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 *Віннік Артем Вадимович*
(ПІБ)

академічної групи *122-20-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми Комп'ютерні науки
(назва освітньої програми)

на тему: *Розробка месенджеру для організації листування між користувачами з використанням ASP.NET Core, WebSocket, MS SQL Server.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

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 *122-20-3* (група) *Віннік Артем Вадимович* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка месенджеру для організації листування між користувачами з використанням ASP.NET Core, WebSocket, MS SQL Server.*

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

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***20.05.2024.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***03.06.2024 р.***

Завдання видав _____ *доц. Спірінцев В.В*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Віннік А.В*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 88 с., 8 рис., 3 дод., 20 джерел.

Об'єкт розробки: веб-застосунок для організації листування між користувачами.

Мета кваліфікаційної роботи: розробка та впровадження ефективного та безпечного месенджера для організації листування між користувачами. Для досягнення цієї мети передбачається використання сучасних технологій, таких як ASP.NET Core для серверної частини, WebSocket для забезпечення реального часу комунікації та MS SQL Server для зберігання даних.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформу для розробки, виконано проектування і розробку системи віртуального середовища, описана робота системи, алгоритм і структура його функціонування, а також виклик та завантаження додатку, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленого застосунку, проведений підрахунок вартості роботи по створенню додатку та розраховано час на його створення.

Практичне значення полягає у створенні месенджера, який дозволить керувати спілкуванням між користувачами та взаємодіяти з різноманітними функціями обміну повідомленнями, а також забезпечить реалізацію механік сповіщень та підтримки реального часу.

Актуальність розробки програмного продукту полягає в попиті на готові рішення в галузі комунікаційних технологій, а також постійному розвитку цієї сфери, що створює платформу для розробки інноваційних рішень.

Список ключових слів: WEBSOCKET, МЕСЕНДЖЕР, КЛІЄНТ, БАЗА ДАНИХ, ЗАСТОСУНОК, ПОВІДОМЛЕННЯ.

ABSTRACT

Explanatory note: 88 p., 8 figures, 3 appendices, 20 sources.

Object of development: a web application for organizing correspondence between users.

The purpose of the qualification work: development and implementation of an effective and safe messenger for organizing correspondence between users. To achieve this goal, the use of modern technologies such as ASP.NET Core for the server part, WebSocket for real-time communication and MS SQL Server for data storage is envisaged.

In the introduction, the analysis and current state of the problem is considered, the purpose of the qualification work and the field of its application are specified, the justification of the relevance of the topic is given, and the statement of the task is clarified.

In the first section, the subject area is analyzed, the relevance of the task and the purpose of the development is determined, the task statement is formulated, and the requirements for software implementation, technologies and software tools are specified.

In the second section, available solutions are analyzed, a platform for development is selected, the design and development of the virtual environment system is performed, the operation of the system is described, the algorithm and structure of its functioning, as well as the application call and download, the input and output data are determined, and the composition of technical means parameters is characterized.

In the economic section, the labor intensity of the developed application is determined, the cost of work on creating the application is calculated, and the time for its creation is calculated.

The practical value is to create a messenger that will allow you to manage communication between users and interact with various messaging functions, as well as provide the implementation of notification mechanics and real-time support.

The relevance of software product development lies in the demand for ready-made solutions in the field of communication technologies, as well as the constant development of this field, which creates a platform for the development of innovative solutions.

List of keywords: WEBSOCKET, MESSENGER, CLIENT, DATABASE, APPLICATION, MESSAGE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ-	Програмне забезпечення
ПК-	Персональний комп’ютер
БД-	База даних
SQL-	Structured Query Language
API-	Application Programming Interface
XSS-	Cross-Site Scripting
CSRF-	Cross-Site Request Forgery

ЗМІСТ

ВСТУП

У сучасному світі швидкий обмін інформацією є однією з найважливіших складових успіху. Інтернет-спілкування відіграє ключову роль у цьому ландшафті. Спостерігаючи за особистим досвідом і сучасними тенденціями, стає очевидним, що сучасні соціальні мережі часто не можуть забезпечити ту зручність і безпосередність, які пропонують месенджери.

Попит на ефективні, надійні та зручні платформи обміну повідомленнями постійно зростає. Хоча соціальні мережі чудово об’єднують людей і сприяють широким обговоренням, їм часто бракує спрощених можливостей спілкування в реальному часі, які надають месенджери. Месенджер може подолати цю прогалину, пропонуючи розширені функції, адаптовані до потреб користувачів, які віддають перевагу швидкості та простоті спілкування.

Основні причини створення нової платформи обміну повідомленнями:

– Покращена комунікація в режимі реального часу: на відміну від соціальних мереж, месенджери створені для миттєвого обміну повідомленнями, що є вирішальним як для особистої, так і для професійної взаємодії.

– Цілеспрямована функціональність: спеціальна платформа може запропонувати такі функції, як шифрування, спільний доступ до файлів і безперебійну синхронізацію на кількох пристроях, забезпечуючи більш згуртоване спілкування.

– Конфіденційність і безпека користувачів: користувачі все більше турбуються про конфіденційність своїх даних. Нова платформа може інтегрувати розширені заходи безпеки для захисту інформації та розмов користувачів.

– Налаштування та гнучкість. Пропонування інтерфейсів та індивідуальних налаштувань може підвищити задоволеність та залучення користувачів.

Перш ніж приступати до розробки нового месенджера, необхідно вирішити кілька важливих завдань, щоб забезпечити його успіх і стабільність:

1) Економічна доцільність:
    a) Потрібно провести ретельний аналіз ринку, щоб визначити попит на новий месенджер
    b) Потрібно розробити бізнес-модель, яка окреслює джерела доходу, такі як підписки, реклама або покупки в програмі.
    c) Потрібно оцінити необхідні початкові інвестиції та спрогнозувати повернення інвестицій (ROI) з часом.
2) Основні вимоги:
    a) Визначити чіткі критерії прийнятності для платформи, включаючи контрольні показники продуктивності, цілі взаємодії з користувачем і стандарти безпеки.
    b) Встановити вимірювані цілі для оцінки успіху платформи, такі як показники залучення користувачів, показники утримання та показники задоволеності користувачів.
3) Технічні характеристики:
    a) Окреслити технічні вимоги та архітектуру платформи, що забезпечує масштабованість і надійність.
    b) Вибрати відповідні технології та фреймворки (в нашому випадку, ASP.NET Core, WebSocket, MS SQL Server), щоб створити надійну серверну частину.

Практичний сенс полягає в тому, щоб створити веб-орієнтовану програму, яка забезпечує користувачам безперебійне спілкування в реальному часі. Ця програма має на меті покращити спосіб взаємодії окремих людей і команд, пропонуючи платформу, яка є не тільки ефективною та надійною, але також безпечною та зручною для користувачів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

1.1.1 Вплив месенджерів у сучасних умовах України

Розвиток цифрових технологій і загальна тенденція до цифровізації призвели до широкого використання Інтернету практично в усіх секторах. У наш час неможливо уявити спілкування без інтернет-месенджерів. Ці месенджери інтегрувалися в соціальну структуру мільйонів, слугуючи не лише інструментами спілкування, але й інструментами для маркетингу, бізнесу та освіти. Майже кожен користувач смартфона використовує їх, використовуючи цифрові канали зв’язку для передачі даних.

Оскільки технічний і програмний доступ до Інтернету продовжує розвиватися, захист інформації під час зберігання та передачі стає серйозною проблемою. Критичним елементом інформаційної безпеки є етап передачі цифровими каналами зв’язку, де ризик перехоплення даних найбільш виражений. Перехоплення пакетів даних, що проходять через незахищені (або частково захищені) канали, створює можливості для спотворення інформації, блокування та перехоплення особистих, конфіденційних або обмежених даних.

У сучасному світі інформація є стратегічним ресурсом. Спотворення, перехоплення або блокування інформації може мати серйозні наслідки. В умовах стрімких глобальних змін все більшого значення набувають як існуючі, так і нові загрози національній безпеці України [1].

Згідно досліджень Київського міжнародного інституту соціології на момент липня 2022 року 78% респондентів за останні 7 днів використовували

месенджери для спілкування. У середньому ті, хто користуються цими додатками, використовують 2–3 різних месенджери.

Безумовний лідер серед месенджерів України – Viber, який зараховують до топ-2 найважливіших для себе месенджерів 58% серед усіх респондентів, а загалом 66% за останні 7 днів користувалися ним. Далі йдуть Telegram (32% і 41% відповідно) і Facebook Messenger (20% і 34%). Інші месенджери менш популярні серед населення країни [2].

Взагалі що таке месенджер в сучасному розумінні цього слова — це програма обміну повідомленнями, яка дозволяє надсилати повідомлення, фотографії, відео, документи, а також здійснювати голосові та відеодзвінки. Його політика конфіденційності базується на шифруванні даних і відмові від співпраці з органами влади. На жаль це працює в ідеальному світі, наприклад в Telegram end-to-end шифрування повідомлень працює тільки в режимі Секретний чат, та і самі месенджери використовуються для розповсюдження пропаганди.

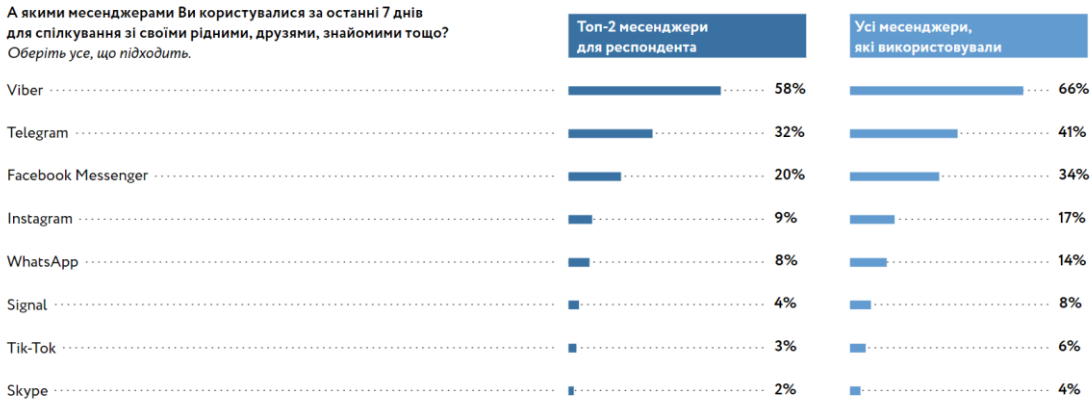

Рис. 1.1.1. Статистика використання месенджерів в Україні

1.1.2 Аналіз готових існуючих рішень

Було обрано два сайти для аналізу: Telegram [17], та WhatsApp [18], так як по суті тільки ці месенджери мають веб версію.

Ці веб-застосунки мають привабливий мінімалістичний дизайн, використовують стримані кольори, що приваблює око користувача та не дає очам втомлюватися.

Обидва сайти мають таку структуру:

- Хедер
- Сайдбар з чатами користувача
- Основне тіло де знаходиться вибраний чат.

Загалом з точки зору дизайну сайти не сильно відрізняються, що не можна сказати з точки зору функціоналу. Обидві програми мають специфічні функції та інтеграції. WhatsApp збирає основні дані, забезпечує наскрізне шифрування і дозволяє видаляти повідомлення протягом обмеженого часу. На противагу цьому, Telegram не має стандартного наскрізного шифрування, але пропонує самознищуванні повідомлення та секретні чати для покращеної конфіденційності.

Щодо групових можливостей, WhatsApp підтримує менші групи до 1024 учасників, тоді як Telegram може вміщувати групи до 200 000 учасників. Telegram перевершує WhatsApp у можливостях обміну файлами, дозволяючи користувачам ділитися файлами розміром до 2 ГБ. Функції автоматизації дещо відрізняються: WhatsApp надає API [16] для автоматизованих повідомлень, а Telegram підтримує сторонні програми для автоматизації.

Telegram виділяється можливостями співпраці, дозволяючи доступ з кількох пристроїв для безперебійної взаємодії команди. Однак активна база користувачів і функції безпеки WhatsApp роблять його більш привабливим для компаній, які цінують взаємодію та захист даних. Хоча Telegram пропонує такі

переваги, як більша місткість груп та контроль, WhatsApp виділяється такими функціями, як історії та групові дзвінки, що задовольняють різноманітні потреби бізнесу.

Інтеграція Telegram або WhatsApp Business в існуючі бізнес-процеси часто потребує додаткових інструментів сторонніх розробників. Більшості компаній знадобляться додаткові програми для керування проєктами, обміну файлами та співпраці.

Наприклад, компанії можуть інтегрувати Telegram із програмами керування завданнями, такими як Trello чи Asana, для відстеження проєктів або WhatsApp Business із системами CRM для керування взаєминами з клієнтами. Налаштування цих інтеграцій може зайняти багато часу та вимагати додаткових витрат.

