МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного

забезпечення, канд.техн.наук, доцент,

_____ Є. О. Давиденко

«___»_____2022р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ВЕБЗАСТОСУНОК ОБМІНУ ПОВІДОМЛЕННЯМИ НА ОСНОВІ REST API ТА WEBSOCKET

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408. 21810824

Студент _____ Р. В. Скрипнік

«__»_____2022р.

Керівник ст. викладач _____ С. В. Дворецька

«__»_____2022р.

Консультант канд. техн. наук _____ А. О. Алексєєва

«__»_____2022р.

Миколаїв – 2022

ЗМІСТ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри інженерії
програмного забезпечення,
канд.техн.наук, доцент,
_____Є.О. Давиденко

«____»_____2022 р.

ЗАВДАННЯ

на виконання роботи бакалавра

Видано студенту групи _408_ факультету комп'ютерних наук

_____Скрипнік Роман Володимирович_____
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

«Вебзастосунок обміну повідомленнями на основі REST API та Websocket»_____.

Затверджена наказом по ЧНУ від «01» _____грудня_____ 2022 р. № 314

2. Строк представлення кваліфікаційної роботи «____» _____2022 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Вхідні дані до роботи – функціональні та нефункціональні вимоги до вебзастосунка обміну повідомленнями на основі REST API та Websocket. Результат – вебзастосунок обміну повідомленнями на основі REST API та Websocket_____.

4. Перелік питань, що підлягають розробці

- Аналіз сучасних засобів реалізації вебзастосунка обміну повідомленнями;
- Розробка вимог до вебзастосунка обміну повідомленнями;
- Аналіз ринку наявних технологій і рішень для вирішення завдань;

– Проектування структури бази даних та принцип роботи вебзастосунка обміну повідомленнями на основі REST API та Websocket;
– Реалізація, тестування та налагодження вебзастосунку.

5. Перелік графічних матеріалів:

Презентація_____.

6. Завдання до спеціальної частини

Дослідження питань охорони праці, які безпосередньо пов'язані з діяльністю розробника програмного забезпечення_____.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алєксєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи _____ст. викладач Дворецька Світлана Володимирівна_____
(посада, прізвище, ім'я, по батькові)

_____
(*підпис*)

Завдання прийнято до виконання

_____Скрипнік Роман Володимирович_____
(прізвище, ім'я, по батькові студента)

_____
(*підпис*)

Дата видачі завдання «_____»_____2022 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання кваліфікаційної робити

Тема: «Вебзастосунок обміну повідомленнями на основі REST API та Websocket» _____.

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	20.01.2022	24.01.2022	Виконано
2.	Огляд літератури за темою роботи	23.01.2022	29.01.2022	Виконано
3.	Складання календарного плану КРБ	25.02.2022	03.03.2022	Виконано
4.	Аналіз сучасних засобів реалізації процесу дистанційного навчання	05.02.2022	15.02.2022	Виконано
5.	Розробка вимог до системи дистанційного навчання	22.02.2022	02.03.2022	Виконано
6.	Аналіз ринку наявних технологій і рішень для вирішення завдань	03.03.2022	11.03.2022	Виконано
7.	Проектування структури бази даних та принцип роботи вебзастосунка обміну повідомленнями на основі REST API та Websocket	04.03.2022	20.03.2022	Виконано
8.	Реалізація, тестування та налагодження вебзастосунку	23.03.2022	13.05.2022	Виконано
9.	Розробка спеціальної частини з охорони праці	15.05.2022	12.06.2022	Виконано
10.	Відгук керівника КРБ	14.06.2022	14.06.2022	Виконано
11.	Оформлення КРБ та презентації	24.01.2022	19.05.2022	Виконано
12.	Попередній захист	20.05.2022	08.06.2022	Виконано
13.	Рецензування	15.06.2022	15.06.2022	Виконано
14.	Завершення оформлення КРБ та презентації	15.06.2022	21.06.2022	Виконано
15.	Захист кваліфікаційної роботи	22.06.2022	22.06.2022	Виконано

Розробив студент __Скрипнік Роман Володимирович__ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

«____»_____2022р.

Керівник роботи ст. викладач Дворецька Світлана Володимірівна
_____
(посада, прізвище, ім'я, по батькові) (*підпис*)

«____»_____2022 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Вебзастосунок обміну повідомленнями на основі REST API та Websocket»

Студент 408 гр.: Скрипнік Роман Володимирович

Керівник: ст.викладач Дворецька С. В.

Тема кваліфікаційної роботи має актуальність, оскільки пандемія коронавіруса та вторгення Російської Федерації на територію України вплинули на можливість громадян обмінюватись інформацією між собою.

Метою кваліфікаційної роботи є підвищення зручності процесу обміну онлайн повідомленнями шляхом розробки вебзастосунку основі REST API та websocket-ів.

Об’єктом дослідження є процес обміну повідомленнями між користувачами у режимі реального часу.

Предметом дослідження є інформаційні технології, методи та інструментальні засоби розробки вебзастосунків обміну повідомленнями у режимі реального часу.

Для досягнення поставленної мети необхідно розв’язати наступні завдання:

– проаналізувати сучасні аналоги та процес обміну повідомленнями;

– розробити вимоги до вебзастосунку обміну повідомленнями;

– проаналізувати ринок наявних технологій і рішень для вирішення завдань;

– розробити структуру баз даних та принцип роботи вебзастосунка обміну повідомленнями на основі REST API та Websocket;

– реалізувати, протестувати та налогодити вебзастосунок;

КРБ викладена на ___сторінки, вона містить ___ розділи, ___ілюстрацій, ___ таблиці, ___ джерел в переліку посилань.

Ключові слова: API, Websockets, обмін повідомленнями, мессенжер.

ABSTRACT

of the Bachelor's Thesis

«Web Application for messaging based on REST API on Websocket»

Student of group 408: Skrypnik Roman Volodymyrovych

Supervisor: senior lecturer Dvoretska Svetlana Volodymyrivna

The topic of qualification work is relevant, as the coronavirus pandemic and the invasion of the Russian Federation on the territory of Ukraine affected the ability of citizens to exchange information with each other.

The purpose of the qualification work is to develop a web application for messaging based on REST API on Websocket.

The object of research is the process of real-time messaging between users.

The subject of the research is information technologies, methods and tools for developing web messaging web applications in real time.

To achieve this goal you need to solve the following tasks:

– analyze modern analogues and the process of messaging;

– develop requirements for the distance learning system;

– analyze the market of available technologies and solutions to solve problems;

– develop the structure of databases and the principle of operation of web messaging applications based on REST API and Websocket;

– implement, test and tax web applications;

The work is ¬¬___ pages long and includes ___ sections, ___ illustrations, ___ tables, and ___ sources.

Keywords: API, Websockets, messaging, messenger.

ПЕРЕЛІК СКОРОЧЕНЬ

ЗВО – Заклад вищої освіти

ПЗ – Програмне забезпечення

СКБД – Система керування базою даних

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

REST – Representational State Transfer

ВСТУП

Веб-додаток (Web app) — це прикладна програма, яка зберігається на віддаленому сервері та доставляється через Інтернет через інтерфейс браузера. Веб-сервіси за визначенням є веб-додатками, і багато веб-сайтів, хоча й не всі, містять веб-програми. Кожний компонент веб-сайту, який виконує певну функцію для користувача, кваліфікується як веб-програма.

Веб-додатки можуть бути розроблені для широкого спектру використання і можуть використовуватися будь-ким; від організації до окремої особи з багатьох причин. Зазвичай використовувані веб-додатки можуть включати веб-пошту, онлайн-калькулятори або магазини електронної комерції. До деяких веб-програм можна отримати доступ лише через певний браузер; однак більшість із них доступні незалежно від браузера.

Веб-програми не потрібно завантажувати, оскільки доступ до них здійснюється через мережу. Користувачі можуть отримати доступ до веб-програми через веб-браузер, такий як Google Chrome, Mozilla Firefox або Safari.

Веб-додатки зазвичай мають короткі цикли розробки і можуть бути створені невеликими командами розробників. Більшість веб-програм написані на JavaScript, HTML5 або каскадних таблицях стилів (CSS). Програмування на стороні клієнта зазвичай використовує ці мови, які допомагають створювати інтерфейс програми. Програмування на стороні сервера виконується для створення сценаріїв, які використовуватиме веб-додаток. Такі мови, як Python, Java і Ruby, зазвичай використовуються в програмуванні на стороні сервера.

Веб-додатки мають багато різних застосувань, і з цим використанням приходить багато потенційних переваг. Деякі загальні переваги веб-програм:

– Надання доступу кільком користувачам до однієї версії програми.
– Веб-програми не потрібно встановлювати.

– Доступ до веб-програм можна отримати через різні платформи, такі як настільний комп’ютер, ноутбук або мобільний.
– Можна отримати доступ через кілька браузерів.

У секторі мобільних обчислень веб-програми іноді протиставляють рідним додаткам, які є програмами, розробленими спеціально для певної платформи або пристрою та встановленими на цьому пристрої. Однак ці два не виключають один одного. Нативні програми – це програми, які зазвичай завантажуються та створюються спеціально для типу пристрою, на який вони завантажуються. Нативні програми зазвичай можуть використовувати спеціальне обладнання для пристрою, наприклад GPS або камеру в мобільному рідному додатку.

Програми, що поєднують обидва підходи, іноді називають гібридними додатками. Гібридні програми працюють подібно до веб-додатків, але встановлюються на пристрої так само, як і рідні програми. Гібридні додатки також можуть використовувати ресурси, що стосуються пристрою, використовуючи внутрішні API. Завантажені рідні програми іноді можуть працювати в автономному режимі; однак гібридні програми не мають цієї функції. Гібридний додаток зазвичай має подібні елементи навігації, як і веб-програми, оскільки вони засновані на веб-програмах.

Також останнім часом набирає більшої популярності технологія WebSocket, яка не вимагає постійних запитів від клієнта до сервера, а створює двонаправлене з'єднання, при якому сервер може надсилати дані клієнту без запиту від останнього.

Протокол WebSocket створив нові можливості для спілкування через Інтернет і відкрив двері до справжньої мережі реального часу. У цій статті детально досліджуються WebSockets, глибоко занурюючись, щоб пояснити, як виникли WebSockets, що вони собою являють і як вони працюють, і як WebSockets працюють під капотом деяких реальних програм.

WebSocket забезпечує лише транспортний рівень, на якому цей процес обміну повідомленнями може бути реалізований, тому більшість поширених підпротоколів не є винятковими для комунікацій на основі WebSocket.

Написати програму чату з популярними стеками веб-програм, як-от LAMP (PHP), зазвичай було дуже важко. Це включає в себе опитування сервера щодо змін, відстеження часових позначок, і це набагато повільніше, ніж має бути.

Сокети традиційно були рішенням, навколо якого будується більшість систем чату в реальному часі, забезпечуючи двонаправлений канал зв’язку між клієнтом і сервером.

Це означає, що сервер може надсилати повідомлення клієнтам. Щоразу, коли ви пишете повідомлення в чаті, ідея полягає в тому, що сервер отримає його і передасть усім іншим підключеним клієнтам.

Socket.IO — це бібліотека, яка забезпечує двонаправлений зв’язок між клієнтом і сервером із низькими затримками та на основі подій. Він побудований на основі протоколу WebSocket і надає додаткові гарантії, такі як відхід до тривалого опитування HTTP або автоматичне повторне підключення.

Основна ідея Socket.IO полягає в тому, що ви можете надсилати та отримувати будь-які події, які хочете, з будь-якими даними, які ви хочете. Підтримуються будь-які об’єкти, які можна закодувати як JSON, а також підтримуються двійкові дані.

Метою роботи є підвищення зручності процесу обміну он-лайн повідомленнями шляхом розробки вебзастосунку основі REST API та websocket-ів.

Об’єктом роботи є процеси обміну повідомленнями між користувачами у режимі реального часу.

Предметом роботи є інформаційні технології, методи та інструментальні засоби розробки вебзастосунків обміну повідомленнями у режимі реального часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Головною метою вебзастосунку обміну повідомленнями на основі REST API та Websocket є можливість користувача спілкуватись з іншими користувачами, обмінюватись голосовими та медіа повідомленнями у режимі реального часу з застосуванням Websockets. Розкривається актуальність дослідження за обраним напрямом, ставиться проблема, мета і завдання дослідження, визначаються об'єкт та предмет дослідження, обґрунтування основних проектних рішень, вказується його теоретична, практична значущість. Спочатку у розділі роботи проводиться системний аналіз, потім формулюється постановка задачі та специфікація вимог до програмного забезпечення. У наступному розділі розробляються проектні рішення, що забезпечують виконання специфікації вимог до ПЗ, потім описується виконана робота з моделювання та конструювання ПЗ В останньому розділі представляється виконана робота з кодування, тестування розробленого ПЗ.

1.1 Функціональні та структурні особливості

Як і більшість вебзастосунків, вебзастосунок обміну повідомленнями має трьохярусну архітектуру. У триланкових програмах з архітектурою клієнт-сервер всі три програмні шари розділені і не залежать один від одного. У триланкових схемах обробки бізнес-логіка сама грає роль службової функції і також може виконуватися спеціально виділеному при цьому комп'ютері. Коли бізнес-логіка набуває характеру самостійної служби, її

зазвичай називають сервером прикладних програм або просто сервером додатків.

Засіб, який приймає дані та показує їх у зрозумілому людині вигляді, називається клієнтською стороною, або клієнтом. Цим засобом є браузер, мобільний додаток, що працює з даними сервера, комп'ютер, телефон, планшет, телевізор або радіоприймач.

