName = document.getElementById('new-chat-name').value;
    const response = await fetch('/create_chat', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ chatName })
    });
    if (response.status === 201) {
        alert('Chat created');
```

```
            currentChat = chatName;
            socket.emit('joinChat', currentChat);

            const chatButton = document.createElement('button');
            chatButton.textContent = chatName;
            chatButton.onclick = () => openChat(chatName);
                document.getElementById('app').insertBefore(chatButton,   document.getElementById('register-
login'));
        } else {
            alert('Creating chat failed');
        }
    }

    function switchChat(chat) {
        currentChat = chat;
        document.getElementById('current-chat').textContent = `Current Chat: ${chat}`;
        socket.emit('joinChat', currentChat);
    }

    function openModal() {
        document.getElementById('private-chat-modal').style.display = 'block';
    }

    function closeModal() {
        document.getElementById('private-chat-modal').style.display = 'none';
    }

    function openChat(chat) {
        currentChat = chat;
        document.getElementById('current-chat').textContent = `Current Chat: ${chat}`;
        socket.emit('joinChat', currentChat);
        document.getElementById('register-login').style.display = 'none';
        document.getElementById('chat').style.display = 'block';
    }

    async function register() {
        const username = document.getElementById('register-username').value;
```

```
    const password = document.getElementById('register-password').value;
    const response = await fetch('/register', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ username, password })
    });
    if (response.status === 201) {
        alert('User registered');
    } else {
        alert('Registration failed');
    }
}

async function login() {
    const username = document.getElementById('login-username').value;
    const password = document.getElementById('login-password').value;
    const response = await fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ username, password })
    });
    if (response.ok) {
        document.getElementById('register-login').style.display = 'none';
        document.getElementById('chat').style.display = 'block';
        socket.emit('joinChat', currentChat);
    } else {
        alert('Invalid credentials');
    }
}

function sendMessage() {
    const sender = document.getElementById('login-username').value;
    const content = document.getElementById('message').value;
```

```
    socket.emit('sendMessage', { chat: currentChat, sender, content });
    document.getElementById('message').value = '';
}

async function createPrivateChat() {
    const chat_name = document.getElementById('private-chat-name').value;
    const user1 = document.getElementById('login-username').value;
    const user2 = document.getElementById('private-chat-user2').value;
    const response = await fetch('/create_private_chat', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ chat_name, user1, user2 })
    });
    if (response.status === 201) {
        alert('Private chat created');
        currentChat = chat_name;
        socket.emit('joinChat', currentChat);
        closeModal();
    } else {
        alert('Creating private chat failed');
    }
}

async function createChat() {
    const chatName = document.getElementById('new-chat-name').value;
    const response = await fetch('/create_chat', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ chatName })
    });
    if (response.status === 201) {
        alert('Chat created');
        currentChat = chatName;
```

```
            socket.emit('joinChat', currentChat);


            const chatButton = document.createElement('button');
            chatButton.textContent = chatName;
            chatButton.onclick = () => openChat(chatName);
                document.getElementById('app').insertBefore(chatButton,   document.getElementById('register-
login'));
        } else {
            alert('Creating chat failed');
        }
    }

    function switchChat(chat) {
        currentChat = chat;
        document.getElementById('current-chat').textContent = `Current Chat: ${chat}`;
        socket.emit('joinChat', currentChat);
    }

    function openModal() {
        document.getElementById('private-chat-modal').style.display = 'block';
    }

    function closeModal() {
        document.getElementById('private-chat-modal').style.display = 'none';
    }

    function openChat(chat) {
        currentChat = chat;
        document.getElementById('current-chat').textContent = `Current Chat: ${chat}`;
        socket.emit('joinChat', currentChat);
        document.getElementById('register-login').style.display = 'none';
        document.getElementById('chat').style.display = 'block';
    }
```

Додаток В

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

Додаток С

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
ПЗ_Клименко.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
ПЗ_Клименко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
messenger.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Клименко.ppt	Презентація кваліфікаційної роботи