Далі узагальнено по функціоналу:

Функціонал WhatsApp:

– Кількість учасників групи 1024 особи
– До 32 учасників у голосовому або відеодзвінку
– Автоматично стискає зображення, відео та інші файли
– Боти доступні для облікових записів WhatsApp Business і Whatsapp Business API
– Зберігає медіафайли на мобільному сховищі
– Обмеження обміну файлами 2 Гб
– Можливість використовувати до чотирьох пристроїв з одним обліковим записом
– Можливість створювати та ділитися опитуваннями в групах
– Опція зникнення повідомлень для повідомлень, які не були прочитані протягом 24 годин, 7 днів або 90 днів
– Можливість видаляти повідомлення для всіх протягом двох днів
– Нова функція спільнот WhatsApp

Функціонал Telegram:

– Місткість групи 200 000 учасників

– Немає обмежень на кількість учасників голосового дзвінка

– До 30 учасників можуть транслювати свою камеру або екран, і до 1000 учасників можуть дивитися

– Запитує дозвіл на стиснення файлів

– Боти доступні для всіх користувачів

– Зберігає медіафайли на сервері (необмежене зберігання)

– Можливість редагувати повідомлення протягом 48 годин

– Ліміт обміну файлами 2 ГБ (і 4 ГБ з Telegram Premium)

– Можливість імпортувати чати з інших програм обміну повідомленнями

– Можливість створити до трьох облікових записів на одному мобільному телефоні та прив’язати обліковий запис до кількох пристроїв

– Створення та обмін опитуваннями в групах

– Опція секретного чату з повідомленнями, які самознищуються

– Можливість видаляти повідомлення, надіслані з будь-якого кінця чату

– Функція каналів Telegram

– Вбудована програма для створення наклейок

Як бачимо функціонал Telegram набагато ширше, що дозволяє йому конкурувати навіть з соціальними мережами [3].

1.2 Призначення розробки та галузь застосування

Насправді галузь застосування месенджерів вкрай висока. В більшості зазвичай залежить все від наявного функціоналу в застосунка. Це може бути як

повсякденне життя, так і організація бізнес процесів за допомогою групових чатів, каналів і т.д..

1.2.1 Галузь застосування у повсякденному житті

Сучасні месенджери мають величезний функціонал, тому по суті все опирається у фантазію та потреби кожної людини.

Спілкування з родиною та друзями в сучасному світі значно полегшене завдяки використанню месенджерів. Миттєві повідомлення дозволяють швидко обмінюватися текстовими повідомленнями, що робить щоденне спілкування легким та зручним. Голосові та відеодзвінки забезпечують можливість проведення віртуальних зустрічей, що є особливо важливим, коли друзі чи члени родини знаходяться далеко один від одного. Ці функції дозволяють відчувати присутність близьких, навіть якщо вони перебувають на іншому кінці світу. Додатково, групові чати створюють простір для організації групових розмов, де можна обговорювати спільні плани та події, що сприяє підтримці соціальних зв'язків і зміцненню відносин у колективі. Завдяки таким можливостям, месенджери стають незамінним інструментом для підтримки активного і насиченого соціального життя.

Організація повсякденного життя значно спрощується завдяки можливостям, які надають месенджери. Однією з важливих функцій є планування подій, що здійснюється через використання календарів і нагадувань у месенджерах. Це дозволяє легко організовувати зустрічі, дні народжень, поїздки та інші важливі події, забезпечуючи своєчасне нагадування про них. Також корисною є можливість створення списків покупок та завдань. Користувачі можуть складати списки справ, які потрібно виконати, а також спільні списки покупок, що дуже зручно для сімей або співмешканців. Крім того, взаємодія з сервісами через месенджери стала простою та ефективною завдяки чат-ботам. За їх допомогою можна замовляти різноманітні послуги, такі як

доставка їжі або бронювання столиків у ресторані. Це значно спрощує повсякденні завдання і робить життя більш організованим і менш стресовим.

Обмін мультимедіа, такий як надсилання фотографій, відео, мемів та іншого контенту, забезпечує користувачам чудовий спосіб розважатися та ділитися цікавими моментами зі своїми друзями і близькими. Крім того, месенджери дозволяють брати участь у групах за інтересами, де люди можуть обговорювати спільні хобі, такі як музика, спорт, ігри чи мистецтво. Ці групи створюють спільноти однодумців, де можна знайти підтримку, нові ідеї та натхнення. Деякі месенджери також підтримують інтеграцію з онлайн-іграми, що дозволяє користувачам спільно проводити час, змагаючись або співпрацюючи в ігровому середовищі. Це не тільки розважає, але й сприяє зміцненню соціальних зв'язків та розвитку командних навичок. Таким чином, месенджери надають різноманітні інструменти для розваг та розвитку хобі, роблячи їх доступними і захопливими для всіх користувачів.

Інформаційна підтримка, яку надають месенджери, є важливою складовою повсякденного життя користувачів. Однією з основних можливостей є отримання новин. Користувачі можуть підписуватися на новинні канали та групи, щоб завжди бути в курсі актуальної інформації. Це дозволяє їм оперативно отримувати свіжі новини, важливі оновлення та інші необхідні дані, що забезпечує обізнаність і поінформованість у будь-який момент.

1.2.2 Галузь застосування у організації бізнес процесів

Внутрішня комунікація в межах компанії вдосконалюється завдяки застосуванню месенджерів. Вони служать зручним засобом для оперативного обміну інформацією між співробітниками. Завдяки цьому, співробітники можуть швидко ділитися ідеями, вирішувати питання, а також спільно працювати над різними проєктами та завданнями. Месенджери створюють сприятливе

середовище для відкритого обговорення та ефективної співпраці, що сприяє підвищенню продуктивності та досягненню більш високих результатів.

Бізнеси активно використовують месенджери для забезпечення ефективної комунікації зі своїми клієнтами. Це стає не лише засобом швидкої відповіді на запити клієнтів, але і можливістю надання консультаційної підтримки в режимі реального часу. Крім цього, месенджери використовуються для отримання та обробки замовлень, що дозволяє клієнтам зручно здійснювати покупки через цей канал зв'язку. Окрім цього, бізнеси можуть використовувати месенджери для надання додаткової інформації про свої продукти чи послуги, що допомагає клієнтам зробити обдуманий вибір. Також месенджери використовуються для обробки скарг чи повернень, що сприяє підвищенню задоволеності клієнтів та підвищенню рівня сервісу.

Месенджери є важливим інструментом у взаємодії з партнерами, сприяючи ефективному обговоренню та узгодженню умов співпраці. Вони дозволяють легко обмінюватися ідеями, обговорювати стратегії та планувати проекти в режимі реального часу. Месенджери також використовуються для проведення переговорів і планування зустрічей з партнерами. Крім того, вони надають можливість організувати віртуальні конференції, що дозволяє зберігати зв'язок та координувати спільні дії, незалежно від місця знаходження учасників. Таким чином, месенджери сприяють підтримці та розвитку взаємовідносин з партнерами, що є важливим для успішної діяльності компанії.

Попри це месенджери виконують важливу роль у керуванні завданнями та проєктами. Команди можуть створювати спільні чати для зручного обговорення, планування та координації завдань та проєктів. Це дозволяє учасникам швидко обмінюватися ідеями, встановлювати пріоритети та розподіляти обов'язки. Крім цього, месенджери надають можливість створювати нагадування про важливі дедлайни, що допомагає уникнути прострочення завдань та вчасно завершувати проекти. Також вони забезпечують зручний інтерфейс для відстеження прогресу

виконання завдань та оцінки їхнього статусу. У цілому, месенджери стають невід'ємною частиною робочого процесу, сприяючи ефективному керуванню завданнями та досягненню поставлених цілей.

Обмін документами та файлами через месенджери виявляється дуже зручним та ефективним способом спільної роботи. Він дозволяє користувачам не лише легко надсилати та отримувати різноманітні матеріали, такі як документи, фотографії, відео, а й швидко ділитися ними з колегами та клієнтами. Це сприяє збереженню часу та покращує комунікацію в рамках бізнес-процесів, оскільки дозволяє учасникам команди оперативно отримувати необхідну інформацію та матеріали для виконання своїх завдань. Такий обмін файлами допомагає підвищити ефективність співпраці та зробити робочий процес більш організованим та продуктивним.

За останні роки зросла популярність віддаленої роботи, і месенджери стали важливим інструментом для забезпечення зв'язку між віддаленими співробітниками та командами. Вони надають можливість легко та ефективно утримувати зв'язок навіть у віддалених робочих умовах. Це дозволяє співробітникам зв'язуватися між собою для обговорення проєктів, обміну інформацією та координації дій, незалежно від їхнього місця перебування. Месенджери стають важливим інструментом для підтримки продуктивної віддаленої роботи, дозволяючи співробітникам легко спілкуватися та спільно працювати над завданнями, навіть коли вони подалі один від одного.

1.3 Підстави для розробки

Підставами для розробки (виконання кваліфікаційної роботи) є:

- освітня програма спеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська

– політехніка» №469-с від 23.05.2024 р.;

– завдання на кваліфікаційну роботу на тему «Розробка месенджеру для організації листування між користувачами з використанням ASP.NET Core, WebSocket, MS SQL Server.»;

1.4. Постановка завдання

Мета цієї роботи: розробка месенджеру для організації листування між користувачами з використанням таких технологій як ASP.NET Core, WebSocket, MS SQL Server.

Призначення месенджера полягає у спрощенні комунікації між користувачами, надаючи їм можливість відправляти текстові повідомлення.

Завдання створення месенджера охоплює як технічні, так і економічні аспекти, які необхідно врахувати для досягнення успіху. Технічна сутність цього завдання полягає у визначенні вимог до системи, що включають функціональні та нефункціональні аспекти. Функціональні вимоги описують основні дії, які має виконувати месенджер, такі як реєстрація та аутентифікація користувачів, обмін текстовими повідомленнями, голосові та відеодзвінки, передача мультимедійних файлів, створення групових чатів і забезпечення захищеного спілкування. Нефункціональні вимоги охоплюють характеристики, як-от висока швидкість обробки даних, надійна безпека, масштабованість системи та доступність на різних платформах і пристроях.

Технічно-економічна сутність завдання: створення веб застосунку з дизайном до якого звикли користувачі, застосунок має відповідати мінімальним стандартам по безпеці. Економічна сутність завдання включає оцінку витрат на розробку та експлуатацію веб-застосунку.

1.5 Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблений застосунок має мати такий функціонал:

- шифрування особистих даних користувача включаючи повідомлення
- можливість видалення повідомлень
- можливість редагування повідомлень
- можливість видалення чату у обох користувачів
- пошук користувачів за тегом
- можливість блокування та розблокування користувача
- Реєстрація користувача
- Видалення акаунта

1.5.2. Вимоги до інформаційної безпеки

Забезпечення безпеки месенджерів передбачає реалізацію комплексного набору вимог до інформаційної безпеки. Ці вимоги є критично важливими для захисту даних користувача та цілісності зв’язку.

Наскрізне шифрування, яке часто називають E2EE, є заходом безпеки, який забезпечує конфіденційність повідомлень під час спілкування. За допомогою E2EE процес шифрування повідомлень відбувається безпосередньо на пристрої відправника, перетворюючи повідомлення в закодований формат, який неможливо легко розшифрувати. Після зашифрування ці повідомлення надсилаються через мережу та можуть бути розшифровані назад у вихідний читабельний формат лише одержувачем, який має необхідний ключ розшифровки.

Цей механізм шифрування відіграє вирішальну роль у захисті конфіденційності та безпеки зв'язку. Шифруючи повідомлення в джерелі та зберігаючи це шифрування, доки вони не досягнуть одержувача, E2EE ефективно блокує будь-яких посередників від доступу до вмісту. Це стосується не лише потенційних зловмисників, які можуть перехопити дані під час передачі, але й самих постачальників послуг. Постачальники послуг, незважаючи на сприяння передачі цих повідомлень, не можуть розшифрувати або прочитати зашифрований вміст.

Таким чином, наскрізне шифрування є надійним рішенням для захисту конфіденційної інформації, гарантуючи, що лише сторони, що спілкуються, можуть отримати доступ до фактичного вмісту повідомлень. Ця технологія особливо важлива в середовищах, де конфіденційність має першорядне значення, надаючи користувачам впевненість у тому, що їхні повідомлення залишаються конфіденційними та захищеними від несанкціонованого доступу на всіх етапах передачі.

Впровадження методів безпечного кодування має важливе значення для розробки програмного забезпечення, стійкого до поширених уразливостей безпеки. Ці практики включають написання коду таким чином, щоб проактивно пом'якшувати потенційні загрози безпеці та гарантувати загальну цілісність програми. Однією з головних цілей безпечного кодування є запобігання відомим уразливостям, таким як впровадження SQL, міжсайтовий сценарій (XSS) і підробка міжсайтового запиту (CSRF).

SQL-ін'єкція [19] – це тип атаки, коли зловмисні оператори SQL вставляють у поле введення для виконання, що порушує безпеку бази даних. Використовуючи методи безпечного кодування, розробники можуть запровадити належну перевірку введення та параметризовані запити, щоб запобігти таким атакам. Міжсайтовий сценарій (XSS) включає введення шкідливих сценаріїв на веб-сторінки, які переглядають інші користувачі, потенційно крадіжку їхніх

файлів cookie сеансу або видавання себе за них. Практики безпечного кодування стосуються XSS, гарантуючи, що всі вхідні дані належним чином оброблені та закодовані перед відображенням на веб-сторінці. Підробка міжсайтових запитів (CSRF [20]) змушує користувача надіслати зловмисний запит, зазвичай через веб-браузер, який виконує дію від імені користувача без його згоди. Цьому можна запобігти, включивши анти-CSRF-токени та перевіряючи автентичність запитів.