Поняття про клієнтську сторону вийшло з парадигми клієнт-серверної архітектури. Вона з'явилася разом з першими комп'ютерними мережами: розробники вирішили, що частину загальних ресурсів можна зберігати за межами пристрою. Пристрій в такому випадку - це клієнт: він отримує дані від зовнішнього сервісу, малює їх у собі і відправляє зміни, що вносяться назад на сервіс.

Серверна частина – це термін для програмного забезпечення, апаратного забезпечення та баз даних, які використовуються для управління або забезпечення функціональності програми, системи, програмне забезпечення для електронної комерції або конструктор сайтів. У цій статті ми розглянемо шість причин, з яких наявність надійних серверних технологій є важливим для вашого бізнесу. Продовжуйте читати, щоб дізнатися більше про те, чому вашому бізнесу час впровадити потужну серверну частину.

Серверна частина - це система, в якій дані зберігаються та вилучаються. Вона надає програмне забезпечення, обладнання та бази даних, необхідні для управління або забезпечення функціональності програми або системи.

База даних (БД, або система управління базами даних, СУБД) - програмне забезпечення на сервері, що займається зберіганням даних та їх видачею у потрібний момент. У випадку форуму або блогу, дані, що зберігаються в БД, - це пости, коментарі, новини, і так далі. База даних розміщується на сервері. Серверна частина веб-програми (тобто PHP скрипт)

звертається до бази даних, витягуючи дані, які необхідні для формування сторінки, яку запитує користувач.

Функціональні вимоги до системи що розробляються:

– аутентифікація та реєстрація користувача у системі вебзастосунка;
– пошук та перегляд користувачів зареєстрованих у системі;
– створення чатів для обміну повідомленнями між користувачами;
– редагування та перегляд особистих даних користувача;
– створення кімнат для обміну повідомленнями між групою користувачів;
– завантаження мультимедійних даних (відео, зображення) у чатах та кімнатах.

1.2 Аналіз аналогічних застосунків

При аналізі предметної області виявлено такі аналоги як: Viber, Telegram, Whatsapp.

Viber це сервіс обміну повідомленнями та здійснення дзвінків, представлений у вигляді додатків. Він дозволяє безкоштовно обмінюватися повідомленнями, телефонувати колегам та друзям на будь-які пристрої.

Вебзастосунок Viber представляє собою безкоштовний застосунок-месенджер для обміну повідомленнями між користувачами, який здобув велику затребуваність в усьому світі за свій функціонал, швидкість роботи та простий інтерфейс.

Мовами реалізації є такі мови реалізації як: Java, C, Python, C++ та Objective C.

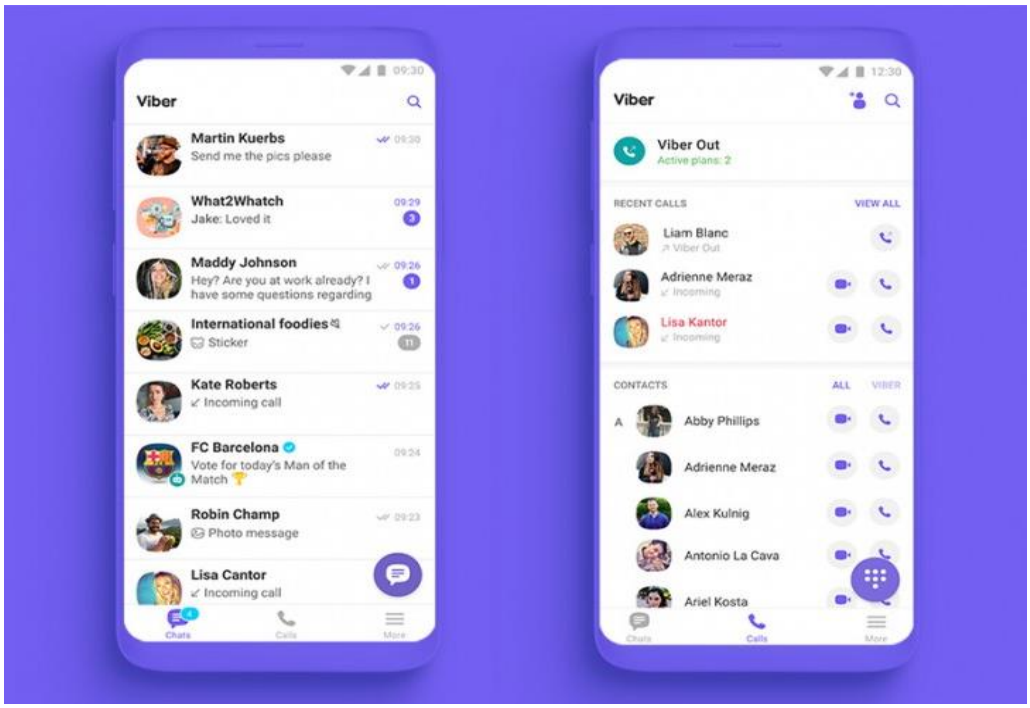

Рисунок 1.1 – Інтерфейс взаємодії Viber

Viber є вебзастосунком обміну повідомленнями між користувачами у режимі реального часу та має такий список функцій:

– обмін повідомленнями між користувачами;

– можливість спілкуватись з іншими користувачами використовуючи відеозв’язок та аудіозв’язок;

– створення груп для спілкування з іншими користувачами.

До переваг застосунка-месенджера Viber можна віднести такі пункти:

– Практичний та простий інтерфейс;

– Можливість безкоштовно розмовляти використовуючи аудіо або відео зв’язок;

– Створення чат-ботів для комерційних цілей;

– Тимчасове хмарне сховище мультимедії.

До основних недоліків вебзастосунка Viber відносяться:

– Слабка система захисту особистих даних;

– Поширення неліцензованих ПЗ та аудиозаписів;

– Велика кількість реклами;

– Погана оптимізація та підтримка старих пристроїв.

WhatsApp - це програма, що дозволяє користувачам мобільних пристроїв спілкуватися один з одним, використовуючи наскрізне шифрування. За його допомогою користувачі можуть безпечно переписуватися один з одним, здійснювати дзвінки, надсилати файли та вести групове листування.

Вебзастосунок WhatsApp є дуже схожим за своєю структурою та будовою з месенджером Viber.

У якості мови розробки застосунка WhatsApp було використано таку мову реалізації як Erlang.

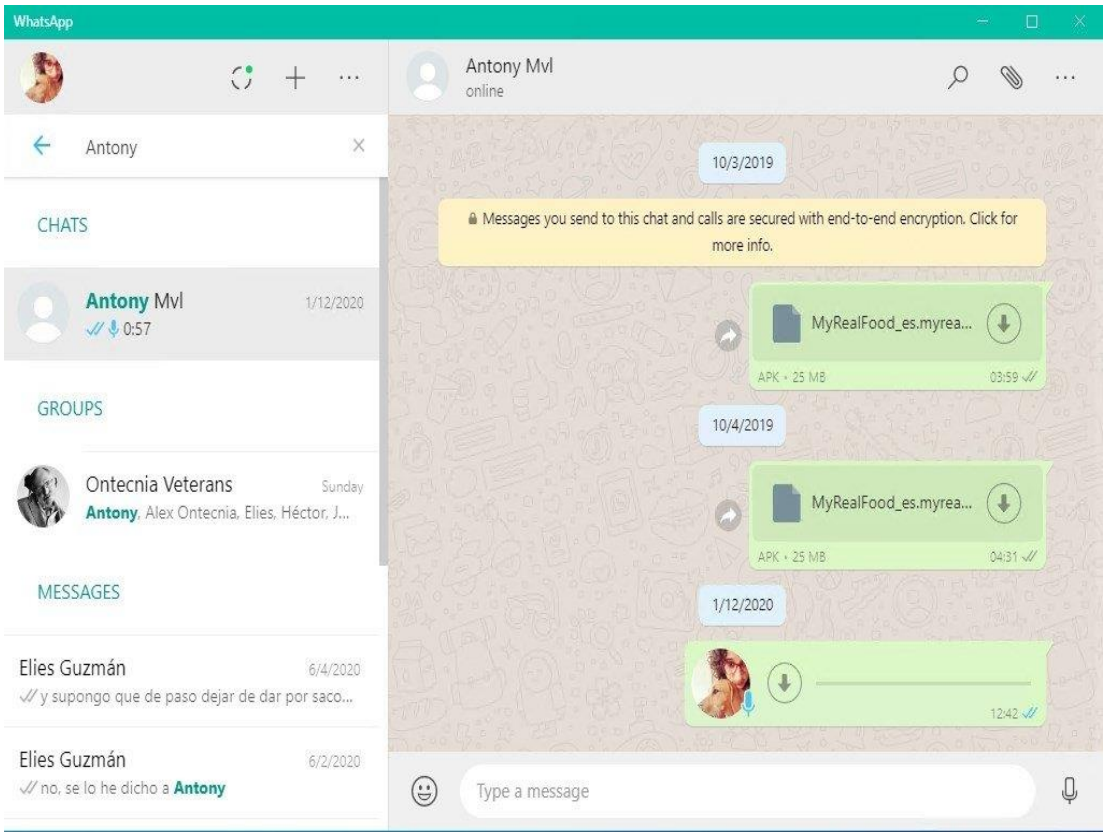

Рисунок 1.2 - Інтерфейс взаємодії з застосунком WhatsApp (десктоп адаптація)

Рисунок 1.3 - Інтерфейс взаємодії з застосунком WhatsApp (мобільна адаптація)

Основним функціоналом WhatsApp-а є:

– обмін повідомленнями між користувачами;

– можливість спілкуватись з іншими користувачами використовуючи відеозв’язок та аудіозв’язок;

– редагування особистих даних користувача.

Перевагами застосунка- меседжера WhatsApp є такий перелік

– простий та гнучкий інтерфейс;

– висока кросплатформова підтримка;

– висока швидкість роботи застосунка.

До основних недоліків вебзастосунка WhatsApp можна віднести такі пункти:

– обмеження на розмір файлів, що відправляються – 200 Мб;

– кількість учасників у групових чатах – не більше 250;

– безліч рекламних повідомлень та спаму.

Останнім за списком вебзастосунком обміну повідомленнями між користувачами є Telegram.

Telegram - це веб-додаток для обміну миттєвими повідомленнями з акцентом на швидкість та безпеку. Це швидкий, простий, безпечний та безкоштовний сервіс. Він легко синхронізується на всіх пристроях, працює на настільних ПК, планшетах та телефонах. За допомогою нього можна надсилати необмежену кількість повідомлень, фотографій, відео та файлів будь-якого типу.

Сервіс використовує розподілену інфраструктуру з центрами обробки даних у всьому світі для прискорення доступу до серверів та роботи програм. При цьому Telegram заснований на протоколі MTProto, який побудований на перевірених часом алгоритмах, забезпечуючи високу швидкість доставки та надійність. Telegram пропонує безпечний доступ до даних разом із збереженням історії листування у хмарі.

За допомогою Telegram можна сформувати великі групові чати та широкомовні списки із підключенням до 200 членів, для швидкого обміну відео, документами (.doc, .ppt, .zip, тощо) та фотографіями. Сервіс абсолютно безкоштовний і у ньому відсутня реклама. Для тих самих користувачів, хто зацікавлений у максимальній конфіденційності, Telegram пропонує Секретні чати.

Мовами реалізації Telegram є такі мови програмування як C++ та Java

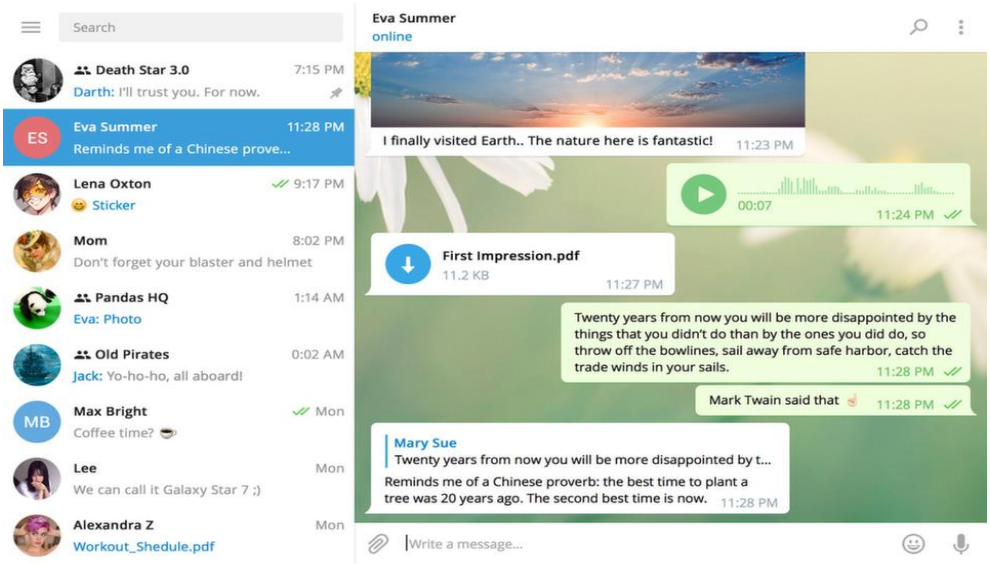

Рисунок 1.4 - Інтерфейс взаємодії з застосунком Telegram (десктоп адаптація)

Рисунок 1.5 - Інтерфейс взаємодії з застосунком Telegram (мобільна адаптація)

Основними функціями вебзастосунку Telegram є:

– обмін повідомленнями між користувачами;

– можливість спілкуватись з іншими користувачами використовуючи відеозв’язок та аудіозв’язок;

– редагування особистих даних користувача.