Окрім написання безпечного коду, дуже важливо проводити регулярні перевірки коду та статичний аналіз коду. Перевірка коду передбачає систематичне вивчення вихідного коду іншими розробниками для виявлення потенційних недоліків безпеки та забезпечення дотримання стандартів безпечного кодування. Цей спільний підхід не тільки допомагає виявляти вразливості, але й сприяє обміну знаннями та покращує загальну якість коду. Статичний аналіз коду, з іншого боку, передбачає використання автоматизованих інструментів для сканування вихідного коду на наявність проблем із безпекою без виконання програми. Ці інструменти можуть ідентифікувати шаблони та конструкції коду, які можуть призвести до вразливостей, дозволяючи розробникам усунути їх на ранніх стадіях процесу розробки.

Разом ці практики утворюють комплексний підхід до безпечного кодування. Інтегруючи методи безпечного кодування, регулярні перевірки коду та статичний аналіз коду в життєвий цикл розробки, організації можуть значно знизити ризик порушення безпеки та створювати надійні безпечні програми.

Мінімізація та анонімізація даних є критично важливими практиками у сфері безпеки та конфіденційності даних, зосереджуючись на відповідальному зборі та зберіганні інформації користувачів. Принцип мінімізації даних передбачає, що організації повинні збирати та зберігати лише найменшу кількість даних, необхідних для ефективного функціонування їхніх послуг. Такий підхід не тільки зменшує потенційний ризик, пов’язаний із порушенням даних, але й узгоджується з правилами конфіденційності та етичними

стандартами. Обмеживши збір даних виключно необхідним, компанії можуть мінімізувати розголошення конфіденційної інформації, тим самим захищаючи користувачів від можливого зловживання чи несанкціонованого доступу.

Крім того, там, де це можливо, зібрані дані повинні бути анонімізовані. Анонімізація — це процес видалення або зміни особистих ідентифікаторів у наборі даних таким чином, щоб людей не можна було легко ідентифікувати. Ця практика має вирішальне значення для захисту ідентичності користувачів, гарантуючи, що навіть якщо до даних отримають доступ неавторизовані особи, залишається складним відстежити певних осіб. Анонімні дані все ще можуть бути корисними для аналітичних і оперативних цілей без шкоди для особистої конфіденційності.

Запровадження цих практик вимагає глибокого розуміння того, що є необхідними даними для роботи служби, і визначення методів ефективної анонімізації даних. Організації повинні оцінити свої процеси збору даних і політику зберігання, переконавшись, що вони дотримуються принципу мінімізації даних. Це включає перегляд типів даних, що збираються, причин їх збору та тривалості, протягом якої вони зберігаються. Постійно оцінюючи та вдосконалюючи ці процеси, компанії можуть гарантувати, що вони зберігають лише важливу інформацію, таким чином зменшуючи обсяг даних.

Паралельно процес анонімізації має бути інтегрований у процедури обробки даних. Для ефективної анонімізації даних можна використовувати такі методи, як маскування даних, псевдонімізація та агрегація. Маскування даних передбачає приховування вихідних даних із зміненим вмістом, псевдонімізація замінює приватні ідентифікатори підробленими ідентифікаторами або псевдонімами, а агрегація об’єднує дані так, що окремі записи неможливо розрізнити в більшому наборі даних. Ці методи допомагають трансформувати дані таким чином, щоб вони зберігали свою корисність, одночасно захищаючи ідентифікаційні дані користувачів.

Разом мінімізація та анонімізація даних утворюють надійну основу для відповідального керування даними користувачів. Дотримуючись цих правил, організації можуть посилити заходи безпеки своїх даних, відповідати вимогам законодавства та зберегти довіру своїх користувачів. Такий підхід не тільки зменшує ризики, пов’язані з витоком даних, але й дотримується етичних стандартів конфіденційності та захисту користувачів.

1.5.3. Вимоги до складу та параметрів технічних засобів

Забезпечення відповідності та виконання вимог до апаратного забезпечення для веб-додатків, написаних на ASP.NET Core та з використанням MS SQL Server, передбачає глибоке розуміння технічних аспектів, безпеки, продуктивності та нормативних аспектів. Ось розширене обговорення цих вимог і параметрів:

Процесор:

– Потрібно використовувати багатоядерні процесори (чотирьох ядерні чи новіші) для ефективної обробки одночасних запитів.

– Процесори Intel Xeon або AMD EPYC через їхню надійну продуктивність у серверних середовищах.

Пам'ять (RAM):

– Для малих і середніх додатків рекомендовано мінімум 16 ГБ.

– У майбутньому можна масштабувати до 32 ГБ або більше для якщо буде великий попит на платформу.

Зберігання:

– Бажано використовувати твердотільні накопичувачі (SSD) для більшої швидкості читання/запису порівняно з традиційними жорсткими дисками (HDD).

– Потрібно забезпечте достатній обсяг пам’яті для обробки розміру бази даних, файлів програм, журналів і резервних копій.

– Рекомендовано застосувати конфігурації RAID (RAID 1 або RAID 10) для резервування та підвищення продуктивності.

Мережа:

– Забезпечте високошвидкісне мережеве з’єднання (1 Гбіт/с або вище) для ефективної передачі даних і зв’язку.

– Розглянемо надлишкові мережеві інтерфейси для підвищення надійності та часу безвідмовної роботи.

1.5.4. Вимоги до інформаційної та програмної сумісності

Так як використовується MS SQL Server, то по суті сумісність буде тільки з Windows, але так як при необхідності можна використати контейнеризацію, то розглянемо не тільки Windows:

Операційна система:

– Windows Server 2016/2019/2022 для оптимальної сумісності з ASP.NET Core і MS SQL Server.

– Крім того, дистрибутиви Linux (такі як Ubuntu, CentOS), якщо використовуються кросплатформні можливості .NET Core.

Веб-сервер:

– Буде використано IIS як основний веб-сервер для програм ASP.NET Core.

– За бажанням можна використати Kestrel, Nginx або Apache як зворотний проксі для додаткових функцій і безпеки.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Функціональне призначення системи

Розробка месенджера спрямована на сприяння безперебійному листуванню між користувачами шляхом використання сучасних веб-технологій, таких як ASP.NET Core, WebSocket і MS SQL Server. Система розроблена для забезпечення зв'язку в режимі реального часу, надійного керування даними та зручного використання. Функціональне призначення системи охоплює такі основні компоненти та функції:

– Користувачі можуть створювати облікові записи, надаючи необхідну інформацію, таку як ім'я користувача, адреса електронної пошти та пароль.

– Система використовує безпечні механізми автентифікації (наприклад, OAuth, JWT) для перевірки облікових даних користувача під час входу.

– Функції включають відновлення пароля та скидання для підвищення безпеки користувачів.

– Користувачі можуть створювати та налаштовувати свої профілі з особистою інформацією та зображенням профілю.

– Користувачі можуть оновлювати дані свого профілю та налаштування.

– Встановлює та підтримує з'єднання WebSocket для передачі даних у реальному часі.

– Підтримує миттєве надсилання та отримання текстових повідомлень між користувачами через WebSocket.

– Зберігає надіслані та отримані повідомлення в структурованому форматі в базі даних MS SQL Server.

– Дозволяє користувачам отримувати історію повідомлень із бази даних.

– Дозволяє користувачам починати чати один на один з іншими користувачами.

– Надає зручний інтерфейс для створення, надсилання та читання повідомлень.

– Дозволяє користувачам створювати групові чати з кількома учасниками.

– Включає функції для додавання та видалення учасників, призначення адміністраторів групи та керування параметрами групи.

– Push-сповіщення про вхідні повідомлення та інші відповідні події.

– Надсилає сповіщення електронною поштою або в програмі користувачам, які перебувають у режимі офлайн, гарантуючи, що вони отримують інформацію про пропущені повідомлення.

– Використовує MS SQL Server для надійного зберігання та керування даними користувача, історіями повідомлень та іншими даними програм.

– Забезпечує шифрування конфіденційних даних під час передачі та зберігання.

– Реалізує контроль доступу на основі ролей для захисту цілісності даних і конфіденційності користувачів.

– Реалізує стратегії для балансування навантаження для обробки великих обсягів одночасних користувачів і повідомлень.

– Розроблено для горизонтального масштабування шляхом додавання додаткових серверів для ефективного керування збільшеним навантаженням.

– Дозволяє адміністраторам керувати обліковими записами користувачів і контролювати дії користувачів.

– Надає інструменти для моніторингу в реальному часі продуктивності системи, взаємодії користувачів і потоку повідомлень.

– Веде детальні журнали дій користувачів і системних подій для цілей аудиту.

– Відстежує та реєструє помилки для усунення несправностей і підвищення надійності системи.

– Гарантує, що програма доступна та функціональна на різних пристроях, включаючи настільні ПК, планшети та смартфони.

– Основна увага приділяється інтуїтивно зрозумілій навігації, простоті використання та привабливому користувальницькому досвіду.

– Підтримує надсилання та отримання мультимедійних повідомлень, таких як зображення, відео та аудіофайли.

– Надає надійні функції пошуку, які допомагають користувачам ефективно знаходити повідомлення, контакти та групи.

– Показує статус присутності користувача (онлайн, офлайн, введення тексту) для покращення спілкування в реальному часі.

2.2. Опис застосованих математичних методів

У розробці месенджера для організації листування між користувачами математичні методи відіграють вирішальну роль у забезпеченні ефективності, безпеки та надійності. У цьому розділі розглядаються різні математичні методи та алгоритми, що застосовуються в системі, зосереджуючись на шифруванні, цілісності даних, балансуванні навантаження та оптимізації продуктивності.

2.2.1 Криптографічні методи

Симетричне шифрування використовується для захисту вмісту повідомлень під час передачі та зберігання. Розширений стандарт шифрування (AES) — це широко поширений алгоритм симетричного шифрування, відомий

своєю надійністю та ефективністю. AES працює з фіксованими розмірами блоків (128 біт) і підтримує розміри ключів 128, 192 і 256 біт.

– Процес шифрування: перетворює відкритий текст на зашифрований за допомогою секретного ключа.

– Процес дешифрування: перетворює зашифрований текст назад у відкритий текст за допомогою того самого секретного ключа.

Математична основа:

$$C = E_k(P), \quad (2.1)$$

$$P = D_k(C), \quad (2.2)$$

де P – це відкритий текст, C – зашифрований текст, E_k – це функція шифрування з ключом k, та D_k – це функція дешифрування з ключом k.

Асиметричне шифрування або криптографія з відкритим ключем використовується для безпечного обміну ключами та процесів автентифікації. Алгоритм RSA є поширеним вибором для асиметричного шифрування, яке покладається на математичні властивості великих простих чисел.

– Генерація ключів: передбачає створення пари ключів – відкритого та закритого.

– Шифрування: використовує відкритий ключ одержувача для шифрування повідомлення.

– Розшифровка: для розшифровки повідомлення використовується особистий ключ одержувача.

Математична основа:

$$C = M^e \bmod n, \quad (2.3)$$

$$M = C^d \bmod n, \quad (2.4)$$

де M – це повідомлення, C – зашифрований текст, e та d є приватним і публічним ключів, n – модуль (добуток двох великих простих чисел)

2.2.2 Цілісність даних і перевірка

Хеш-функції використовуються для забезпечення цілісності даних шляхом створення унікальних хеш-значень для блоків даних. Зазвичай використовується безпечний хеш-алгоритм (SHA-256), який створює 256-бітне хеш-значення.

– Хешування: перетворює вхідні дані в хеш-значення фіксованого розміру.

– Перевірка: порівнює хеш-значення отриманих даних із вихідним хеш-значенням для виявлення змін.

Математична основа:

$$h = H(m), \tag{2.5}$$

де h – хеш-значення, H – хеш-функці, m – це повідомлення.

2.2.3 Балансування навантаження та оптимізація продуктивності

Балансування навантаження має вирішальне значення для рівномірного розподілу навантажень між серверами, щоб запобігти перевантаженню та забезпечити високу доступність. Загальні алгоритми включають:

– Кругова система: розподіляє запити послідовно між серверами.

– Зважена кругова система: призначає ваги серверам на основі їхньої потужності та відповідно розподіляє запити.

– Найменша кількість підключень: спрямовує запити на сервер із найменшою кількістю активних підключень.

Математична основа (зважена кругова система):

$$S_i = \frac{W_i}{\sum_{j=1}^{n} W_j}, \tag{2.6}$$

де S_i – частка запитів на сервер i, W_i – вага сервера, n – загальна кількість серверів.

2.2.4 Оптимізація бази даних

Індексування — це техніка оптимізації бази даних, яка використовується для прискорення обробки запитів. Індекси створюються на стовпцях, щоб полегшити операції пошуку та вилучення.

Індексування B-дерева: організовує записи індексу у збалансованій деревоподібній структурі, дозволяючи логарифмічний час пошуку.

2.2.5 Спілкування в реальному часі

Протокол WebSocket забезпечує двонаправлений зв’язок між клієнтом і сервером у реальному часі. Він підтримує постійне з’єднання, зменшуючи накладні витрати на встановлення нових з’єднань для кожного повідомлення.

– Процес рукостискання: ініціює з’єднання WebSocket за допомогою HTTP-запиту з подальшим оновленням до протоколу WebSocket.

– Кадрування повідомлень: кодує повідомлення у кадри, що забезпечує ефективну передачу даних.

Структура кадру: визначає формат для кадрів WebSocket, включаючи поля для довжини корисного навантаження, ключа маскування та даних.

2.2.6 Статистичні методи

Статистичні методи застосовуються для аналізу моделей активності користувачів, що допомагає оптимізувати ресурси сервера та покращити взаємодію з користувачем.

– Описова статистика: узагальнює дані про дії користувача за допомогою таких показників, як середнє значення, медіана та стандартне відхилення.

– Аналіз часових рядів: аналізує тенденції та закономірності в активності користувачів з часом.

Математична основа:

– Середнє значення:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad (2.7)$$

– Стандартне відхилення:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}, \quad (2.8)$$

Розробка месенджера включає різноманітні математичні методи для забезпечення безпечного, ефективного та надійного зв'язку між користувачами. Від криптографічних методів до перевірки цілісності даних, алгоритмів балансування навантаження та статистичного аналізу, ці методи складають основу функціональності системи. Застосовуючи ці математичні принципи, програма досягає високої продуктивності, надійної безпеки та оптимальної взаємодії з користувачем.

2.3 Опис використаних технологій та мов програмування

2.3.1 Опис ASP.NET Core

ASP.NET – це популярне середовище веб-розробки для створення веб-застосунків на платформі .NET.

ASP.NET Core – це версія ASP.NET з відкритим вихідним кодом, яка працює у macOS, Linux та Windows. ASP.NET Core був вперше випущений в 2016 році і є переробленою версією більш ранніх версій ASP.NET тільки для Windows.

Основні особливості ASP.NET Core:

– Сучасне міжплатформне середовище виконання, яке можна розробити та запускати в Windows, Linux, Docker і macOS.

– Покращена продуктивність завдяки новому модульному дизайну, який забезпечує краще кешування та оптимізацію.

– Спрощена розробка з уніфікованою структурою, яка поєднує MVC і Web API в єдиний конвеєр.

– Нова модель програмування Razor Pages, яка полегшує створення програм, орієнтованих на сторінки.

ASP.NET Core пропонує низку переваг порівняно з оригінальною платформою ASP.NET, включаючи покращену продуктивність, кросплатформну підтримку, підтримувану платформу та спрощену розробку. Так як ми прагнемо створити високопродуктивну веб-програму, яку можна розгортати на кількох платформах, тоді ASP.NET Core — це основа для вас [4].

Однією з найбільших переваг ASP.NET Core є його продуктивність. Завдяки меншій площі та вдосконаленій архітектурі програми ASP.NET Core можуть обробляти більше запитів на секунду та мають швидший час запуску [7].

Best plaintext responses per second, Dell R440 Xeon Gold + 10 GbE (486 tests)

Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	IA
1	gnet	7,013,961	100.0% (100.0)	0	Plt	Go	Non	Non	Lin	Rea
2	faf	7,010,014	99.9% (99.9)	0	Plt	rs	Non	Non	Lin	Rea
3	aspcore	7,006,142	99.9% (99.9)	0	Plt	C#	.NE	kes	Lin	Rea
4	pico.v	7,005,808	99.9% (99.9)	0	Mcr	v	Non	Non	Lin	Rea
5	firenio-http-lite	7,004,779	99.9% (99.9)	0	Plt	Jav	fir	Non	Lin	Rea
6	ntex [raw]	7,000,539	99.8% (99.8)	0	Plt	rs	Non	nte	Lin	Rea
7	cinatra	6,998,076	99.8% (99.8)	0	Ful	C++	Non	Non	Lin	Rea
8	wizzardo-http	6,995,129	99.7% (99.7)	0	Mcr	Jav	Non	Non	Lin	Rea
9	lithium	6,993,976	99.7% (99.7)	0	Mcr	C++	Non	Non	Lin	Rea
10	redkale-graalvm	6,981,831	99.5% (99.5)	0	Ful	Jav	red	red	Lin	Rea
11	libreactor	6,971,119	99.4% (99.4)	0	Plt	C	Non	Non	Lin	Rea
12	mormot [orm]	6,970,546	99.4% (99.4)	0	Ful	pas	Non	Non	Lin	Rea
13	just-js	6,961,689	99.3% (99.2)	0	Plt	JS	jus	Non	Lin	Rea
14	silverlining	6,947,056	99.0% (99.0)	987,930	Plt	Go	Non	Non	Lin	Rea
15	silverlining	6,936,372	98.9% (98.9)	764,589	Plt	Go	Non	Non	Lin	Rea
16	may-minihttp	6,909,552	98.5% (98.5)	0	Mcr	rs	rs	may	Lin	Rea
17	libreactor	6,907,166	98.5% (98.5)	0	Mcr	C	lib	Non	Lin	Rea
18	libsniper	6,883,107	98.1% (98.1)	0	Plt	C++	Non	Non	Lin	Rea

Рис. 2.1. Показники продуктивності різних фрейморків

ASP.NET Core замінює ASP.NET, тому це версія, яку слід використовувати, якщо ми хочемо переконатися, що наші програми працюють на підтримуваній платформі, яка постійно розробляється.

Деякі версії ASP.NET все ще отримують незначні оновлення, випуски патчів і виправлення помилок, але більше не отримають жодних основних нових функцій. Усі інновації та подальший розвиток відбуваються лише в ASP.NET Core, і тому це версія, яку варто використовувати, якщо ми хочете скористатися перевагами новітніх технологій і функцій, доступних для програм .NET.

2.3.2 Опис WebSocket API

WebSocket API — це передова технологія, яка дає змогу відкривати двосторонній інтерактивний сеанс зв’язку між браузером користувача та сервером. За допомогою цього API ви можете надсилати повідомлення на сервер і отримувати відповіді, керовані подіями, без запиту відповіді на сервері [5].

WebSockets — це протокол, стандартизований IETF як RFC 6455. Вони забезпечують двосторонній канал зв’язку через одне довготривале з’єднання TCP. Це дозволяє серверу надсилати повідомлення клієнту без запиту клієнта, і навпаки.

Основні характеристики WebSockets:

– Повно дуплексний зв’язок: дозволяє одночасний двосторонній зв’язок між клієнтом і сервером.

– Постійне з’єднання: після встановлення з’єднання залишається відкритим, що зменшує накладні витрати на встановлення кількох HTTP-з’єднань.

– Низька затримка: ідеально підходить для додатків, які потребують оновлень у реальному часі завдяки меншій затримці порівняно з традиційним опитуванням HTTP.

Використання WebSockets з ASP.NET Core забезпечує потужне рішення для веб-додатків у реальному часі, пропонуючи плавний та ефективний спосіб обробки безперервного потоку даних між клієнтом і сервером. Його інтеграція з надійними функціями ASP.NET Core робить його ідеальним вибором для створення сучасних, адаптивних і масштабованих програм [6].

2.4 Опис структури системи та алгоритмів її функціонування

Система месенджера складається з трьох основних компонентів:

– Frontend: клієнтська програма, з якою взаємодіють користувачі.

– Backend (ASP.NET Core): програма на стороні сервера, яка обробляє бізнес-логіку та зв’язок.

– БД (MS SQL Server): система зберігання даних, повідомлень і розмов користувачів.

Система обміну повідомленнями розділена на frontend і backend компоненти, кожен з яких має окремі обов’язки. Кожен компонент взаємодіє з іншими, щоб забезпечити безперебійний обмін повідомленнями, забезпечуючи надсилання та отримання повідомлень у режимі реального часу, а також безпечне керування даними користувачів. Далі наведено приклади такої взаємодії у вигляді таблиць.

Метод структурованої декомпозиції розбиває систему обміну повідомленнями на керовані компоненти, кожен з яких має певні обов’язки та взаємодію. Таблиці надають чіткий огляд функцій і обов’язків кожного компонента, гарантуючи, що система добре організована, придатна для обслуговування та масштабування. Цей підхід сприяє ефективній розробці та майбутнім удосконаленням, забезпечуючи надійний проект для створення системи обміну повідомленнями в реальному часі з використанням ASP.NET Core, WebSockets і MS SQL Server.

Таблиця 2.1

Frontend компоненти

Компонент	Функція	Обов’язки
WebSocket Клієнт	Встановлює з’єднання WebSocket	Керує спілкуванням у реальному часі
	Надсилає повідомлення	Надсилає повідомлення користувача через підключення WebSocket
	Отримує повідомлення	Слухає вхідні повідомлення та оновлюйте інтерфейс користувача
Вікно чату	Надає користувачам інтерфейс для перегляду та надсилання повідомлень	Відображення повідомлень розмови, поле введення для нових повідомлень
Список чатів	Перераховує всі бесіди, в яких бере участь користувач	Відображення розмов користувачів, можливість перемикання між розмовами
Форми автентифікації	Полегшує реєстрацію та вхід користувачів	Керує реєстрацією та входом користувачів

Таблиця 2.2

Backend компоненти

Компонент	Функція	Обов’язки
UserController	Керує реєстрацією та входом користувачів	Обробляє дії пов’язані з користувачем, перевіряє введені дані,

		автентифіковує користувачів
MessageController	Керує надсиланням і отриманням повідомлень	Керує надсиланням і отриманням повідомлень
UserService	Містить бізнес-логіку, пов’язану з користувачами	Створення користувачів, автентифікація користувачів
MessageService	Керує повідомленнями, включаючи надсилання, зберігання та отримання повідомлень	Надсилає, зберігає, отримує повідомлення
ConversationService	Керує розмовами між користувачами	Створює та відновлює розмови
ChatHub	Керує обміном повідомленнями в реальному часі за допомогою WebSockets	Надсилає й отримує повідомлення в режимі реального часу за допомогою WebSockets

Таблиця 2.3

Взаємодія між компонентами

Взаємодія	Компоненти, що залучені	Функція
WebSocket Клієнт ↔ ChatHub	WebSocket Клієнт, ChatHub	Обслуговує обмін повідомленнями в реальному часі між клієнтом і сервером

UI Компоненти ↔ Контролери	UI Компоненти, UserController, MessageController	Використовує виклики RESTful API для керування користувачами та отримання повідомлень
Контролери ↔ Сервіси	Контролери, UserService, MessageService, ConversationService	Виконує бізнес-логіку та операції з базою даних
Сервіси ↔ БД	UserService, MessageService, ConversationService, БД	Виконує операції збереження та отримання даних за допомогою Entity Framework Core

Система обміну повідомленнями покладається на ключові алгоритми для обробки взаємодії користувачів і обробки повідомлень: реєстрація користувача підтверджує та зберігає нові дані користувача, вхід користувача перевіряє облікові дані та генерує маркери автентифікації, надсилання повідомлень передбачає створення або перевірку розмов, зберігання повідомлень і сповіщення одержувачів у режимі реального часу через WebSocket, і отримання повідомлень аналізує вхідні дані та оновлює інтерфейс користувача. Ці алгоритми забезпечують безпечне керування користувачами, зв’язок у режимі реального часу та ефективну обробку повідомлень у системі обміну повідомленнями.

Алгоритм реєстрації користувача:

– Введення: електронна пошта, пароль.
– Процес:
    a. Перевірка вхідних даних, щоб переконатися, що вони відповідають вимогам.

b. Хешування паролю для безпечного зберігання.
c. Створення нову сутність користувача з електронною поштою користувача та хешованим паролем.
d. Збереження сутності користувача в базі даних.

– Вивід: Повідомлення про успіх/невдачу.

Алгоритм входу користувача:

– Введення: електронна пошта, пароль.
– Процес:
  a. Отримання сутності користувача з бази даних за дпоомогою пошти користувача.
  b. Звірення наданого пароля зі збереженим хешованим паролем.
  c. Генерування токену автентифікації, якщо пароль правильний.
– Вивід: Токен автентифікації або повідомлення про помилку.

Алгоритм надсилання повідомлень:

– Введення: id відправника, id отримувача, контент повідомлення.
– Процес:
  a. Перевірка, чи існує розмова між відправником і одержувачем. Якщо ні, створення нової розмови.
  b. Створення нової сутності повідомлення з ідентифікатором розмови, ідентифікатором відправника, ідентифікатором отримувача, вмістом і міткою часу.
  c. Збереження сутності повідомлення в базі даних.
  d. Повідомити одержувача через WebSocket про нове повідомлення.
– Вивід: Повідомлення про успіх/невдачу.

Алгоритм отримання повідомлень:

– Введення: Повідомлення WebSocket.
– Процес:

a. Перевірка, чи існує розмова між відправником і одержувачем. Якщо ні, створення нової розмови.
b. Створення нової сутності повідомлення з ідентифікатором розмови, ідентифікатором відправника, ідентифікатором отримувача, вмістом і міткою часу.
c. Збереження сутності повідомлення в базі даних.
d. Повідомити одержувача через WebSocket про нове повідомлення.

– Вивід: Повідомлення про успіх/невдачу.