До головних переваг застосунка-месенджера Telegram можна віднести такі пункти:

– можливість створити секретні чати, які забезпечують end-to-end шифрування;

– великий обсяг захисту персональних даних користувача. Наприклад: двофакторна автентифікація;

– Висока кросплатформова підтримка усіх існуючих платформ гаджетів.

Серед основних недоліків вебзастосунка Telegram можна віднести такі пункти::

– слабка підтримка старих гаджетів;

– потребує багато системних ресурсів та трафік.

1.3 Аналіз існуючих методів і засобів вирішення поставлених завдань

Виконання поставленного завдання потребує використання сучасних методів та шляхів призначених для реалізації трирівневого застосунку обміну повідомленнями між користувачами на основі REST API та Websocket.

Головна перевага трирівневої архітектури полягає в тому, що кожен рівень працює на власній інфраструктурі. кожен рівень може розроблятися одночасно окремою командою розробників, і його можна оновлювати або масштабувати при необхідності, не торкаючись інших рівнів.

Основна трирівнева перевага – покращена масштабованість, тому що сервери додатків можуть бути розгорнуті на багатьох машинах. Крім того, база даних не встановлює триваліших з'єднань з кожним клієнтом — для цього потрібні з'єднання лише з меншою кількістю серверів програм. Це покращує цілісність даних.

Трирівнева архітектура є клієнт-серверною архітектурою, в якій функціональна логіка процесу, доступ до даних, комп'ютерне зберігання

даних і інтерфейс користувача розробляються і підтримуються як незалежні модулі на окремих платформах.

JavaScript — гнучка мова, яку можна використовувати як для фронтенду, так і для бекенда. Це хороша мова для початківців, оскільки в ньому мало налаштувань, і можна почати писати код у браузері.

Значно розширює можливості JS програмної платформи Node.js. З її допомогою код, написаний на JS, можна запускати без браузера на бекенді. А наявність величезної кількості готових рішень у пакетній екосистемі npm дозволяє розробнику не витрачати час створення більшості типових рішень.

MySQL – це реляційна система управління базами даних із відкритим вихідним кодом. В даний час ця СУБД одна з найбільш популярних у веб-додатках - переважна більшість CMS використовує саме MySQL (часто тільки її, без альтернатив), а майже всі веб-фреймворки підтримують MySQL вже на рівні базової конфігурації (без додаткових модулів).

Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, і таблиці InnoDB, підтримують транзакції лише на рівні окремих записів. Існують інші типи таблиць, розроблені спільнотою.

1.4 Специфікація вимог до ПЗ

Вебзастосунок обміну повідомленнями на основі REST API та Websocket являє собою застосунок-месенджер призначений для спілкування між користувачами у режимі реального часу. Система є доступною для кожного користувача та має безкоштовну реєстрацію з можливістю аутентифікації.

Застосунок передбачає можливість реалізації лише однієї основної ролі користувачів – користувача вебзастосунка.

Вебзастосунок обміну повідомленнями на основі REST API та Websocket є класичним представником трирівневого вебзастосунка який складається з трьох основних складових:

– клієнтська частина
– серверна частина
– сервер баз даних

Функціональні вимоги до системи що розробляються:

Аутентифікація користувача у системі вебзастосунка.

Вхідні дані: запит від клієнтської частини до серверної частини сайта з даними логіна та пароля користувача.

Вихідні дані: відповідь від серверної частина до клієнтської частини з токеном та перенаправленням на авторизовні шляхи клієнтської частини застосунку.

При введені невірних даних буде відпрацьована валідація та запит до серверної частини не буде відправлено.

Пошук та перегляд користувачів зареєстрованих у системі.

Вхідні дані: запит від клієнтської частини до серверної частини сайта з даними ім’я користувача.

Вихідні дані: відповідь від серверної частина до клієнтської частини з даними знайденого користувача.

Створення чатів для обміну повідомленнями між користувачами.

Вхідні дані: запит від клієнтської частини до серверної частини сайта з даними хеші користувача та даних відправленого повідомлення.

Вихідні дані: відповідь від серверної частина до клієнтської частини з даними створеного чату.

Редагування та перегляд особистих даних користувача.

Вхідні дані: запит від клієнтської частини до серверної частини сайта зі зміненними персональними даними користувача.

Вихідні дані: відповідь від серверної частина до клієнтської частини з новими даними користувача.

При введені невірних даних буде відпрацьована валідація та запит до серверної частини не буде відправлено.

Створення кімнат для обміну повідомленнями між групою користувачів.

Вхідні дані: запит від клієнтської частини до серверної частини сайта з даними назви кімнати.

Вихідні дані: відповідь від серверної частина до клієнтської частини з даними створенної кімнати.

При введені невірних даних буде відпрацьована валідація та запит до серверної частини не буде відправлено.

Завантаження мультимедійних даних (відео, зображення) у чатах та кімнатах.

Вхідні дані: запит від клієнтської частини до серверної частини сайта з даними про файли.

Вихідні дані: відповідь від серверної частина до клієнтської частини з даними створенного повідомлення з посиланнями на мультимедійні файли що були збережені на серверній частині застосунка.

Висновки до розділу 1

В ході описання вебзастосунка у поточному розділі був проведений детальний аналіз предметної області завдання – існуючих аналогів. Насамперед, був визначен функціонал, переваги та недоліки кожного з існуючих аналогів.

Проведений аналіз визначив засоби та методи, призначені для вирішення поставлених завдань за допомоги створення програмного коду для досягнення основної мети роботи. Враховуючи проведений аналіз аналогів та

їх недоліків, можна вважати, що тема роботи «Вебзастосунок обміну повідомленнями на основі REST API та Websocket» є актуальною.

2 МОДЕЛЮВАННЯ СТРУКТУРИ БД ЗАСТОСУНКУ

Вивчення системи за її моделлю дозволяє перевірити проектні рішення без участі в роботі реальної системи, розтягнути або стиснути час її розробки, зрозуміти складну взаємодію елементів усередині системи, оцінити ступінь дослідження зовнішніх та внутрішніх факторів, виявити «вузькі місця».

Одним з найбільш важливих елементів інформаційної системи є база даних, адже вона має повністю задовольнити зазначений у попередньому розділі функціонал. Для цього необхідно пройти етап проектування бази даних, який в тому числі полягає у створенні логічної та фізичної моделі даних.

База даних становить ключове поняття технології БД і стрижневою об'єкт управління в системах баз даних. Визначення бази даних як розділяється інформаційного ресурсу комп'ютеризованих технологій вимагає уточнення самих понять дані та інформація. Іноді база даних трактується як "подоби електронної картотеки", "сховища для деякого набору занесених в комп'ютер файлів даних", маючи на увазі під терміном файл "абстрактний набір даних, не обов'язково збігається з фізичним дисковим файлом". Очевидно, що при такому погляді дані і інформація розглядаються як синоніми. Як наслідок, істинним стає твердження про те, що в цьому випадку будь-які дані, витягнуті будь-яким способом з БД, є інформацією.

При створенні бази даних (ІС) зазвичай виділяється кілька рівнів моделювання, з яких відбувається перехід від предметної області до конкретної реалізації бази даних засобами конкретної СУБД. Можна виділити такі рівні: сама предметна область, концептуальна модель

предметної області, логічна модель даних, фізична модель даних, власне база даних та програми.

2.1 Проектування логічної моделі даних

Логічна (даталогічна) модель є модель бази даних, яка не прив'язана до конкретної СУБД. У ній виділяють основні об'єкти БД та визначають зв'язки між цими об'єктами. Іноді визначаються типи даних окремих об'єктів. Ця модель побудована шляхом Сутність-зв'язок (Entity Relationship).

Проектування бази даних - процес створення схеми бази даних та визначення необхідних обмежень цілісності.

Основні завдання:

– забезпечення зберігання у БД всієї необхідної інформації;
– забезпечує можливість отримання даних за всіма необхідними запитами;
– скорочення надмірності та дублювання даних;
– забезпечення цілісності бази даних.

Сутність - це клас однотипних об'єктів, інформація про які має бути врахована у моделі. Слово “клас” немає тут відношення до объектно – орієнтованого програмування. Проте, існує поняття сутності. Подсутність успадковує властивості сутності, що вона належить. У даній роботі можна виділити такі сутності:

– користувачі;
– чати;
– повідомлення;
– файли повідомлень;
– заблоковані користувачі;
– приглушені користувачі;
– кімнати;
– кімнатні повідомлення;

– файли кімнатних повідомлень;

Сутність «користувач» містить у собі записи про користувачів у вебзастосунку, реалізована для прав доступу у використанні власним обліковим записом користувача.

Сутність «чати» відповідає за чати створені користувачем задля спілкування з іншими користувачами.

Сутність «повідомлення» відповідає за повідомлення створенні у чатах

Сутність «файли повідомлень» містить у собі файли прив'язані до відправленних повідомлень.

Сутність «заблоковані користувачі» містить у собі користувачів заблокованих у чатах.

Сутність «приглушені користувачі» відповідає за користувачів приглушених у чатах.

Сутності баз даних які відносяться до кімнат мають аналогічну структуру як і у звичайних чатах, окрім сутності user_room, яка має зв’язок many-to-many та містить у собі записи користувачів, які перебувають у різних кімнатах.

Наступним кроком проектування баз даних є створення схеми баз даних. Основна інформація наведена у таблиці 2.1.

Таблиця 2.1 – Таблиця баз даних

Таблиця	**Стовпець**	**Опис стовпця**
user	id	Первинний ключ
	username	Унікальне ім’я
	name	Ім’я
	email	Електронна пошта
	password	Пароль
	bio	Коротка інформація
	hash	Унікальний код користувача

	avatar	Посилання на фото
	activated	Статус активованого акаунта
	online	Статус онлайн
	lastSeen	Дата останнього заходу на сайт

Продовження таблиці 2.1

chat	id	Первинний ключ
	user1Id	Первинний ключ першого користувача
	user2Id	Первинний ключ другого користувача
	updatedAt	Дата останнього оновлення
message	id	Первинний ключ
	text	Текст повідомлення
	isRead	Статус прочитаного повідомлення
	userId	Первинний ключ користувача
	chatId	Первинний ключ чата
	createdAt	Дата створення
file	id	Первинний ключ
	filename	Назва файла
	ext	Розширення файла
	messageId	Первинний ключ повідомлення
blocked	id	Первинний ключ
	chatId	Первинний ключ чата
	blockerId	Первинний ключ блокувальника
muted	id	Первинний ключ
	chatId	Первинний ключ чата
	muterId	Первинний ключ користувача який заглушив чат
	id	Первинний ключ

room	name	Назва
	description	Опис
	avatar	Посилання на фото
	hash	Унікальний код
	createdAt	Дата створення

Продовження таблиці 2.1

room_message	id	Первинний ключ
	text	Текст повідомлення
	isRead	Статус прочитаного повідомлення
	userId	Первинний ключ користувача
	chatId	Первинний ключ чата
	createdAt	Дата створення
room_file	id	Первинний ключ
	filename	Назва файла
	ext	Розширення файла
	messageId	Первинний ключ повідомлення
room_user	id	Первинний ключ
	roomId	Первинний ключ кімнати
	userId	Первинний ключ користувача

Згідно з [4] в результаті проєктування логічної моделі ER-методом результатом самого проєктування має бути ER-діаграма – спеціальна модель даних, що описує визначені сутності та зв’язки між ними.

Остаточна схема даних таблиць баз даних стовпців представлена на рисунку 2.1

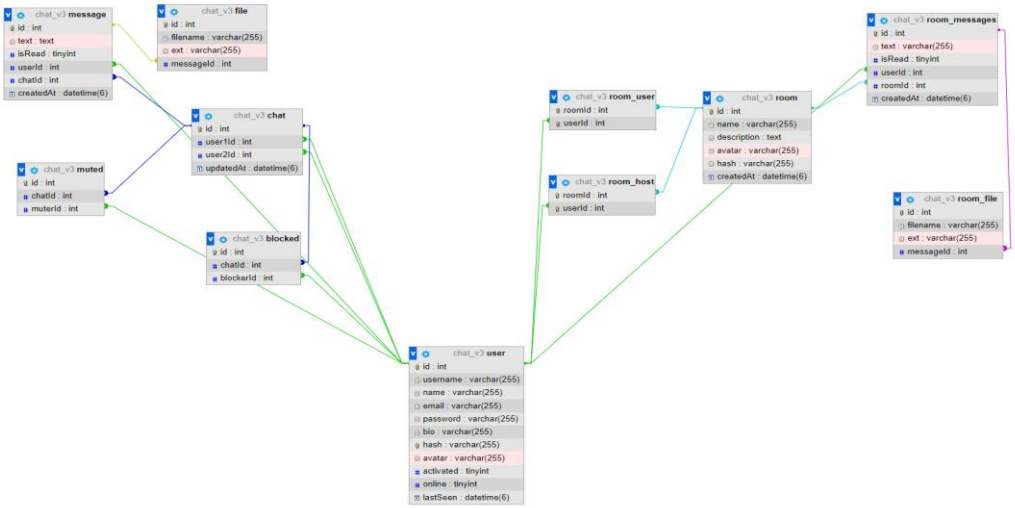

Рисунок 2.1 – Схема даних у базі

2.2 Проектування фізичної моделі даних

Фізична модель бази даних - це модель даних, яка визначає, яким чином надаються дані, і містить всі деталі, необхідні СУБД для створення бази даних.