2.5 Опис розробленої системи

2.5.1 Використані технічні засоби

Месенджер було розроблено і протестовано на ПК з такими характеристиками та периферією:

– процесор: Чотириядерний Intel Core vPRO i7-1165G7 (2.8 — 4.7 ГГц)
– оперативна пам’ять: DDR4-3200 МГц 32 GB
– операційна система: Windows 11 Enterprise
– тип системи: 64-розрядна операційна система, процесор x64
– накопичувач даних: SSD 470 GB
– інтегрована графіка: Intel Iris Xe Graphics
– розмір екрану: 15.6 дюймів
– роздільна здатність: Full HD (1920x1080)

2.5.2. Використані програмні засоби

Під час розробки цього застосунку були використані наступні програмні

засоби:

– Visual Studio 2022
– Microsoft SQL Server Management Studio 18
– Figma

Visual Studio 2022

Visual Studio 2022 (рис. 2.2) - це потужне середовище розробки програмного забезпечення (IDE), створене компанією Microsoft. Воно використовується для розробки різноманітних додатків, включаючи веб-додатки, мобільні додатки, хмарні сервіси та десктопні додатки.

Visual Studio 2022 є 64-бітовою програмою, що дозволяє використовувати більше оперативної пам'яті та забезпечує кращу продуктивність при роботі з великими проектами.

В ній присутня підтримка різних мов програмування, таких як C#, C++, Visual Basic, F#, Python, JavaScript, TypeScript та інших, вбудована підтримка Git для керування версіями та спільної роботи над проектами, інтеграція з GitHub та Azure DevOps та розширені можливості редагування коду, такі як IntelliSense, рефакторинг, підсвічування синтаксису та інші інструменти для підвищення продуктивності розробника [8].

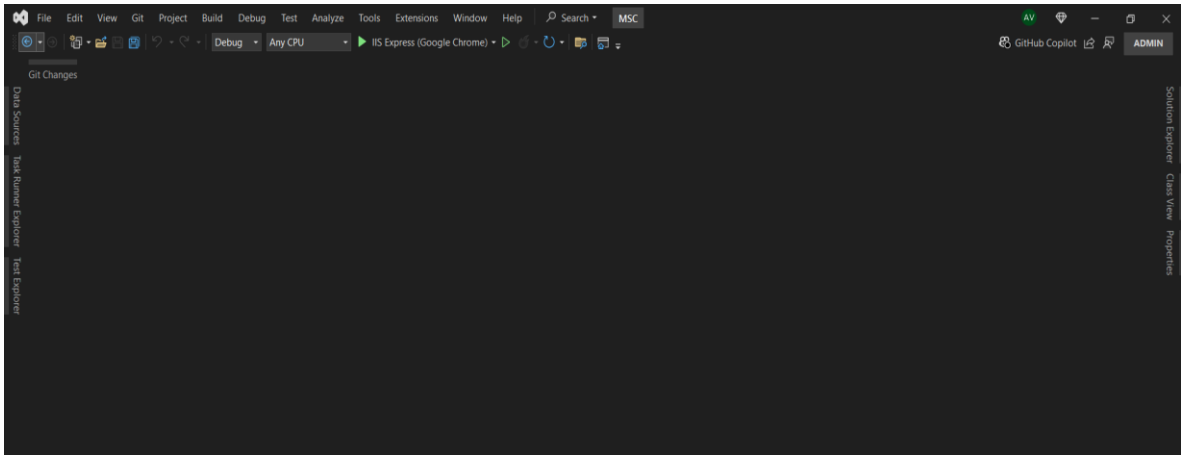

Рис. 2.2. Інтерфейс Visual Studio 2022

Microsoft SQL Server Management Studio 18

Microsoft SQL Server Management Studio (SSMS) 18 (рис. 2.3) — це інтегроване середовище для управління інфраструктурою SQL Server, яке використовується для налаштування, управління та адміністрування всіх компонентів SQL Server. Воно забезпечує доступ до інструментів для налаштування, моніторингу та адміністрування баз даних SQL Server. Ось основні характеристики та можливості SSMS 18 [9]:

– Інтуїтивно зрозумілий графічний інтерфейс користувача для адміністрування SQL Server.

– Підтримка різних видів підключень, включаючи локальні та віддалені SQL Server екземпляри.

– Розширений редактор T-SQL з підтримкою IntelliSense, підсвічуванням синтаксису та іншими функціями для полегшення написання коду.

– Інструменти для створення, редагування та управління базами даних, таблицями, індексами, процедурами, тригерами та іншими об'єктами баз даних.

– Інструменти для моніторингу продуктивності SQL Server.

– Використання SQL Server Profiler для відстеження запитів і аналізу продуктивності.

– Оптимізація продуктивності за допомогою Database Engine Tuning Advisor.

– Управління користувачами та ролями для забезпечення безпеки баз даних.

– Інструменти для налаштування безпеки SQL Server, включаючи управління дозволами та політиками безпеки.

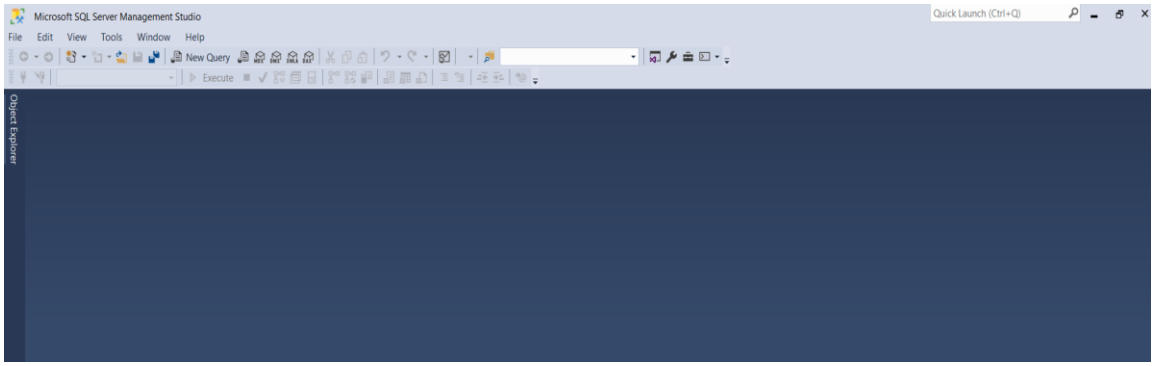

Рис. 2.3. Інтерфейс Microsoft SQL Server Management Studio 18

Figma

Figma (рис. 2.4) - векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи, що розробляється однойменною компанією. Працює у двох форматах: у браузері та як клієнтський застосунок на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач.

Сервіс є безкоштовним для індивідуальних користувачів і платним для фахових команд. Даний редактор підходить як для створення простих прототипів і дизайн-систем, так і складних проєктів (мобільні додатки, портали).

Модель розвитку Figma, спростила співробітництво в усьому процесі створення цифрових продуктів для дизайнерів, розробників, менеджерів і маркетологів дозволила на початку 2018 року залучити додаткові 25 мільйонів доларів інвестицій від партнерів.

2019 року автори редактора залучили $40 млн інвестицій від венчурного фонду Sequoia Capital при оцінці компанії в $400 млн. Рішення про інвестиції було прийнято на основі факту, що близько половини портфельних компаній фонду використовують редактор в роботі. До початку 2019 року Figma вийшла на 1 мільйон зареєстрованих користувачів, ставши серйозним конкурентом для традиційних графічних редакторів і засобів прототипування [10].

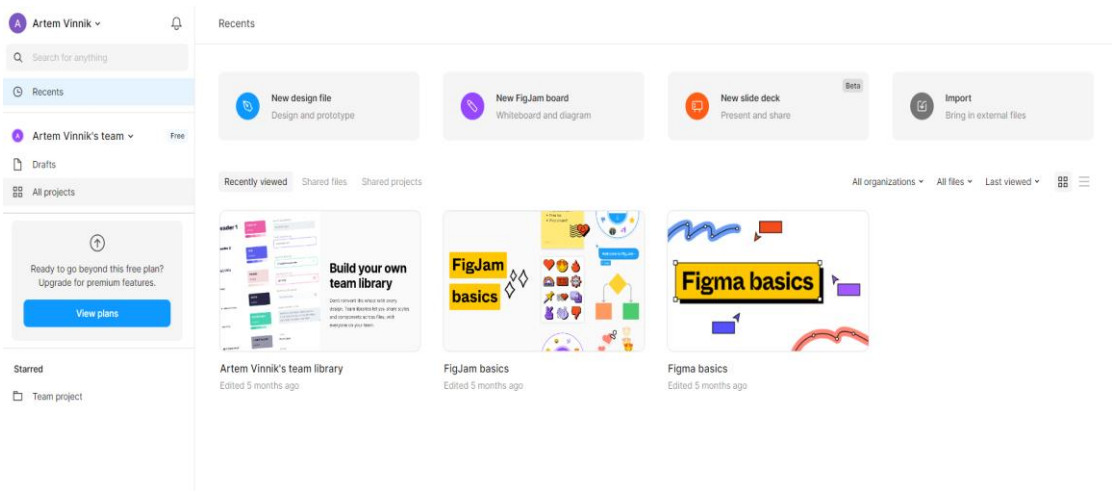

Рис. 2.4. Інтерфейс Figma

2.5.3 Виклик і завантаження програми

Щоб розгорнути веб-програму, написану за допомогою ASP.NET Core, WebSocket, MS SQL Server і SignalR, потрібно почати із підготовки середовища, наприклад Azure, AWS або виділений сервер.

Далі треба налаштувати базу даних, встановивши MS SQL Server на свій сервер або використовуючи хмарну службу SQL. Треба оновити рядок підключення програми у файлі `appsettings.json`, щоб вказувати на сервер бази даних.

Після цього треба опублікуват програму ASP.NET Core, використовуючи Visual Studio. Для цього треба клацнути правою кнопкою миші на проекті в Solution Explorer і вибрати «Опублікувати», потім обрати цільове розташування, наприклад Azure, папку або FTP, і дотримуватися підказок. У разі публікації в папці скопіюйте опубліковані файли на свій сервер.

Потрібно налаштувати сервер, переконавшись, що середовище виконання .NET Core встановлено. За потреби завантажити його з офіційного сайту .NET. Для IIS інсталювати роль веб-сервера IIS і пакет хостингу ASP.NET Core, а потім створити новий сайт у IIS і вказати його в папці програми. Увімкнути протокол

WebSocket в IIS для підтримки WebSocket і переконатися, що наші кінцеві точки SignalR правильно налаштовані та доступні.

Далі потрібно перевірити своє розгортання, щоб переконатися, що воно правильно функціонує у робочому середовищі, перевіряючи обмін повідомленнями в реальному часі та взаємодію з базою даних. Налаштуйте інструменти моніторингу для відстеження продуктивності програми та помилок, використовуючи такі рішення, як Azure Application Insights або ELK Stack. Реалізуйте журнал у вашій програмі, щоб отримувати критичну інформацію та помилки.

Нарешті, налаштуйте конвеєр безперервної інтеграції/безперервного розгортання (CI/CD) за допомогою таких інструментів, як GitHub Actions, Azure DevOps або Jenkins, щоб автоматизувати процес розгортання. Налаштуйте конвеєр для створення, тестування та розгортання вашої програми під час кожної зміни коду. Виконуючи ці дії, ви зможете успішно розгорнути свою веб-програму та забезпечити її безперебійну роботу у робочому середовищі.

2.5.4 Опис інтерфейсу користувача

Вперше зайшовши у месенджер, застосунок переведе користувача на сторінку входу, або реєстрації (рис. 2.5)

На рис. 2.5 зображена стартова сторінка веб-додатка під назвою "WebChatNow". У верхній частині сторінки розміщено заголовок з логотипом, який представляє собою блакитну іконку чату та назву "WebChatNow".

Ліва частина сторінки містить форму входу, яка складається з двох полів для введення даних: ім'я користувача (Username) та пароль (Password). Під полями є кнопка "Login" для входу до системи, а також посилання "Forgot Password?" для відновлення пароля у випадку його втрати.