Фізична модель містить усю інформацію, необхідну реалізації конкретної БД. Трансформаційна модель містить інформацію для реалізації окремого проекту, який може бути частиною загальної ІВ та описувати підмножину предметної області. ERwin підтримує ведення окремих проектів, дозволяючи проектувальнику виділяти підмножину моделі як предметних областей (Subject Area). Трансформаційна модель дозволяє проектувальникам та адміністраторам БД краще уявляти, які об'єкти БД зберігаються у словнику даних, та перевірити, наскільки фізична модель даних задовольняє вимогам до ІС.

Модель СУБД автоматично генерується з трансформаційної моделі та є точним відображенням системного каталогу СУБД. ERwin безпосередньо підтримує цю модель шляхом створення системного каталогу.

Висновки до розділу 2

У данному розіділі було проаналізовано методики створення баз даних, основні завдання баз даних, основні складові та сутності баз даних вебзастосунка обміну повідомленнями. Було спроектовано логічну модель даних, яка була представлена схемою даних у вигляді ER-діаграми.

3 АРХІТЕКТУРА, МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Прототипування мобільних програм, інтернет-сервісів та інших цифрових проектів є ключовим етапом розробки будь-яких програмних продуктів. Він дозволяє візуально уявити та при необхідності скоригувати всі вимоги технічного завдання. Прототипування переводить формальні та неформальні побажання замовника, представлені за допомогою тексту, записів та наборів ілюстрацій у продуманий та наочно оформлений прототип цифрового продукту.

Сучасна розробка цифрових послуг та мережевих сервісів, чи то створення мобільних чи веб-рішень у сфері інтернет-торгівлі, онлайн-банкінгу, бухгалтерії, сфери обслуговування, будівництва чи логістики, базується на об'єктно-орієнтованому підході. Такий підхід дозволяє надати цифрову послугу за допомогою набору програмних об'єктів із певними властивостями.

3.1 Вибір технологій та мов програмування

Основними мовами реалізації програмного забезпечення є такі мови програмування як: Javascript, HTML, CSS. Використання даного списку мов програмування необхідно для побудови будь-якого вебзастосунку у теперішньому часі, про що свідчить головний вебпортал W3schools.

Архітектура програмного забезпечення системи відображає організацію або структуру системи та дає пояснення того, як вона поводиться. Система є набором компонентів, що виконують певну функцію або набір функцій. Іншими словами, архітектура програмного забезпечення забезпечує міцну основу, на якій може бути побудоване програмне забезпечення.

Ряд архітектурних рішень та компромісів впливають на якість, продуктивність, ремонтопридатність та загальний успіх системи. Нездатність врахувати загальні проблеми та довгострокові наслідки може піддати вашу систему ризику.

Існує безліч архітектурних схем та принципів високого рівня, які зазвичай використовуються в сучасних системах. Їх часто називають архітектурними стилями. Архітектура програмної системи рідко обмежується одним архітектурним стилем. Натомість, комбінація стилів часто складає повну систему.

Як було зазначено, вебзастосунок обміну повідомленнями складається з трирівневої архітектури, яка поділяється на клієнт, сервер сторони та сервер баз даних.

Клієнтська частина вебзастосунка буде використовувати frontend фреймворк React TS. React це елемент програмування. Спеціалізована технологія, що використовується JavaScript для розробки різноманітного контенту.

Являє собою бібліотеку для створення інтерфейсів користувача. Має відкритий вихідний код. Вперше React з'явився у 2013 році. Виступає як кросплатформова бібліотека.

Зараз цей компонент має підтримку, а також розробляється Facebook та Instagram. Їм допомагає спільнота незалежних розробників та організацій.

React вже зарекомендував себе як потужний інструмент для створення інтерфейсів користувача. Інтерфейс користувача (UI) — одна з найважливіших частин веб-програми, адже це те, що користувач бачить, з чим він взаємодіє.

До основних переваг бібліотеки можна віднести:

1. Віртуальна об'єктна модель документа;
2. Повторне застосування компонентів;
3. Східний потік даних;
4. Велике співтовариство;
5. Браузерні інструменти React-розробника.

Для створення серверної частини вебзастосунка буде використано фреймворк мови програмування Node JS – Nest TS.

Nest (NestJS) — це платформа для створення ефективних, масштабованих додатків Node.js на стороні сервера. Він використовує прогресивний JavaScript, побудований на основі TypeScript і повністю підтримує його (проте все ще дозволяє розробникам кодувати на чистому JavaScript) і поєднує елементи OOP (об’єктно-орієнтоване програмування), FP (функціональне програмування) і FRP (функціональне реактивне програмування).

Під капотом Nest використовує надійні фреймворки HTTP-сервера, такі як Express (за замовчуванням), і за бажанням його можна також налаштувати на використання Fastify!

Nest забезпечує рівень абстракції вище цих поширених фреймворків Node.js (Express/Fastify), але також надає їхні API безпосередньо розробнику.

Це дає розробникам свободу використовувати безліч модулів сторонніх розробників, які доступні для базової платформи.

До переваг фреймворку можна віднести такі пункти:

1. **Розширюваність**: завдяки модульній архітектурі Nest дозволяє використовувати інші існуючі бібліотеки у вашому проекті;
2. **Архітектура**: Nest має архітектуру проекту, яка забезпечує легкість тестування, масштабованість та ремонтопридатність;
3. **Універсальність**: Nest надає екосистему для створення всіх видів серверних програм;
4. **Прогресивність**: Nest використовує новітні функції JavaScript та реалізує зрілі рішення та шаблони проектування у розробці програмного забезпечення.

В основі серверу баз даних було обрано реляційну систему управління MySQL. MySQL – це популярний сервер баз даних, що використовується у різних додатках. SQL означає мову структурованих запитів - (S)tructured (Q)uery (L)anguage, яку MySQL використовує для комунікації з іншими програмами. Крім того, MySQL має власні розширені функції SQL для того, щоб забезпечити користувачам додатковий функціонал.

До переваг СКБД MySQL можна віднести

1. Це безкоштовне програмне забезпечення, тому ви можете використовувати і налаштовувати його для своєї мети без будь-яких витрат;
2. IF може пропонувати різні функції навіть якщо вони надаються безкоштовно;
3. Для MySQL необхідно реалізувати безліч інтерфейсів користувача;
4. Це може працювати у поєднані із іншими базами даних, як-от DB2, Oracle тощо.

Для розробки графічного інтерфейсу вебзастосунку було використано графічну CSS-бібліотеку Boostrap. Bootstrap – це відкритий та безкоштовний HTML, CSS та JS фреймворк, який використовується веб-розробниками для швидкої верстки адаптивних дизайнів сайтів та веб-додатків.

3.2 Розробка сценаріїв використання та UML-діаграм

Робота з UML-діаграмами - важлива частина проекту, тому що на цьому етапі продумується його структура. Проектування допомагає не заплутатися в коді, знизити кількість помилок і спростити роботу.

UML покликаний дати стандартну нотацію, використовувану усіма об'єктно-орієнтованими методами. Крім того, UML дозволяє вибирати та впроваджувати напрацювання нотацій-попередників. Оскільки UML створено широкого спектра програм, з його допомогою можна конструювати різні системи.

Основні цілі дизайну UML:

1. Проектування. Завдяки UML розробники отримали можливість створювати моделі різних процесів, аналізувати, проектувати та впроваджувати програмні системи, малювати схеми додатків, за якими згодом пишеться код;
2. Забезпечення зростання ринку об'єктно-орієнтованих інструментів та розвитку галузі;
3. Створення UML так, щоб можна було працювати з будь-якою мовою програмування та будь-якого користувача;
4. Комунікація всередині команди та при спілкуванні із замовником.

Переважно, більшість сценаріїв відображається у графічному вигляді. Приклади сценаріїв взаємодії користувача з системою вебзастосунка обміну повідомленнями між користувачами приведені нижче: сценарій аутентифікації користувача на сайті (Рисунок 3.1), сценарій обміну

повідомленнями з іншим користувачем (Рисунок 3.2), сценарій створення нової кімнати (Рисунок 3.3).

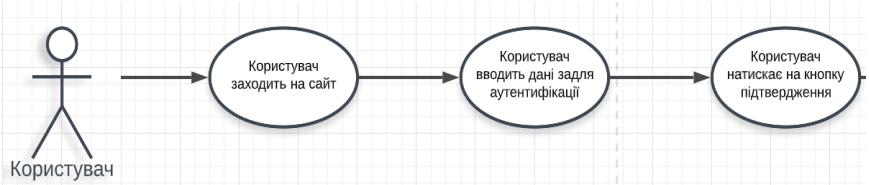

Рисунок 3.1 – Сценарій аутентифікації користувача на сайті

Рисунок 3.2 – Сценарій обміну повідомленнями між користувачами

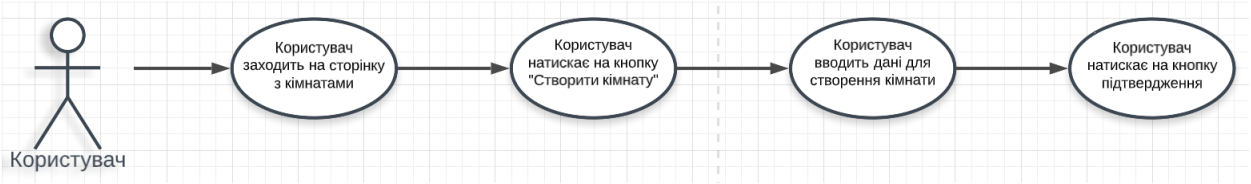

Рисунок 3.3 – Сценарій створення нової кімнати

Також, сценарії можуть бути представлені у текстовому вигляді. Вони можуть поділятись на: короткі, поверхневі та повні. У даній роботі було розроблено три сценарії з різними типами текстових сценаріїв.

Перший сценарій (коротка форма): аутентифікація користувача на сайті. Користувач заходить на сайт, переходить на сторінку аутентифікації. Користувач вводить дані задля входу у систему. Після введення даних, користувач натискає на кнопку “підтвердити” та отримує відповідь від сервера.

Другий сценарій (поверхнева форма): обмін повідомленнями між користувачами.

Головний сценарій (успішний):

Користувач заходить на сторінку з чатами та обирає чат з користувачем з яким хоче поспілкуватись. Користувач вводить повідомлення у текстове

поле та натискає на кнопку “підтвердити”, яка знаходиться у інтерфейсі вебзастосунку або натискає на кнопку Enter на клавіатурі.

Альтернативні сценарії:

1. Користувач намагається відправити повідомлення не підготувавши повідомлення для відправки. Валідація повідомить про помилку.

Третій сценарій (повна форма): створення нової кімнати.

У даному сценарії описується процес створення нової кімнати для обміну повідомленнями між користувачами.

Scope: модальне вікно з формою для створення кімнати.

Level: сторінка з кімнатами.

Primary Actor: Користувач.

Stackholders and interests:

1. Користувач: створює нову кімнату для спілкування з іншими користувачами.

Preconditions:

1. Користувач зайшов на сторінку з кімнатами, нажав на кнопку “Створити кімнату”.

Main Success Scenario:

1. Користувач переходить на сторінку з кімнатами;
2. Користувач натискає на кнопку “Створити кімнату”;
3. Користувач отримує модальне вікно з полями для створення кімнати;
4. Користувач заповнює дані у поля;
5. Користувач натискає на кнопку підтвердження;
6. Дані відправляються на серверну частину;
7. Результат відповіді від серверної частини відображається на клієнтській стороні вебзастосунку.

Extensions:

1. В будь який час користувач не виконує дії, передбачені сценарієм.
    a. Користувач переглядає сторінку з кімнатами;
    b. Користувач обирає кімнату для обміну повідомленнями;
    c. Користувач закриває сторінку з кімнатами.
2. Виникає технічна помилка.
    a. Виникла помилка на серверній частині вебзастосунку.
        i. Клієнт сторона застосунку повідомляє про помилку у інтерфейсі.
3. Виникає логічна помилка.
    a. Користувач не заповнює або заповнює не усі поля для створення нової кімнати.
        i. Спрацьовує валідація вебзастосунка та повідомляє про помилку у інтерфейсі користувача.

Special Requirements:

1. Персональний комп'ютер або ноутбук.
2. Стабільний зв'язок з мережею Інтернета задля швидкого відправлення даних на серверну частину вебзастосунку та отримання збережених даних.

Technology and Data Variations List:

1. Користувач повинен дотримуватись послідовності дій задля створення нової кімнати.

Frequency of Occurrence:

Вебзастосунок повинен працювати безперебійно. На кожну дію пов'язану з взаємодією з серверною частиною повинен видавати відповідь.

Miscellaneous (Open Issues):

1. Необхідно протестувати вебзастосунок на можливі проблеми.
2. Перевірити інтерфейс на простоту та практичність у використанні

Діаграма прецедентів або діаграма варіантів використання - діаграма, що відображає відносини між акторами та прецедентами і є складовою моделі прецедентів, що дозволяє описати систему на концептуальному рівні.

Основне призначення діаграми - опис функціональності та поведінки, що дозволяє замовнику, кінцевому користувачеві та розробнику спільно обговорювати проектовану або існуючу систему.

При моделюванні системи за допомогою діаграми прецедентів системний аналітик прагне:

1. чітко відокремити систему від її оточення;
2. визначити дійових осіб (акторів), їхню взаємодію із системою та очікувану функціональність системи;
3. визначити у глосарії предметної області поняття, що належать до детального опису функціональності системи (тобто прецедентів).

Діаграма варіантів використання яка була розроблена для данного вебзастосунку знаходиться нижче (Рисунок 3.4)

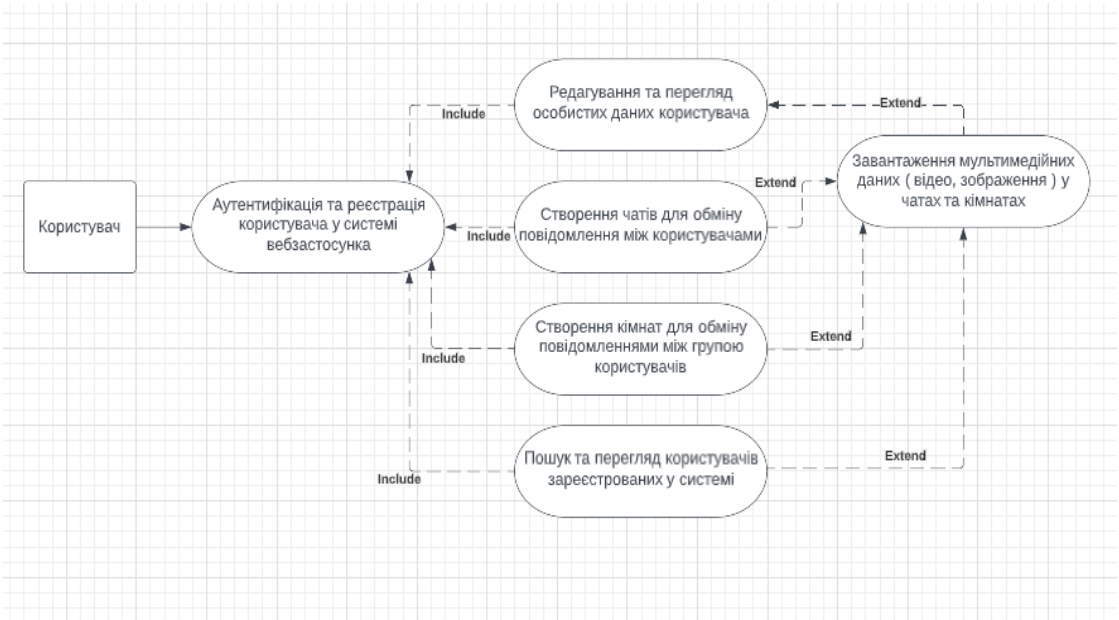

Рисунок 3.4 – Діаграма варіантів використання системи

Діаграма активності UML дозволяє детальніше візуалізувати конкретний випадок використання. Це поведінкова діаграма, що ілюструє потік діяльності через систему.

Діаграми активності UML можуть бути використані для відображення потоку подій у бізнес-процесі. Вони можуть бути використані для вивчення бізнес-процесів з метою визначення їх потоку та вимог.

Діаграми діяльності використовуються для моделювання процесів та робочих процесів. Суть корисної діаграми діяльності у тому, щоб передати певний аспект динамічного поведінки системи. Діаграми дій відбивають динамічні елементи системи.

В ході роботи було створено саме таку діаграму, яка описує процес створення нової кімнати користувачем (Рисунок 3.5).

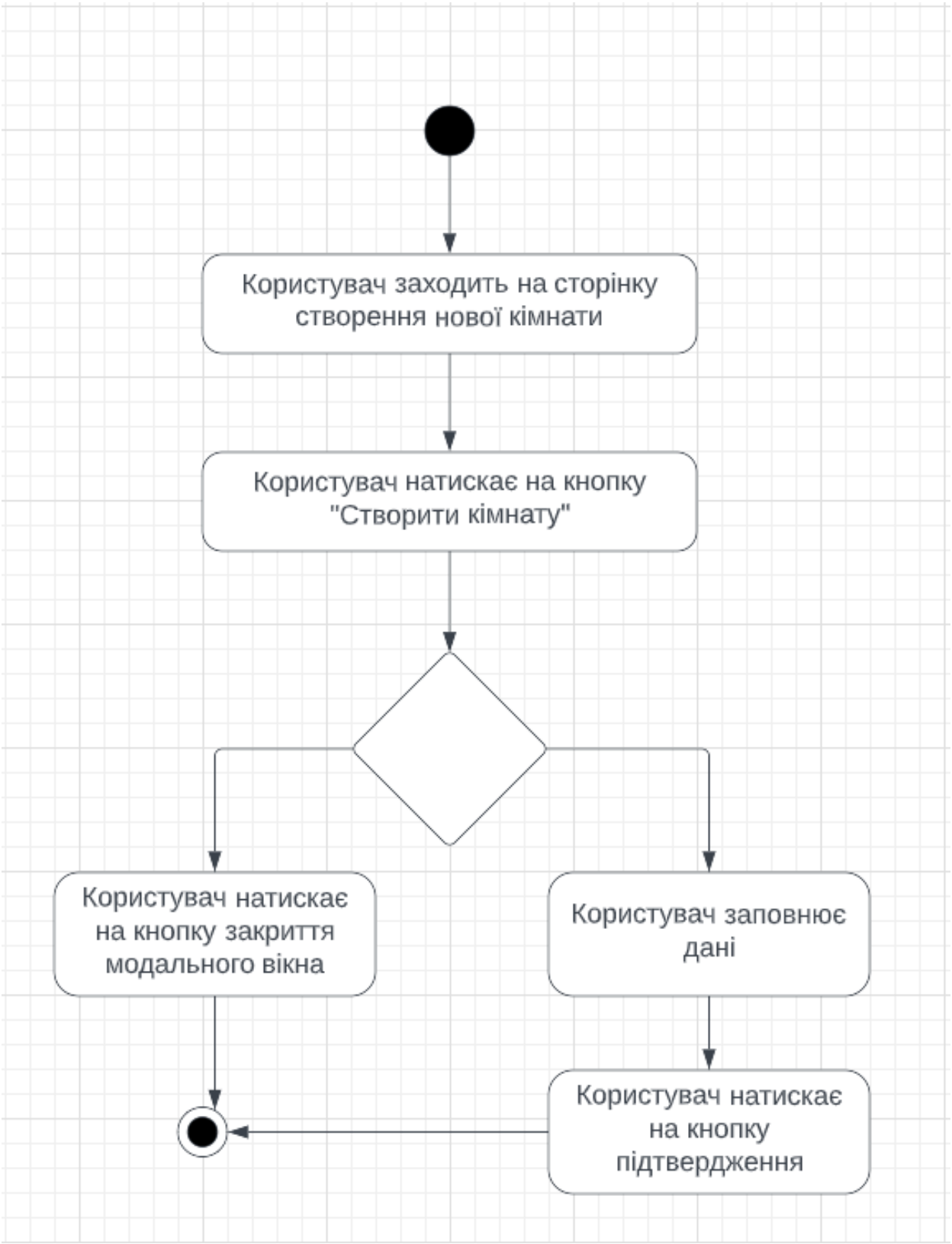

Рисунок 3.5 – Діаграма діяльності

Діаграма класів визначає типи класів системи та різного роду статичні зв'язки, які існують між ними. На діаграмах класів зображуються також атрибути класів, операції класів та обмеження, що накладаються на зв'язки між класами. Вигляд та інтерпретація діаграми класів істотно залежить від точки зору (рівня абстракції): класи можуть представляти сутності предметної галузі (у процесі аналізу) або елементи програмної системи (у процесах проектування та реалізації). В ході роботи було створено діаграму класів вебзастосунку (Рисунок 3.6).

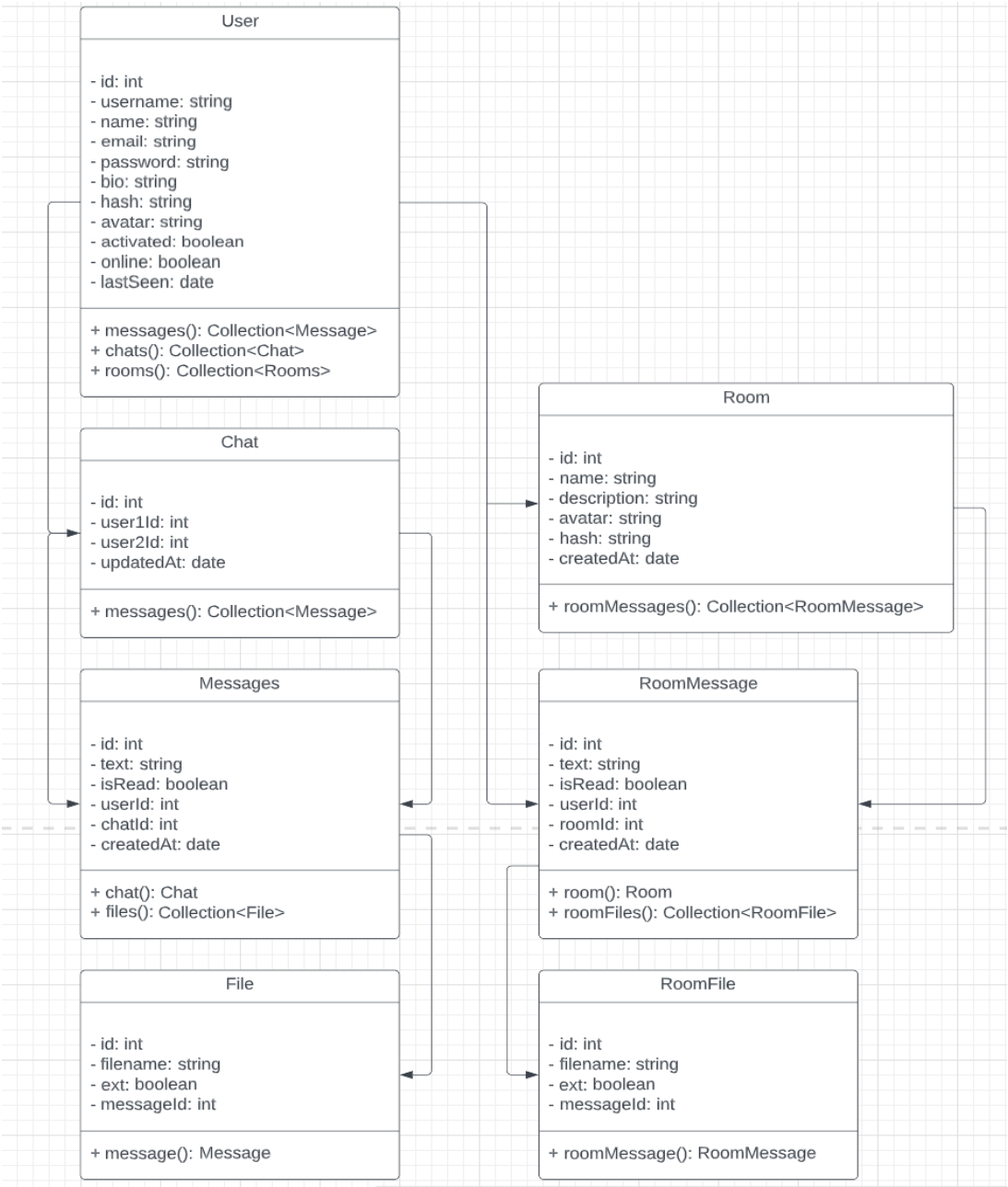

Рисунок 3.6 – Діаграма класів

Діаграма розгортання – це тип UML-діаграми, яка показує архітектуру виконання системи, включаючи такі вузли, як апаратні або програмні середовища виконання, а також проміжне програмне забезпечення, яке їх з'єднує.

Діаграми розгортання зазвичай використовуються для візуалізації фізичного апаратного та програмного забезпечення системи. Використовуючи його, можна зрозуміти, як система буде фізично розгорнута на апаратному забезпеченні.