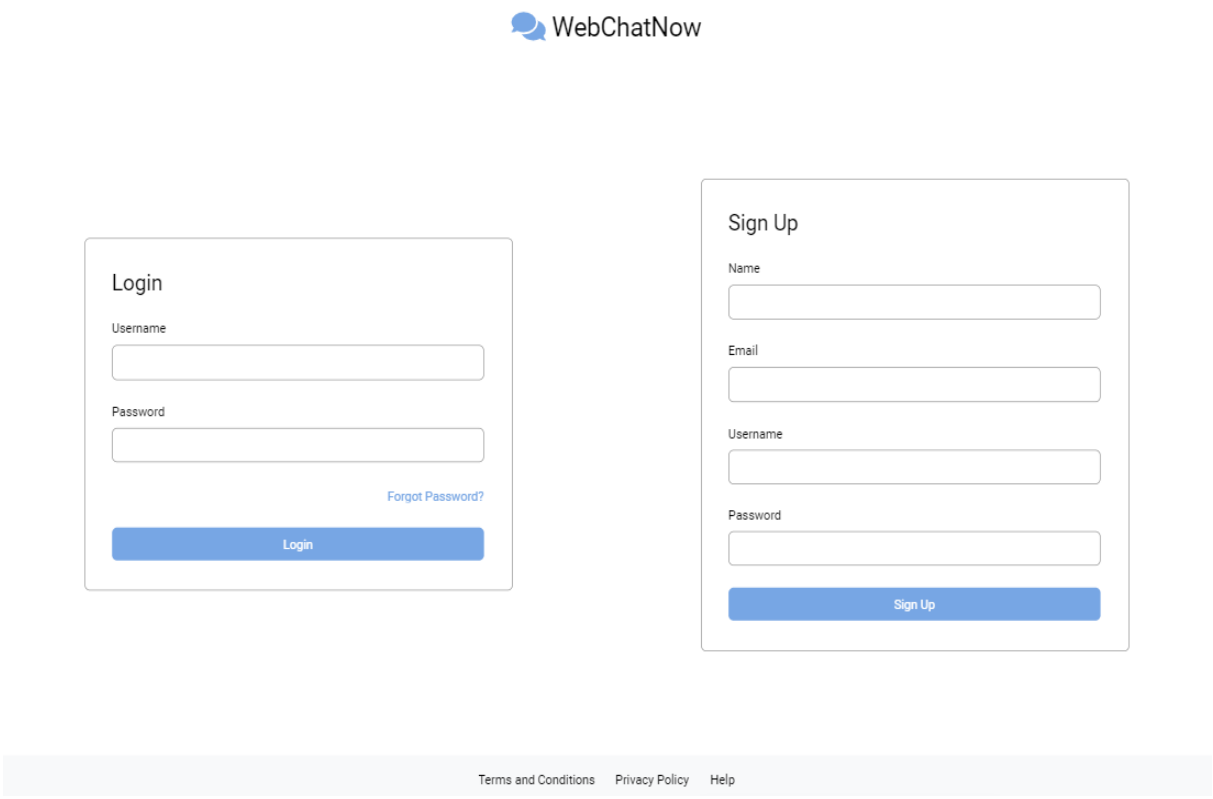

Рис. 2.5. Сторінка входу та реєстрації

Справа розташована форма реєстрації для нових користувачів. Вона включає поля для введення імені (Name), електронної пошти (Email), імені користувача (Username) та пароля (Password). Під полями знаходиться кнопка "Sign Up", яка дозволяє новим користувачам зареєструватися в системі.

У нижній частині сторінки, в області футера, розміщені посилання на важливі інформаційні розділи: "Terms and Conditions" (Умови використання), "Privacy Policy" (Політика конфіденційності) та "Help" (Допомога). Ці посилання допомагають користувачам отримати додаткову інформацію про правила використання сервісу, захист особистих даних та доступ до допомоги у разі виникнення питань.

Таким чином, стартова сторінка "WebChatNow" надає можливість новим користувачам зареєструватися, а існуючим користувачам увійти до системи, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс.

Після реєстрації, або входу у акаунт, користувача перекидує на головну сторінку (рис. 2.6).

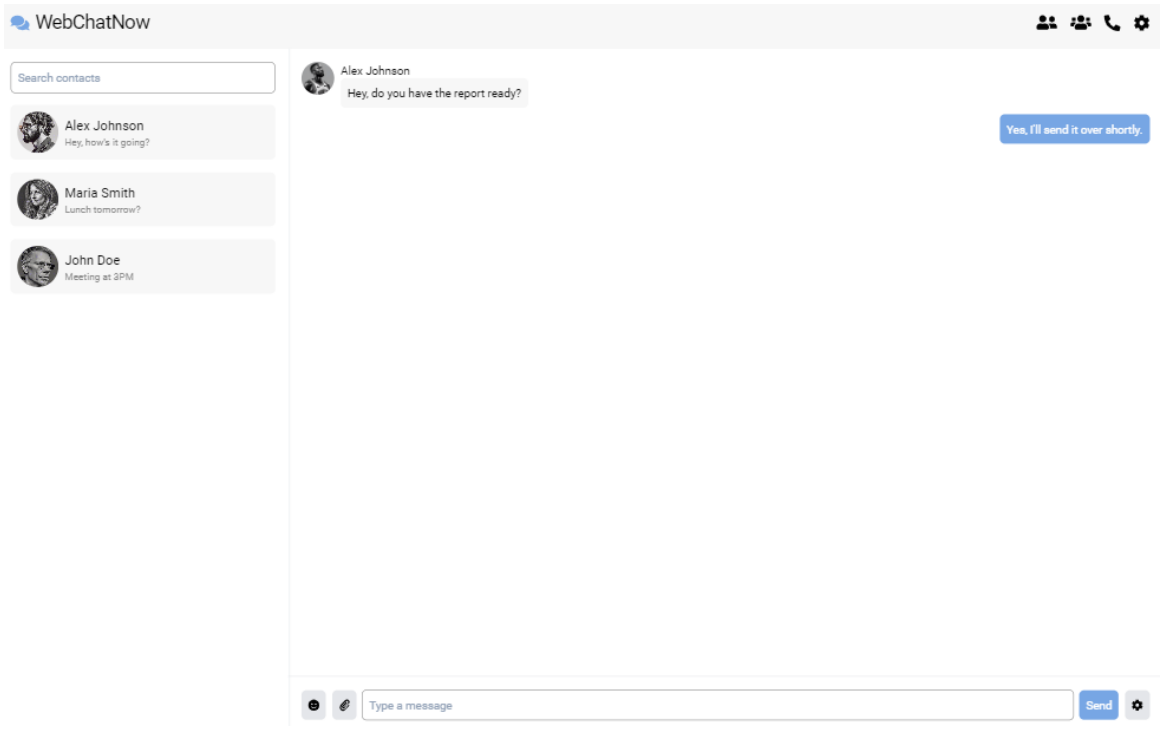

Рис. 2.6. Головна сторінка

На рис. 2.6 зображено інтерфейс веб-додатка для чату під назвою "WebChatNow". Заголовок з логотипом і назвою розташований у верхньому лівому куті. Інтерфейс поділений на дві основні частини: список контактів зліва та вікно чату праворуч.

Ліва частина інтерфейсу включає панель пошуку для швидкого знаходження контактів, а також список контактів із їхніми іменами та аватарами. Кожен контакт має короткий опис останнього повідомлення або теми обговорення. Наприклад, контакти "Alex Johnson", "Maria Smith" та "John Doe" мають підписи "Hey, how's it going?", "Lunch tomorrow?" та "Meeting at 3PM" відповідно.

Права частина інтерфейсу відображає активний чат з обраним контактом, у цьому випадку з "Alex Johnson". У верхній частині вікна чату показано аватар і ім'я контакту. У чаті відображено обмін повідомленнями: повідомлення від "Alex Johnson" з текстом "Hey, do you have the report ready?" та відповідь "Yes, I'll send it over shortly", виділена синім кольором.

У нижній частині вікна чату знаходиться панель введення повідомлень з текстовим полем для введення нового повідомлення та кнопкою "Send" для відправлення. На панелі також є іконки для додавання емодзі та налаштувань чату.

У верхньому правому куті розташовані іконки для управління додатковими функціями: список контактів, групові чати, виклики та налаштування. Ці іконки дозволяють користувачам швидко отримати доступ до різних функцій додатка.

Таким чином, інтерфейс "WebChatNow" забезпечує зручний доступ до контактів і дозволяє легко обмінюватися повідомленнями в режимі реального часу.

При натисканні на іконку налаштування застосунок переведе користувача на сторінку налаштувань (рис. 2.7).

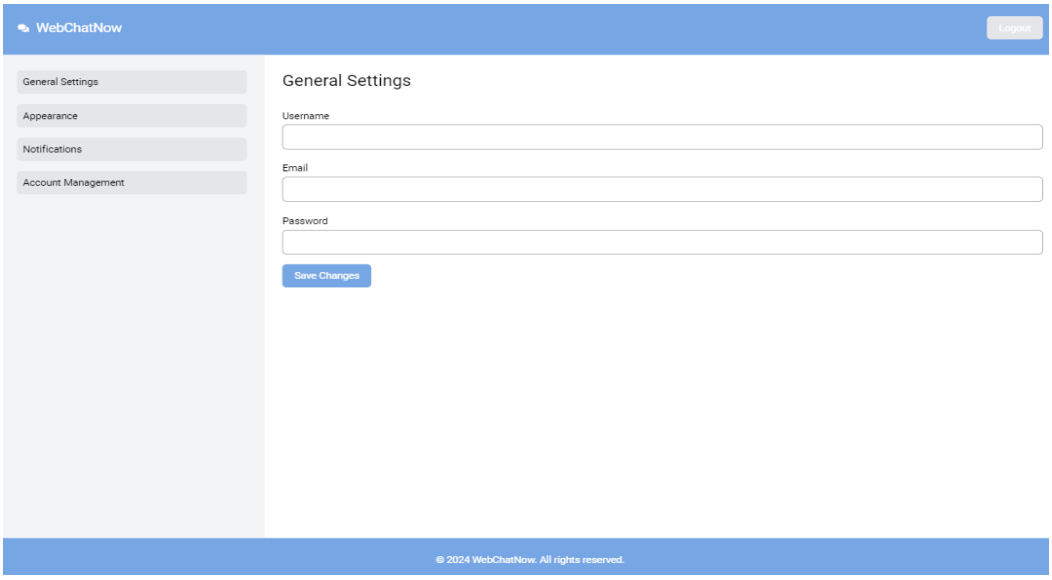

Рис. 2.7. Сторінка налаштувань

На скріншоті зображено сторінку налаштувань веб-додатка для чату під назвою "WebChatNow". Інтерфейс розділений на дві частини: ліву панель навігації та праву область змісту.

У верхній частині сторінки, ліворуч, розміщений логотип і назва додатка "WebChatNow" на синьому фоні. У правому верхньому куті є кнопка "Logout" для виходу з облікового запису.

Ліва панель навігації містить список розділів налаштувань:

– General Settings (Загальні налаштування)
– Appearance (Зовнішній вигляд)
– Notifications (Повідомлення)
– Account Management (Управління обліковим записом)

Права частина відображає вміст вибраного розділу налаштувань. У цьому випадку обрано розділ "General Settings". Вона містить три текстові поля для введення:

– Username (Ім'я користувача)
– Email (Електронна пошта)
– Password (Пароль)

Під полями знаходиться кнопка "Save Changes" (Зберегти зміни) для збереження введених даних.

У нижній частині сторінки розташований футер з інформацією про авторські права: "© 2024 WebChatNow. All rights reserved."

Загалом, ця сторінка дозволяє користувачам змінювати основні налаштування свого облікового запису, такі як ім'я користувача, електронна пошта та пароль, забезпечуючи зручний доступ до управління налаштуваннями.

2.5.6 Опис ключових моментів у коді

Початкова сторінка веб-додатка, яка включає форми для входу та реєстрації користувачів, є важливим компонентом будь-якої системи, що дозволяє користувачам взаємодіяти з додатком. У цьому дипломному проєкті була розроблена така сторінка з використанням ASP.NET Core, WebSocket та SignalR для забезпечення динамічного та інтерактивного досвіду.

Проєкт було створено за допомогою шаблону Razor Pages у середовищі ASP.NET Core, що забезпечує простий та ефективний спосіб створення веб-інтерфейсів. SignalR використовується для реалізації WebSocket-з'єднань, які дозволяють миттєво передавати дані між сервером і клієнтами.

У файлі Startup.cs було додано необхідні служби для Razor Pages та SignalR:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
public class Startup
{
   public void ConfigureServices(IServiceCollection services)
   {
      // Додавання служб Razor Pages
      services.AddRazorPages();
      // Додавання SignalR
      services.AddSignalR();
   }
   public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
   {
      if (env.IsDevelopment())
```

```
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();
        app.UseRouting();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            // Налаштування маршрутизації для Razor Pages
            endpoints.MapRazorPages();
            // Налаштування маршруту для SignalR хабу
            endpoints.MapHub<ChatHub>("/chatHub");
        });
    }
}
```

Далі створено хаб SignalR (ChatHub.cs), який дозволяє обробляти з'єднання та передавати повідомлення:

```
using Microsoft.AspNetCore.SignalR;
public class ChatHub : Hub
{
    // Метод, який буде викликатися клієнтами для відправлення повідомлень
```

```
    public async Task SendMessage(string user, string message)
    {
        await Clients.All.SendAsync("ReceiveMessage", user, message);
    }
}
```

Розроблена початкова сторінка веб-додатка WebChatNow з формою входу та реєстрації користувачів є важливим елементом системи, яка використовує сучасні технології ASP.NET Core та SignalR [14] для забезпечення інтерактивного досвіду користувачів. Завдяки використанню Razor Pages [15] та SignalR, ця сторінка може бути легко розширена для підтримки додаткових функцій, таких як сповіщення в реальному часі та інші інтерактивні можливості.

Один з основних компонентів веб-додатку для обміну повідомленнями - це інтерфейс з панеллю пошуку, списком контактів та активним чатом. Така сторінка дозволяє користувачам легко шукати контакти, переглядати наявні чати та спілкуватися в режимі реального часу.

У файлі Startup.cs виконується конфігурація служб та middleware, необхідних для роботи додатку. Конфігурація служб у методі ConfigureServices:

```
public void ConfigureServices(IServiceCollection services)
{
    // Додавання служб Razor Pages
    services.AddRazorPages();
    // Додавання SignalR
    services.AddSignalR();
    // Додавання служб автентифікації та авторизації (при необхідності)
    services.AddAuthentication();
    services.AddAuthorization();
    // Налаштування підключення до бази даних MS SQL Server
    services.AddDbContext<ApplicationDbContext>(options =>
```

```
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection
")));
        }
```

Конфігурація middleware у методі Configure:

```
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseRouting();
            app.UseAuthentication();
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>
            {
                // Налаштування маршрутизації для Razor Pages
                endpoints.MapRazorPages();
                // Налаштування маршруту для SignalR хабу
                endpoints.MapHub<ChatHub>("/chatHub");
            });
        }
```

Створення бази даних та моделей

Для зберігання даних про користувачів, контакти та повідомлення, необхідно створити моделі та контекст бази даних.

Модель користувача:

```
public class ApplicationUser : IdentityUser
{
    // Додаткові властивості користувача
    public string DisplayName { get; set; }
    public ICollection<Message> Messages { get; set; }
    public ICollection<Contact> Contacts { get; set; }
}
```

Модель повідомлення:

```
public class Message
{
    public int Id { get; set; }
    public string Content { get; set; }
    public DateTime Timestamp { get; set; }
    public string UserId { get; set; }
    public ApplicationUser User { get; set; }
    public int ChatId { get; set; }
    public Chat Chat { get; set; }
}
```

Модель чату:

```
public class Chat
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Message> Messages { get; set; }
```

```
    public ICollection<Contact> Contacts { get; set; }
}
```

Контекст бази даних:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
    public DbSet<Message> Messages { get; set; }
    public DbSet<Chat> Chats { get; set; }
    public DbSet<Contact> Contacts { get; set; }
}
```

Створення SignalR хабу

Хаб SignalR (ChatHub.cs) забезпечує обробку з'єднань та передачу повідомлень між клієнтами.