Діаграми розгортання допомагають моделювати апаратну топологію системи порівняно з іншими типами UML-діаграм, які здебільшого описують логічні компоненти системи. Було реалізовано дану діаграму для розроблюваного вебзастосунку (Рисунок 3.7)

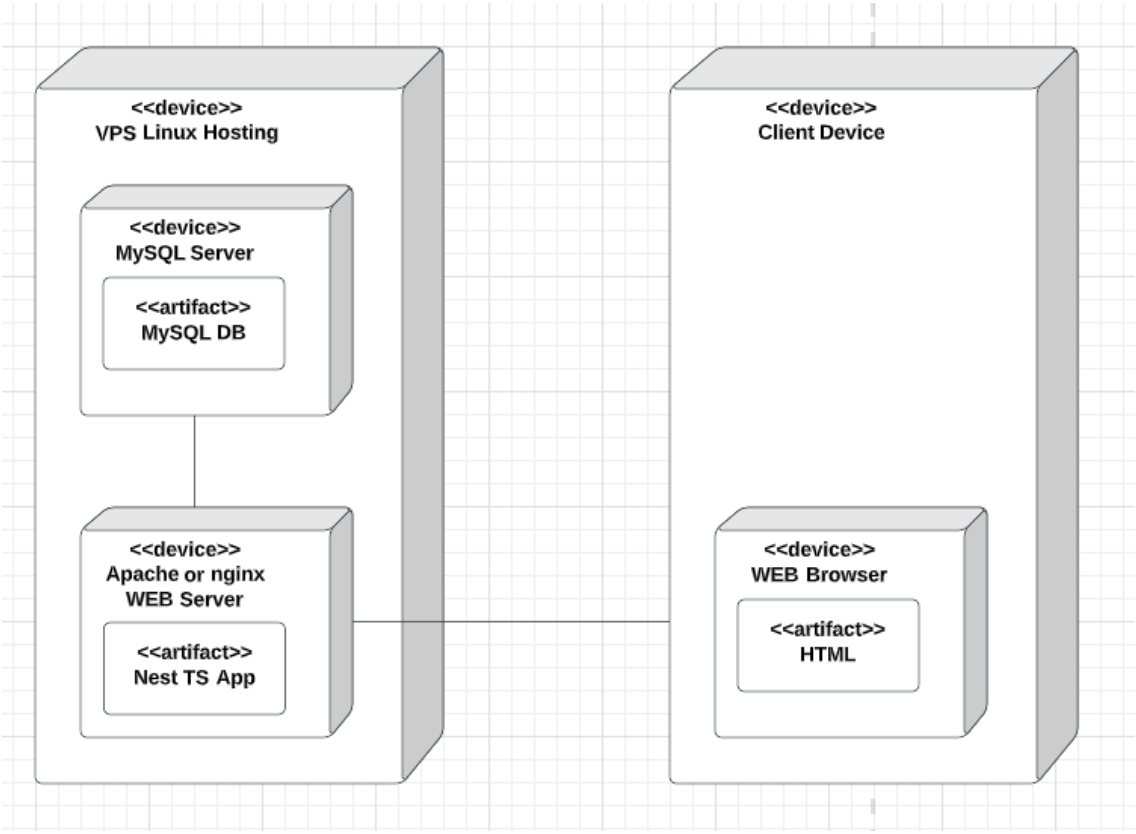

Рисунок 3.7 – Діаграма розгортання

Діаграма послідовності - UML-діаграма, на якій для деякого набору об'єктів на єдиній часовій осі показаний життєвий цикл об'єкта (створення-діяльність-знищення певної сутності) та взаємодія акторів (діючих осіб) інформаційної системи у рамках прецеденту.

Основними елементами діаграми послідовності є позначення об'єктів (прямокутники з назвами об'єктів), вертикальні «лінії життя» (англ. lifeline), що відображають протягом часу, прямокутники, що відображають діяльність об'єкта або виконання ним певної функції (прямокутники на пунктирній «лінії життя»), та стрілки, які показують обмін сигналами чи повідомленнями між об'єктами.

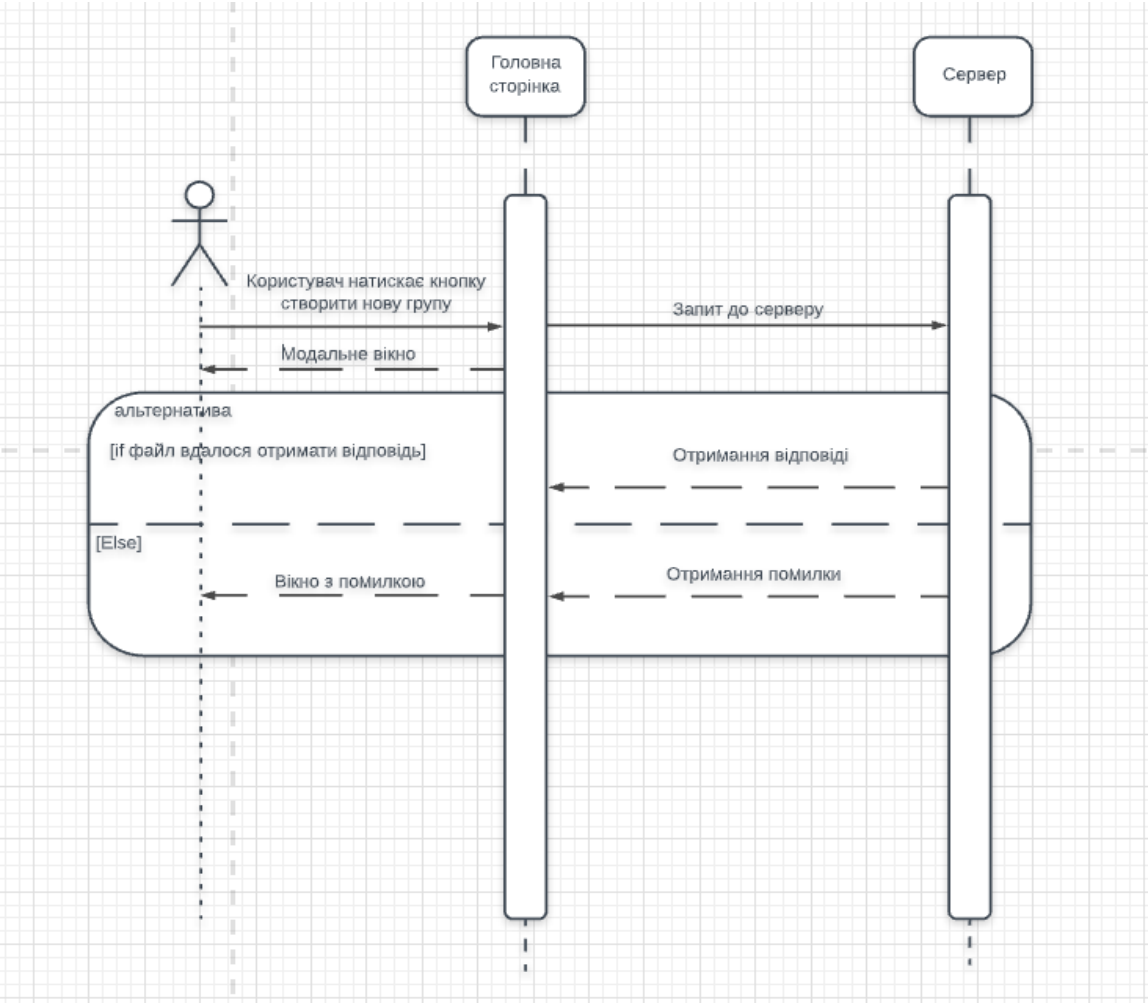

Рисунок 3.8 – Діаграма послідовності

3.3 Опис інтерфейсів ПЗ

Інтерфейс користувача є сукупність програмних і апаратних засобів, що забезпечують взаємодію користувача з комп'ютером. Основу такої взаємодії становлять діалоги. Під діалогом у разі розуміють регламентований обмін інформацією між людиною і комп'ютером, здійснюваний у реальному масштабі часу і спрямований на спільне вирішення конкретної задачі: обмін інформацією та координація дій . Кожен діалог складається з окремих процесів введення-виводу, які фізично забезпечують зв'язок користувача та комп'ютера.

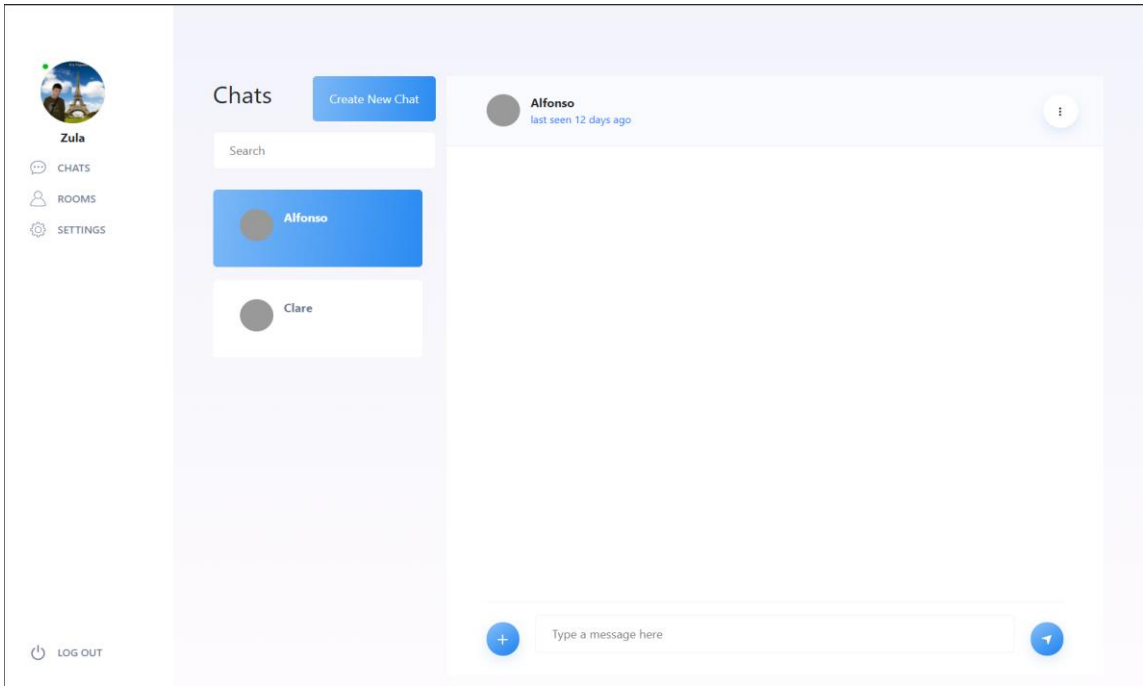

Рисунок 3.9 – Головна сторінка вебзастосунку з чатами

У якості графічного макету для створення клієнтської сторони вебзастосунку було узято у файл з онлайн-сервісу Figma. Figma – це нова програма для веб-дизайнерів. З її допомогою можна створювати як прототипи, а й самі інтерфейси (сайти, програми, панель управління). Важливою перевагою цієї програми є її простота. Саме за рахунок цього Figma завойовує серця спеціалістів у сфері веб-дизайну.

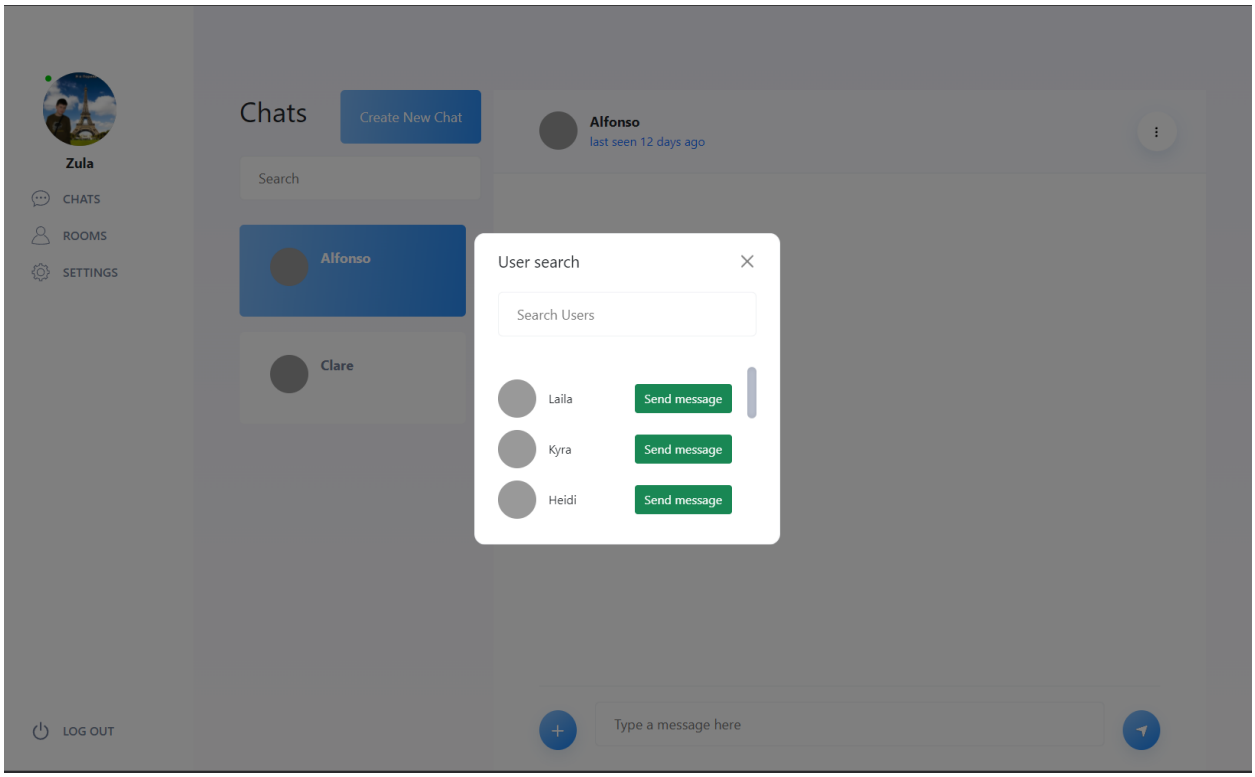

Рисунок 3.10 – Модальне вікно пошуку користувачів

Дизайн сторінки з кімнатами має аналогічне оформлення зі сторінкою чатів. Це було зроблено з ціллю спростити інтерфейс та збільшити швидкість розробки вебзастосунку (Рисунок 3.11).