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;
public class ChatHub : Hub
{
    private readonly ApplicationDbContext _context;
    public ChatHub(ApplicationDbContext context)
    {
        _context = context;
    }
    // Метод для відправлення повідомлення
    public async Task SendMessage(string userId, string messageContent)
```

```
        {
            var user = await _context.Users.FindAsync(userId);
            if (user == null)
            {
                // Обробка випадку, коли користувача не знайдено
                return;
            }
            var message = new Message
            {
                Content = messageContent,
                Timestamp = DateTime.Now,
                UserId = user.Id
            };
            // Збереження повідомлення в базі даних
            _context.Messages.Add(message);
            await _context.SaveChangesAsync();
            // Відправка повідомлення всім клієнтам
            await Clients.All.SendAsync("ReceiveMessage", user.DisplayName,
messageContent);
        }
        // Метод для підключення до чату
        public async Task JoinChat(string chatId)
        {
            await Groups.AddToGroupAsync(Context.ConnectionId, chatId);
        }
        // Метод для відключення від чату
        public async Task LeaveChat(string chatId)
        {
```

```
            await Groups.RemoveFromGroupAsync(Context.ConnectionId, chatId);
        }
    }
```

Реалізація контролерів та API для взаємодії з клієнтами

Для взаємодії з клієнтами через API було створено контролери, що забезпечують реєстрацію, автентифікацію та управління чатами.

Контролер для управління користувачами:

```
    using Microsoft.AspNetCore.Mvc;
    using System.Threading.Tasks;
    [Route("api/[controller]")]
    [ApiController]
    public class AccountController : ControllerBase
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;
        public AccountController(UserManager<ApplicationUser> userManager, SignInManager<ApplicationUser> signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }
        [HttpPost("register")]
        public async Task<IActionResult> Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser { UserName = model.Username, Email = model.Email, DisplayName = model.Name };
```

```
            var result = await _userManager.CreateAsync(user, model.Password);
            if (result.Succeeded)
            {
                await _signInManager.SignInAsync(user, isPersistent: false);
                return Ok();
            }
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }
        }
        return BadRequest(ModelState);
    }
    [HttpPost("login")]
    public async Task<IActionResult> Login(LoginModel model)
    {
        if (ModelState.IsValid)
        {
            var result = await _signInManager.PasswordSignInAsync(model.Username, model.Password, isPersistent: false, lockoutOnFailure: false);
            if (result.Succeeded)
            {
                return Ok();
            }
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
        }
        return BadRequest(ModelState);
```

```
    }
    [HttpPost("logout")]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return Ok();
    }
}
```

Моделі для реєстрації та входу:

```
public class RegisterModel
{
    public string Name { get; set; }
    public string Email { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
}
public class LoginModel
{
    public string Username { get; set; }
    public string Password { get; set; }
}
```

Контролер для управління чатами:

```
using Microsoft.AspNetCore.Mvc;
using System.Linq;
using System.Threading.Tasks;
[Route("api/[controller]")]
[ApiController]
public class ChatController : ControllerBase
```

```
{
    private readonly ApplicationDbContext _context;
    public ChatController(ApplicationDbContext context)
    {
        _context = context;
    }
    [HttpGet("contacts")]
    public IActionResult GetContacts()
    {
        var contacts = _context.Users.Select(user => new { user.Id, user.DisplayName }).ToList();
        return Ok(contacts);
    }
    [HttpGet("messages/{chatId}")]
    public IActionResult GetMessages(int chatId)
    {
        var messages = _context.Messages
            .Where(m => m.ChatId == chatId)
            .Select(m => new { m.User.DisplayName, m.Content, m.Timestamp })
            .ToList();
        return Ok(messages);
    }
    [HttpPost("send")]
    public async Task<IActionResult> SendMessage(SendMessageModel model)
    {
        var user = await _context.Users.FindAsync(model.UserId);
        if (user == null)
        {
```

```
            return NotFound();
        }
        var message = new Message
        {
            Content = model.Content,
            Timestamp = DateTime.Now,
            UserId = user.Id,
            ChatId = model.ChatId
        };
        _context.Messages.Add(message);
        await _context.SaveChangesAsync();
        // Відправка повідомлення всім клієнтам через хаб
        var chatHub = new ChatHub(_context);
        await chatHub.SendMessage(user.Id, model.Content);
        return Ok();
    }
}
public class SendMessageModel
{
    public string UserId { get; set; }
    public string Content { get; set; }
    public int ChatId { get; set; }
}
```

Розроблена бекенд частина веб-додатку забезпечує основні функціональні можливості для автентифікації користувачів, управління чатами та повідомленнями. Використання ASP.NET Core та SignalR дозволяє створити масштабований та ефективний додаток для обміну повідомленнями в режимі

реального часу. Завдяки чіткій структурі та використанню сучасних технологій, розроблений додаток легко розширюється та підтримує високу продуктивність.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості розробки програмного забезпечення

Початкові дані:

1. Передбачуване число операторів програми – 3222;
2. Коефіцієнт складності програми – 1,5;
3. Коефіцієнт корекції програми в ході її розробки – 0,1;
4. Годинна заробітна плата fullstack .NET розробника – 442 грн/год.[11]
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. Коефіцієнт кваліфікації програміста, залежний від стажу роботи – 1,2;
7. Вартість машино-години ЕОМ – 0,76 грн/год. Для розробки месенджера для кваліфікаційної роботи було використано ноутбук, який споживає електроенергію та використовує постійне підключення до інтернету. Щомісячна плата за інтернет від провайдера «Фрегат» становить 225 грн/місяць. Ноутбук при роботі споживає 60Вт. На розробку месенджера було витрачено близько 450 годин впродовж двох місяців. Згідно тарифного плану кВт/год коштує 4,32 грн з урахуванням ПДВ. Маючи всі дані можна порахувати, що під час роботи ноутбук за годину споживав 60 Вт і по тарифу це виходить 0,26 грн/год. Вартість інтернету на годину становить 0,5 грн/год, яку було отримано поділивши місячну плату за інтернет на кількість годин проведених за роботою в одному місяці. Загальна вартість машино-години вийшла 0,76 грн/год.[12-13]

Трудомісткість розробки ПЗ можна розрахувати за допомогою формули:

$$t = t_0 + t_u + t_а + t_n + t_{отл} + t_д\,, \qquad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

$t_п$ - витрати праці на програмування по готовій блок-схемі;

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

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p)\,, \tag{3.2}$$

де q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

Після підстановки значень в формулу (3.2) було отримано такий результат:

$$Q = 3222 \cdot 1{,}5 \cdot (1 + 0{,}1) = 5316{,}3$$

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

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}\,, \tag{3.3}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

Після підстановки значень в формулу (3.3) було отримано такий результат:

$$t_u = \frac{5316{,}3 \cdot 1{,}3}{76 \cdot 1{,}2} = 76\text{ , людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20..25) \cdot k}\,, \tag{3.4}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Після підстановки значень в формулу (3.4) було отримано такий результат:

$$t_{\text{а}} = \frac{5316,3}{21 \cdot 1,2} = 211 \text{ , людино-годин.}$$

Витрати на складання програми по готовій блок-схемі визначається за формулою:

$$t_n = \frac{Q}{(20..25) \cdot \text{k}}, \tag{3.5}$$

Після підстановки значень в формулу (3.5) було отримано такий результат:

$$t_n = \frac{5316,3}{23 \cdot 1,2} = 193 \text{ , людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5) \cdot \text{k}}, \tag{3.6}$$

Після підстановки значень в формулу (3.6) було отримано такий результат:

$$t_{\text{отл}} = \frac{5316,3}{4 \cdot 1,2} = 1108 \text{ , людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1,5 \cdot t_{\text{отл}} \text{ ,} \tag{3.7}$$

Після підстановки значень в формулу (3.7) було отримано такий результат:

$$t_{\text{отл}}^{k} = 1,5 \cdot 1108 \ = 1662 \text{ , людино-годин.}$$

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}} \text{ ,} \tag{3.8}$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису;

$$t_{\text{др}} = \frac{Q}{(15..20) \cdot \text{k}}, \tag{3.9}$$

Після підстановки значень в формулу (3.9) було отримано такий результат:

$$t_{\text{др}} = \frac{5316,3}{16 \cdot 1,2} = 277 \text{ , людино-годин.}$$

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

$$t_{до} = 0{,}75 \cdot t_{др}\ , \tag{3.10}$$

Після підстановки значень в формулу (3.10) було отримано такий результат:

$$t_{до} = 0{,}75 \cdot 277 = 208\ ,$$ людино-годин.

В результаті за формулою (3.8) було отримано такий результат:

$$t_{д} = 277 + 208 = 485\ ,$$ людино-годин.

Повертаючись до формули (3.1), було отримано повну оцінку трудомісткості розробки ПЗ:

$t = 50 + 76 + 211 + 193 + 1108 + 485 = 2123$, людино-годин.

За результатами розрахунків, загальна трудомісткість розробки даного програмного застосунку складає 2123 людино-годин.

3.2. Розрахунок витрат на створення програмного забезпечення

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:

$$K_{ПО} = З_{ЗП} + З_{МВ}\ , \tag{3.11}$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}\ , \tag{3.12}$$

де t - загальна трудомісткість, людино-годин;

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

Після підстановки значень в формулу (3.12) було отримано такий результат:

$$З_{ЗП} = 2123 \cdot 442 = 938366$$ грн.

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

$$З_{МВ} = t_{отл} \cdot C_{мч}\ , \tag{3.13}$$

де $t_{\text{отл}}$– трудомісткість налагодження програми на ЕОМ, год;

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

Після підстановки значень в формулу (3.13) було отримано такий результат:

$$З_{\text{МВ}} = 1108 \cdot 0{,}76 \ = \ 842{,}08 \text{ грн.}$$

Звідси за формулою (3.11) розраховуємо витрати на створення ПЗ:

$$K_{ПО} = \ 938366 \ + \ 842{,}08 = 939208{,}08 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{\text{t}}{B_k \cdot \text{F}_p}, \tag{3.14}$$

де B_k – число виконавців (один розробник був відповідальний за розробку);

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні

$F_p \ = \ 176$ годин).

Після підстановки значень в формулу (3.14) було отримано такий результат:

$T = \frac{2123}{1 \cdot 176} = 12$ міс.

Висновок: за підрахунками для розробки месенджера знадобиться приблизно 2123 людино-години. З урахуванням середньої зарплати fullstack .NET розробника для розробки програмного забезпечення знадобиться 939208,08 грн. Приблизний час який знадобиться на розробку складає 12 місяців при роботі одного програміста, зі стажем роботи 4 роки та тривалістю робочого часу 40 годин на тиждень.

ВИСНОВКИ

Відповідно завданню і метою кваліфікаційної роботи, розроблено месенджер, що може використовуватися для комунікації між користувачами.

Проект був зосереджений на створенні надійної, масштабованої та ефективної системи, здатної підтримувати зв’язок у реальному часі та безпечне керування даними.

Завдяки застосуванню ASP.NET Core серверні служби були ефективно структуровані, забезпечуючи чіткий розподіл проблем і дозволяючи безперебійну обробку автентифікації користувачів, обробки повідомлень і керування розмовами. Використання технології WebSocket полегшило можливості обміну повідомленнями в режимі реального часу, забезпечуючи миттєвий зв’язок між користувачами та покращуючи загальний досвід користувача. MS SQL Server використовувався для збереження даних, забезпечуючи надійне та безпечне зберігання даних користувача, повідомлень і розмов.

Архітектура системи була ретельно спланована та реалізована з чітким розподілом інтерфейсних і бекенд-компонентів. Ключові алгоритми реєстрації користувачів, входу в систему, надсилання та отримання повідомлень були розроблені для забезпечення безпечного керування користувачами та ефективної обробки повідомлень. Ці алгоритми були розроблені для перевірки, автентифікації, спілкування в реальному часі та оновлень інтерфейсу користувача, забезпечуючи безперебійну та інтерактивну роботу користувача.

Загалом проект продемонстрував повне розуміння сучасних практик веб-розробки, включаючи комунікаційні технології в реальному часі та принципи безпечного керування даними. Отримана система обміну повідомленнями є свідченням ефективного використання ASP.NET Core, WebSocket і MS SQL Server у створенні функціональної та надійної комунікаційної платформи. Цей

досвід дав цінну інформацію та практичні навички, необхідні для майбутніх зусиль у сфері розробки веб-додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Використання месенджерів. URL: https://intelmag.com/digitalization/17454-vykorystannya-mesendzheriv-yak-elementiv-czyfrovoyi-rozvidky-problematyka-ta-shlyahy-vyrishennya/ (дата звернення: 20.05.2024)

2. Соціальне дослідження. URL: https://www.oporaua.org/viyna/sotsiologichne-doslidzhennia-demokratiia-prava-i-svobodi-gromadian-ta-mediaspozhivannia-v-umovakh-viini-24261 (дата звернення: 20.05.2024)

3. Порівняння Telegram та WhatsApp. URL: https://dexatel.com/blog/telegram-vs-whatsapp/ (дата звернення: 02.06.2024)

4. Документація по ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0 (дата звернення: 05.06.2024)

5. Підтримка WebSockets у ASP.NET Core. URL: https://learn.microsoft.com/ru-ru/aspnet/core/fundamentals/websockets?view=aspnetcore-8.0 (дата звернення: 07.06.2024)

6. WebSockets. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 07.06.2024)

7. Статистика продуктивності різних фрейморків. URL: https://www.techempower.com/benchmarks/#hw=ph&test=plaintext&section=data-r22 (дата звернення: 19.06.2024)

8. Visual Studio 2022. URL: https://visualstudio.microsoft.com/ (дата звернення: 26.06.2024)

9. Microsoft SQL Server Management Studio 18 URL: https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16 (дата звернення: 26.06.2024)

10. Figma. URL: https://uk.wikipedia.org/wiki/Figma (дата звернення: 26.06.2024)

11. Середня зарплата .NET розробника [Електронний ресурс] URL: https://jobs.dou.ua/salaries/?period=2023-12&position=Middle%20SE&technology=C#/.NET&experience=0-2 (дата звернення: 15.06.2024).

12. Офіційний курс гривні щодо долара США. Національний банк України. [Електронний ресурс] URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD (дата звернення: 15.06.2024).

13. Тариф на електроенергію. Yasno. [Електронний ресурс] URL: https://yasno.com.ua/b2c-tariffs (дата звернення: 15.06.2024)

14. SignalR. URL: https://learn.microsoft.com/ru-ru/aspnet/signalr/overview/getting-started/introduction-to-signalr (дата звернення: 13.06.2024)

15. Razor Pages. URL: https://learn.microsoft.com/ru-ru/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio (дата звернення: 13.06.2024)

16. API. URL: https://hostiq.ua/blog/ukr/what-is-api/ (дата звернення: 02.06.2024)

17. Telegram. URL: https://web.telegram.org/ (дата звернення: 02.06.2024)

18. WhatsApp. URL: https://www.whatsapp.com/ (дата звернення: 02.06.2024)

19. Що таке SQL-ін’єкція? URL: https://foxminded.ua/sql-iniektsii/ (дата звернення: 07.06.2024)

20. CSRF. URL: https://owasp.org/www-community/attacks/csrf (дата звернення: 07.06.2024)

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

webChatNow.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f8f9fa;
    margin: 0;
    padding: 0;
}

.container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #f8f9fa;
}

.row {
    display: flex;
    width: 80%;
    background-color: white;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
    overflow: hidden;
}

.col-3 {
    flex: 1;
    background-color: #f1f1f1;
    padding: 20px;
}

.col-9 {
```

```
    flex: 3;
    padding: 20px;
}

h2 {
    font-size: 24px;
    margin-bottom: 20px;
}

.list-group {
    list-style-type: none;
    padding: 0;
}

.list-group-item {
    padding: 10px 20px;
    margin-bottom: 10px;
    border-radius: 5px;
    background-color: #fff;
    border: 1px solid #ddd;
    cursor: pointer;
    text-decoration: none;
    color: #333;
    display: flex;
    align-items: center;
}

    .list-group-item:hover {
        background-color: #e9ecef;
    }

    .list-group-item.active {
        background-color: #007bff;
        color: white;
    }
```

```
.form-group {
    margin-bottom: 20px;
}

    .form-group label {
        display: block;
        font-weight: bold;
        margin-bottom: 5px;
    }

.form-control {
    width: 100%;
    padding: 10px;
    font-size: 16px;
    border: 1px solid #ced4da;
    border-radius: 5px;
    box-sizing: border-box;
}

.btn {
    display: inline-block;
    padding: 10px 20px;
    font-size: 16px;
    color: white;
    background-color: #007bff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    text-align: center;
}

    .btn:hover {
        background-color: #0056b3;
    }
```

```
header {
    background-color: #007bff;
    padding: 10px 20px;
    color: white;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

    header h1 {
        margin: 0;
        font-size: 24px;
    }

    header .logout-btn {
        background-color: #ffffff;
        color: #007bff;
        border: 1px solid #007bff;
        padding: 5px 10px;
        border-radius: 5px;
        text-decoration: none;
        font-size: 16px;
    }

        header .logout-btn:hover {
            background-color: #e9ecef;
        }

footer {
    background-color: #007bff;
    color: white;
    text-align: center;
    padding: 10px 20px;
    position: fixed;
```

```
    bottom: 0;
    width: 100%;
}

    footer p {
        margin: 0;
    }
```

AccountController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using WebChatNow.EF;
using WebChatNow.Entities;

namespace WebChatNow.Controllers
{
    public class AccountController : Controller
    {
        private readonly ApplicationContext _context;

        public AccountController(ApplicationContext context)
        {
            _context = context;
        }

        [HttpGet]
        public IActionResult Login()
        {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Login(string username, string password)
        {
            var user = await _context.Users.FirstOrDefaultAsync(u => u.Username == username && u.Password
== password);
```

```
            if (user != null)
            {
                // Handle successful login
                return RedirectToAction("Index", "Home");
            }
            ModelState.AddModelError("", "Invalid username or password");
            return View();
        }

        [HttpGet]
        public IActionResult Register()
        {
            return View();
        }

        [HttpPost]
        public async Task<IActionResult> Register(User user)
        {
            if (ModelState.IsValid)
            {
                _context.Add(user);
                await _context.SaveChangesAsync();
                return RedirectToAction("Login");
            }
            return View();
        }
    }
}
```

SettingsController.cs

```
using Microsoft.AspNetCore.Mvc;
using WebChatNow.EF;
using WebChatNow.Entities;

namespace WebChatNow.Controllers
{
```

```
public class SettingsController : Controller
{
    private readonly ApplicationContext _context;

    public SettingsController(ApplicationContext context)
    {
        _context = context;
    }

    [HttpGet]
    public async Task<IActionResult> Update(int id)
    {
        // Отримайте поточного користувача (для прикладу UserId = 1)
        var user = await _context.Users.FindAsync(id);
        if (user == null)
        {
            return NotFound();
        }
        return View(user);
    }

    [HttpPost]
    public async Task<IActionResult> Update(User user)
    {
        if (!ModelState.IsValid)
        {
            return View(user);
        }

        var userToUpdate = await _context.Users.FindAsync(user.Id); // Отримайте поточного користувача
        if (userToUpdate == null)
        {
            return NotFound();
        }
```

```
            userToUpdate.Username = user.Username;
            userToUpdate.Email = user.Email;
            userToUpdate.Password = user.Password;

            await _context.SaveChangesAsync();

            return RedirectToAction(nameof(Index));
        }
    }
}
```

HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using WebChatNow.Models;

namespace WebChatNow.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }
```

```
        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier
});
        }
    }
}
```

ApplicationContext.cs

```
using Microsoft.EntityFrameworkCore;
using WebChatNow.Entities;

namespace WebChatNow.EF
{
    public class ApplicationContext : DbContext
    {
        public DbSet<User> Users { get; set; }
        public DbSet<Message> Messages { get; set; }

        public ApplicationContext(DbContextOptions<ApplicationContext> options) : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

ConnectionStrings.cs

```
namespace WebChatNow.EF
{
    public class ConnectionStrings
    {
```

```
        public string DefaultConnection { get; set; }
    }
}
```

Message.cs

```
namespace WebChatNow.Entities
{
    public class Message
    {
        public int Id { get; set; }
        public int SenderId { get; set; }
        public int ReceiverId { get; set; }
        public string Content { get; set; }
        public DateTime Timestamp { get; set; }
    }
}
```

User.cs

```
namespace WebChatNow.Entities
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
    }
}
```

ChatHub.cs

```
using Microsoft.AspNetCore.SignalR;
using System.Security.Cryptography;
using System.Text;
using WebChatNow.EF;
using WebChatNow.Entities;

namespace WebChatNow.Hubs
{
```

```
public class ChatHub : Hub
{
    private readonly ApplicationContext _context;
    private readonly string secretKey = "your-secret-key"; // Замість цього використовуйте безпечний ключ

    public ChatHub(ApplicationContext context)
    {
        _context = context;
    }

    public async Task SendMessage(string user, string encryptedMessage)
    {
        string decryptedMessage = DecryptString(encryptedMessage, secretKey);

        // Зберігання зашифрованого повідомлення в базі даних
        var message = new Message
        {
            SenderId = GetUserByUsername(user).Id,
            Content = encryptedMessage,
            Timestamp = DateTime.UtcNow
        };
        _context.Messages.Add(message);
        await _context.SaveChangesAsync();

        await Clients.All.SendAsync("ReceiveMessage", user, encryptedMessage);
    }

    private User GetUserByUsername(string username)
    {
        return _context.Users.FirstOrDefault(u => u.Username == username);
    }

    private string DecryptString(string cipherText, string key)
    {
```

```
            byte[] iv = new byte[16];
            byte[] buffer = Convert.FromBase64String(cipherText);

            using (Aes aes = Aes.Create())
            {
                aes.Key = Encoding.UTF8.GetBytes(key);
                aes.IV = iv;
                ICryptoTransform decryptor = aes.CreateDecryptor(aes.Key, aes.IV);

                using (MemoryStream ms = new MemoryStream(buffer))
                {
                    using (CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read))
                    {
                        using (StreamReader reader = new StreamReader(cs))
                        {
                            return reader.ReadToEnd();
                        }
                    }
                }
            }
        }
    }
```

Login.cshtml

```
@{
    ViewData["Title"] = "Login";
}

<h2>Login</h2>

<form asp-action="Login" method="post">
    <div>
        <label>Username</label>
        <input type="text" name="username" />
    </div>
```

```
    <div>
        <label>Password</label>
        <input type="password" name="password" />
    </div>
    <div>
        <input type="submit" value="Login" />
    </div>
</form>
```

Register.cshtml

```
@{
    ViewData["Title"] = "Register";
}

<h2>Sign Up</h2>

<form asp-action="Register" method="post">
    <div>
        <label>Name</label>
        <input type="text" name="Name" />
    </div>
    <div>
        <label>Email</label>
        <input type="text" name="Email" />
    </div>
    <div>
        <label>Username</label>
        <input type="text" name="Username" />
    </div>
    <div>
        <label>Password</label>
        <input type="password" name="Password" />
    </div>
    <div>
        <input type="submit" value="Sign Up" />
    </div>
```

```
</form>
```

Update.cshtml

```
@model WebChatNow.Models.User

@{
    ViewData["Title"] = "Settings";
}

<div class="container mt-5">
    <div class="row">
        <div class="col-3">
            <div class="list-group">
                <a href="#" class="list-group-item list-group-item-action active">General Settings</a>
                <a href="#" class="list-group-item list-group-item-action">Appearance</a>
                <a href="#" class="list-group-item list-group-item-action">Notifications</a>
                <a href="#" class="list-group-item list-group-item-action">Account Management</a>
            </div>
        </div>
        <div class="col-9">
            <h2>General Settings</h2>
            <form asp-action="Update" method="post">
                <div class="form-group">
                    <label for="Username">Username</label>
                    <input type="text" class="form-control" id="Username" name="Username"
value="@Model.Username">
                </div>
                <div class="form-group">
                    <label for="Email">Email</label>
                    <input type="email" class="form-control" id="Email" name="Email" value="@Model.Email">
                </div>
                <div class="form-group">
                    <label for="Password">Password</label>
                    <input type="password" class="form-control" id="Password" name="Password"
value="@Model.Password">
                </div>
```

```
            <button type="submit" class="btn btn-primary">Save Changes</button>
        </form>
    </div>
  </div>
</div>
```

ДОДАТОК Б

ВІДГУК

Керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

«Розробка месенджеру для організації листування між користувачами з використанням ASP.NET Core, WebSocket, MS SQL Server.»

Студента групи 122-20-3 Вінніка Артема Вадимовича

Керівник економічного розділу

доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файла	Опис
Пояснювальні документи	
Кваліфікаційна робота_Віннік.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота_Віннік.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Diplom.zip	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація_Віннік.pptx	Презентація кваліфікаційної роботи