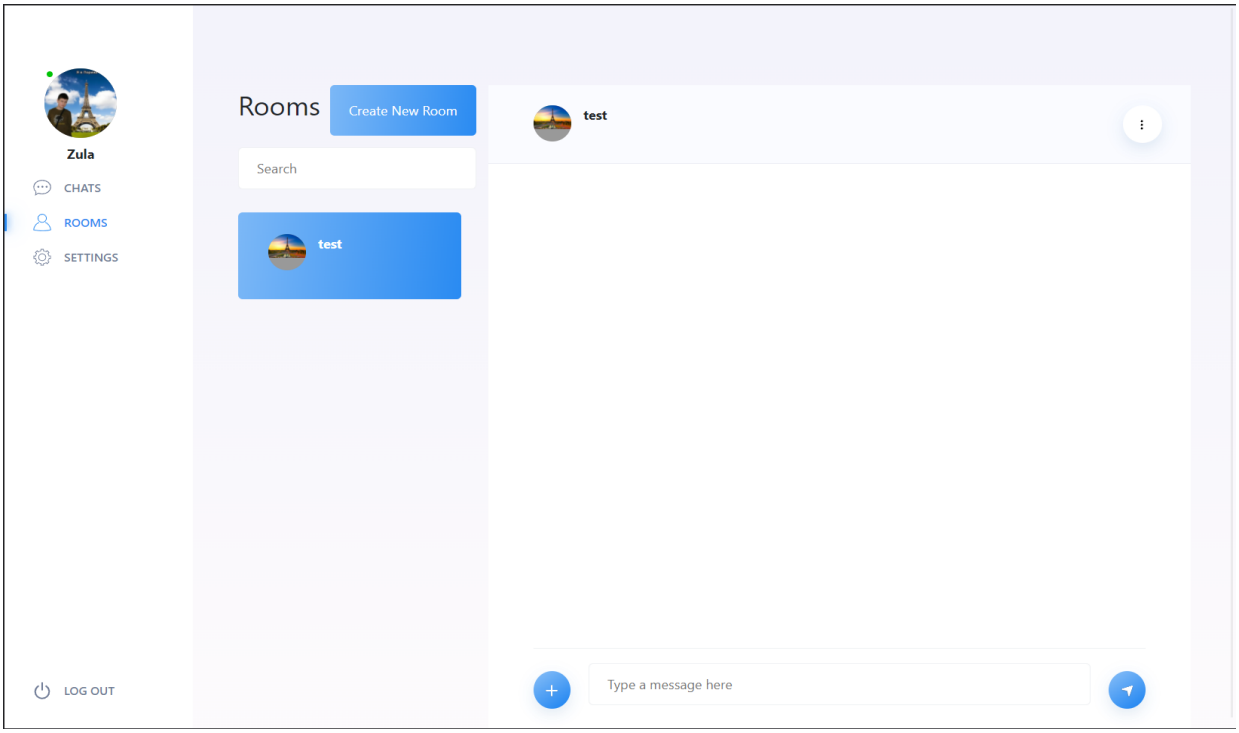

Рисунок 3.11 – Сторінка з кімнатами

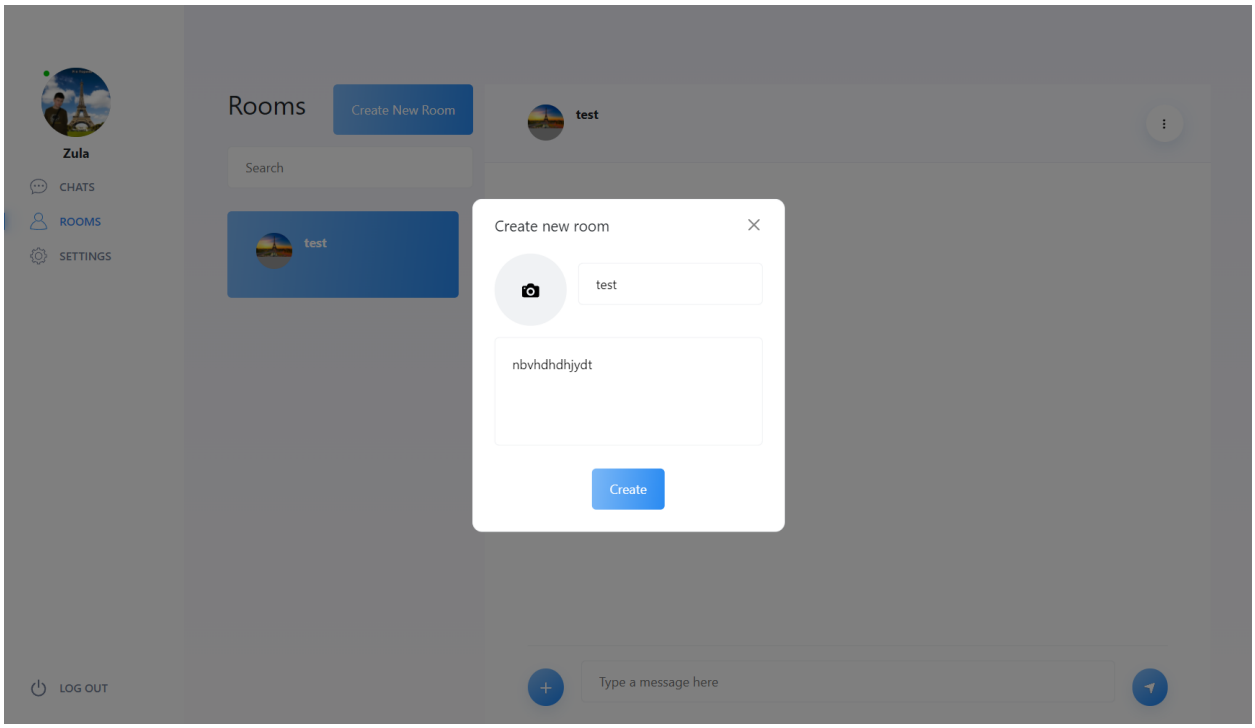

Рисунок 3.12 – Модальне вікно створення нової кімнати

Стосовно дизайну сторінки налаштування персональних даних користувача та зміни пароля – було прийнято рішення зробити сторінки без використання макетів для дизайну.

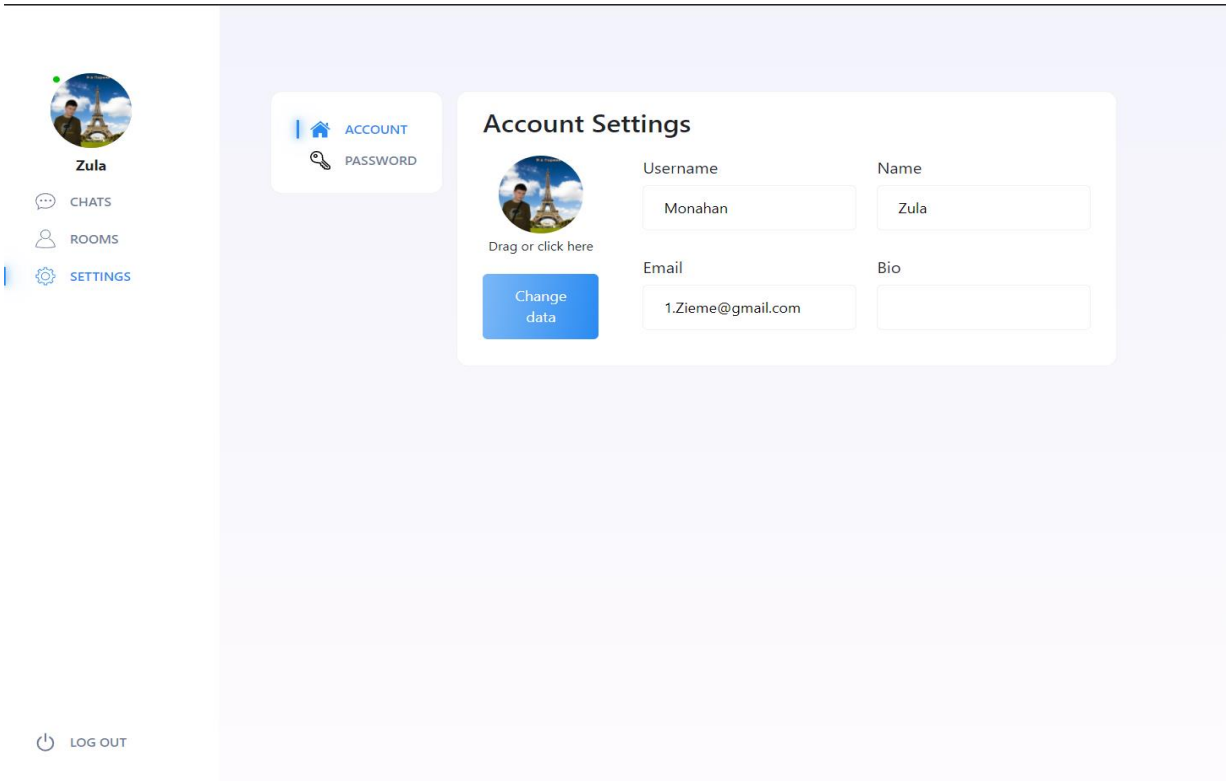

Рисунок 3.13 – Сторінка з налаштуванням персональних даних користувача

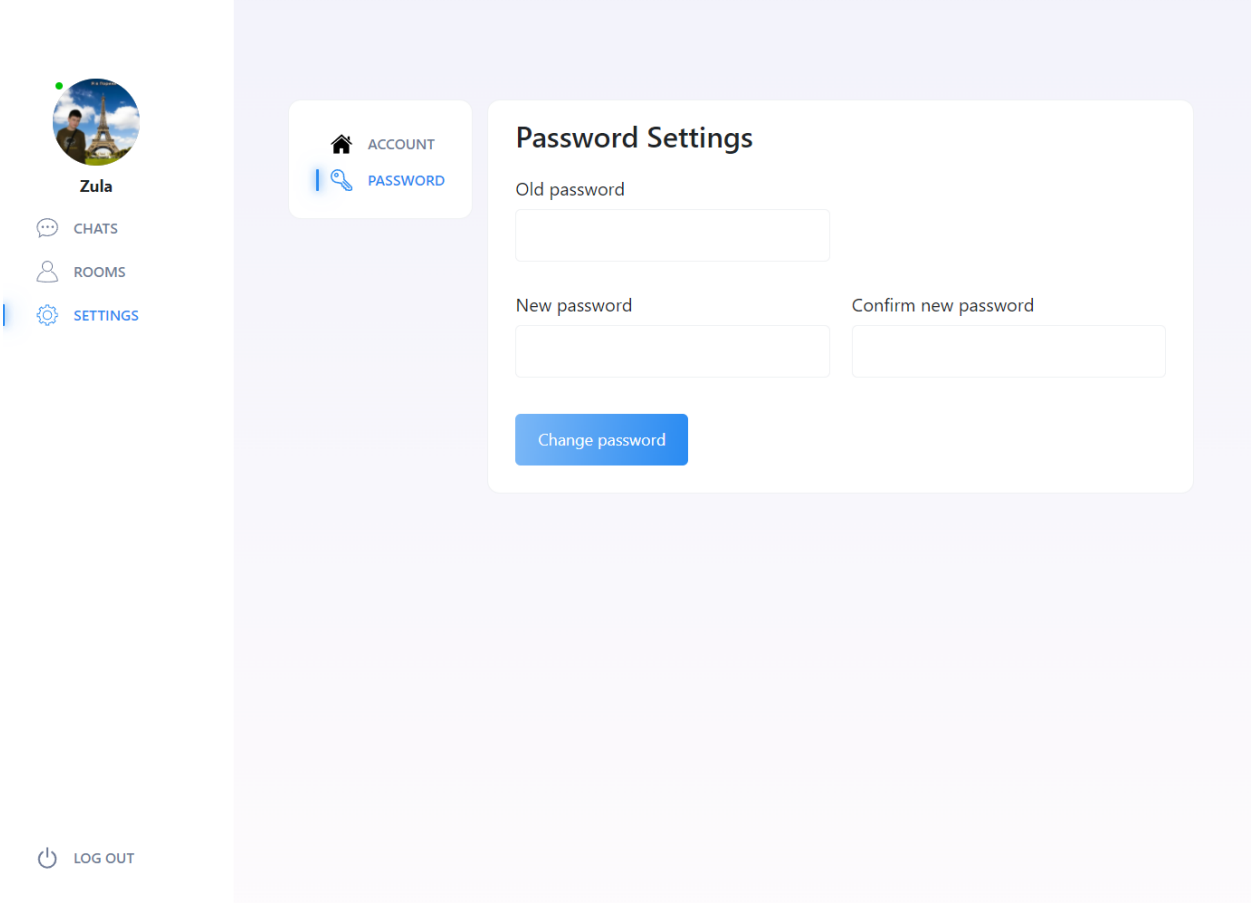

Рисунок 3.14 – Сторінка з зміною пароля

Сторінки аутентифікації та реєстрації користувача у системі вебзастосунку мають схожий дизайн та відрізняються текстовими полями.

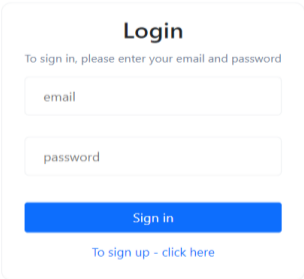

Рисунок 3.15 – Сторінка з логін формою

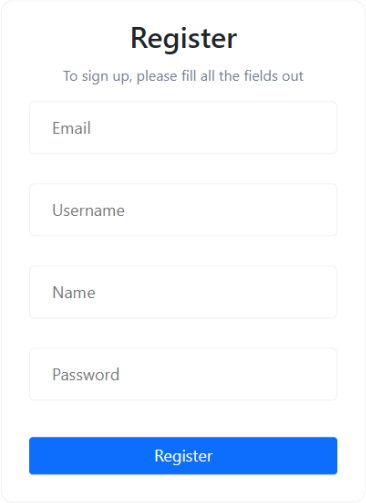

Рисунок 3.16 – Сторінка з формою регістрації

Висновки до розділу 3

У даному розділі було проаналізовано основні технології реалізації вебзастосунку обміну повідомленнями на основі REST API та Websocket, були визначені їх основні переваги над іншими альтернативами.

Були спроектовані сценарії та діаграми до програмного забезпечення. У ході роботи були реалізовані три основних типа сценаріїв – коротка, поверхнева та повна форми. Аналогічно з сценаріями, були спроектовані такі діаграми як: діаграма варіантів використання системи, діаграма діяльності, діаграма класів, діаграма розгортання, діаграма послідовності

4 КОДУВАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вебзастосунок — це програма, одна частина якої завантажується в браузер і взаємодіє з користувачем (візуально-інтерфейсна частина), а інша знаходиться на веб-сервері та виконує запити, що надходять від першої, а потім повертає відповідь. Частина, яка завантажується в браузер і з якою взаємодіє користувач, називається частиною клієнта (фронтенд). На веб-сервері знаходиться серверна частина веб-програми (бекенд).

Як було зазначено, у якості фреймворка задля клієнт частини буде використовуватись React TS, backend частина буде написана на фреймворку Node JS – Nest TS та сервер баз даних буде використовувати MySQL.

4.1 Робота з базою даних

TypeORM — це ORM, який може працювати на платформах NodeJS, Browser, Cordova, PhoneGap, Ionic, React Native, NativeScript, Expo та Electron і може використовуватися з TypeScript та JavaScript (ES5, ES6, ES7, ES8). Його мета — завжди підтримувати найновіші функції JavaScript та надавати додаткові функції, які допоможуть вам розробляти будь-які програми, які використовують бази даних — від невеликих програм з кількома таблицями до великомасштабних корпоративних програм із кількома базами даних.

Найшвидший спосіб розпочати роботу з TypeORM — використовувати його команди CLI для створення початкового проекту. Швидкий старт працює, лише якщо ви використовуєте TypeORM у програмі NodeJS.

Робота з базою даних починається зі створення таблиць. Створення таблиць відбувається через створення моделей.

```
export class Photo {
    id: number
    name: string
    description: string
    filename: string
    views: number
    isPublished: boolean
}
```

Рисунок 4.1 – Приклад створеної моделі Photo

Щоб зберігати речі в базі даних, по-перше, потрібна таблиця бази даних, а таблиці бази даних створюються з моделей. Не всі моделі, а лише ті, які визначені як сутності.

Після того, як ви почнете виробництво, потрібно буде синхронізувати зміни моделі з базою даних. Як правило, використовувати synchronize небезпечно: true для синхронізації схеми на виробництві, коли ви отримаєте дані в базу даних. Тут на допомогу приходить міграція.

Міграція — це лише один файл із запитами sql для оновлення схеми бази даних та застосування нових змін до існуючої бази даних.

```
ALTER TABLE "post" ALTER COLUMN "title" RENAME TO "name";
```

Рисунок 4.2 – Команда для створення таблиці у базах даних

Після виконання цього SQL-запиту схема бази даних готова до роботи з новою базою коду. TypeORM забезпечує місце, де ви можете писати такі запити sql і запускати їх, коли це необхідно. Це місце називається «міграції».

4.2 Програмна реалізація

Nest (NestJS) це фреймворк для побудови ефективних, масштабованих сервер-сайд додатків. У ньому використовується прогресивний JavaScript, він розроблений на TypeScript і повністю його підтримує (проте дозволяє розробникам писати код на чистому JavaScript) і комбінує в собі елементи ООП (Об'єктно орієнтоване програмування), ФП (Функціональне програмування) та ФРП (Функціонально-реактивне програмування)).

Під капотом використовує надійні HTTP фреймворки такі як Express (за замовчуванням) і за бажанням може бути налаштований на використання Fastify.

Nest надає рівень абстракції над цими Node.js фреймворками (Express/Fastify), але також надає їх API для розробника. Це дозволяє розробникам вільно використовувати велику кількість сторонніх модулів, які доступні для базової платформи.

Щоб розпочати роботу, можна скористатися Nest CLI. Nest прагне бути платформо-незалежним фреймворком. Незалежність від платформи уможливлює створення перевикористовуваних частин логіки, які розробники можуть використовувати в різних додатках. Технічно, Nest може працювати з будь-яким Nodejs HTTP фреймворком, якщо написати для нього адаптер. З коробки Nest підтримує два HTTP фреймворки express та fastify. Можна вибрати той, який найкраще відповідає індивідуальним потребам.

Яка б платформа не використовувалася, вона надає власний API. Відповідно як NestExpressApplication і NestFastifyApplication.

4.3 Керівництво користувача

Контролери відповідальні за обробку вхідних запитів та повернення відповіді клієнту. Мета контролера – отримання запитів. Механізм маршрутизації контролює, який контролер отримає запит. Найчастіше контролери містять більше одного маршруту, які відповідають за різні дії.

Щоб створити побутовий контролер, потрібно використовувати декоратор класу. Декоратоти додають у клас необхідні дані для того, щоб Nest зміг побудувати карту маршрутів (прив'язати запити до відповідних контролерів).

У наступному прикладі використовується декоратор @Controller() необхідний визначення базового контролера. Ми передамо в декоратор необов'язковий префікс маршруту – cats. Передача префікса маршруту в декоратор @Controller() дозволяє легко групувати пов'язані маршрути і позбутися дублювання коду, що повторюється. Наприклад, ми можемо згрупувати набір маршрутів, що відповідають за взаємодію з сутністю клієнта в контролері з префіксом /customers, що позбавить від повторення цього префікса в кожному маршруті цього контролера.

```
import { Controller, Get } from '@nestjs/common';

@Controller('cats')
export class CatsController {
  @Get()
  findAll(): string {
    return 'This action returns all cats';
  }
}
```

Рисунок 4.3 – Приклад контролера CatsController

Декоратор HTTP дієслова @Get() перед методом findAll() каже фреймворку створити обробник для конкретного ендпоінта. Цей ендпоінт відповідає HTTP дієслову (в даному випадку - GET) та шляхи маршруту. Що таке шлях маршруту? Шлях маршруту обробника визначається за допомогою конкатенування префікса заданого в декораторі @Controller(<path_prefix>) і шляху заданому в декораторі запиту. Зараз у нас заданий префікс cats для кожного маршруту і не задано жодного шляху в декораторі @Get(), Nest зіставить обробник цього маршруту з GET /cats запитом. Як було вже згадано, шлях маршруту містить необов'язковий префікс шляху визначений у декораторі контролера та рядок визначений у декораторі методу запиту. Наприклад, префікс шляху customers поєднаний із декоратором @Get('profile') створить маршрут зіставлений із запитом GET /customers/profile.

У прикладі вище, коли відбувається GET запит на цей ендпоінт, Nest надсилає запит метод findAll(). Зауважте, що ім'я методу вибрано абсолютно довільне. Очевидно, що ми повинні визначити метод прив'язки до маршруту, але Nest не надає жодного значення вибраному імені методу.

Постачальники є основним поняттям у Nest. Багато базових класів Nest можуть розглядатися як постачальники – послуги, репозиторії, фабрики, помічники тощо. Основна ідея провайдера полягає в тому, що він може

вводити залежності; це означає, що об'єкти можуть створювати різні відносини один з одним, а функцію "підключення" екземплярів об'єктів можна значною мірою делегувати системі середовища виконання Nest. Постачальник — це просто клас, анотований декоратором @Injectable().

Ця служба відповідатиме за зберігання та пошук даних, і призначена для використання CatsController, тому це хороший кандидат, щоб бути визначеним як постачальник. Таким чином, ми прикрашаємо клас за допомогою @Injectable().

```
import { Injectable } from '@nestjs/common';
import { Cat } from './interfaces/cat.interface';

@Injectable()
export class CatsService {
  private readonly cats: Cat[] = [];

  create(cat: Cat) {
    this.cats.push(cat);
  }

  findAll(): Cat[] {
    return this.cats;
  }
}
```

Рисунок 4.4 – Приклад сервісу CatsService

Модуль — це клас, анотований декоратором @Module(). Декоратор @Module() надає метадані, які Nest використовує для організації структури програми.

Кожна програма має принаймні один модуль, кореневий модуль. Кореневий модуль є відправною точкою, яку Nest використовує для побудови графіка програми — внутрішня структура даних, яку Nest використовує для розв’язання взаємозв’язків і залежностей між модулем і постачальником. Хоча дуже маленькі програми теоретично можуть мати

лише кореневий модуль, це не типовий випадок. Ми хочемо підкреслити, що модулі настійно рекомендуються як ефективний спосіб організації ваших компонентів. Таким чином, для більшості додатків результуюча архітектура буде використовувати кілька модулів, кожен з яких інкапсулює тісно пов'язаний набір можливостей.

CatsController і CatsService належать до одного домену програми. Оскільки вони тісно пов'язані, має сенс перемістити їх у функціональний модуль. Функціональний модуль просто організовує код, відповідний для певної функції, зберігаючи код упорядкованим і встановлюючи чіткі межі. Це допомагає нам управляти складністю та розвиватися за принципами SOLID, особливо в міру зростання розміру програми та/або команди.

```
import { Module } from '@nestjs/common';
import { CatsController } from './cats.controller';
import { CatsService } from './cats.service';

@Module({
  controllers: [CatsController],
  providers: [CatsService],
})
export class CatsModule {}
```

Рисунок 4.5 – Приклад модуля CatsModule

Висновки до розділу 4

В межах розділу було реалізовано програмне забезпечення обміну повідомленнями на основі REST API та Websocket з використанням вищенаведених технологій та мов програмування.

В ході роботи було проведено тестування вебзастосунка. Застосунок повністю відповідає функціональним та нефункціональним вимогам, поставленних на початку розробки програмного забезпечення. Вебзастосунок повністю готовий до використання.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було досягнуто її головної мети - підвищення зручності процесу обміну онлайн повідомленнями шляхом розробки вебзастосунку основі REST API та websocket-ів.

В ході описання вебзастосунка був проведений детальний аналіз предметної області завдання – існуючих аналогів. Насамперед, був визначен функціонал, переваги та недоліки кожного з існуючих аналогів.

Проведений аналіз визначив засоби та методи, призначені для вирішення поставлених завдань за допомоги створення програмного коду для досягнення основної мети роботи.

Було проаналізовано методики створення баз даних, основні завдання баз даних, основні складові та сутності баз даних вебзастосунка обміну повідомленнями. Було спроектовано логічну модель даних, яка була представлена схемою даних у вигляді ER-діаграми.

Були спроектовані сценарії та діаграми до програмного забезпечення. У ході роботи були реалізовані три основних типа сценаріїв – коротка, поверхнева та повна форми. Аналогічно з сценаріями, були спроектовані такі діаграми як: діаграма варіантів використання системи, діаграма діяльності, діаграма класів, діаграма розгортання, діаграма послідовності

Результатом виконання роботи є спроектований вебзастосунок обміну повідомленнями між користувачами на основі REST API та Websocket. Проведено тестування вебзастосунка. Застосунок повністю відповідає функціональним та нефункціональним вимогам, поставленних на початку розробки програмного забезпечення. Вебзастосунок повністю готовий до використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Guide for ReactJS on Youtube: веб-сайт, URL: https://www.youtube.com/watch?v=gb7gMluAeao&list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8&ab_channel=IT-KAMASUTRA (дата звертання 30.01.2022);

2. Розширений Backend на Node.js. Nest js Повний курс & Docker: веб-сайт. URL: https://www.youtube.com/watch?v=dDeWWQWMM-Y&ab_channel=UlbiTV (дата звертання 10.02.2022);

3. React Core Documentation: веб-сайт. URL: https://uk.reactjs.org/ (дата звертання 15.02.2022);

4. A progressive Node.js framework for building efficient, reliable and scalable server-side applications: веб-сайт. URL: https://nestjs.com/ (дата звертання 20.02.2022);

5. Startup Guide in MySQL: веб-сайт. URL: hhttps://wiki.gentoo.org/wiki/MySQL/Startup_Guide/ru (дата звертання 22.02.2022);

6. Socket IO Documentation: веб-сайт. URL: https://socket.io/ (дата звертання 05.03.2022);

7. TypeORM for NestTS: веб-сайт. URL: https://typeorm.io/ (дата звертання 07.03.2022);

8. React-Boostrap: веб-сайт. URL: https://react-bootstrap.github.io/ (дата звертання 15.03.2022);

9. React Redux Toolkit: веб-сайт. URL: https://redux-toolkit.js.org/ (дата звертання 18.03.2022);

10. React Context API: веб-сайт. URL: https://uk.reactjs.org/docs/context.html (дата звертання 21.03.2022);

Додаток А

Для клієнт сторони була розроблена маршрутизація задля можливості користувача переходити на інші сторінки вебзастосунку. Розроблений код приведений нижче відповідає за реалізацію навігації.

```
import {BrowserRouter, Route, Routes} from "react-router-dom";
import Home from "./pages";
import Login from "./pages/login";
import Register from "./pages/register";
import Settings from "./pages/settings";
import Rooms from "./pages/rooms";
import Users from "./pages/users";


export const AppRoutes = () => {
    return (
        <BrowserRouter>
            <Routes>

                <Route path="/" element={Home}>
                    <Route path=":hash" element={Home}/>
                </Route>

                <Route path="rooms" element={Rooms}>
                    <Route path=":hash" element={Rooms}/>
                </Route>

                <Route path="users" element={Users}>

                </Route>

                <Route path="/login" element={Login}/>
                <Route path="/register" element={Register}/>
                <Route path="/settings" element={Settings} />
            </Routes>
        </BrowserRouter>
    )
}
```

Робота з аутентифікаційними даними є доволі складною та потребує місце для зберігання отриманих даних з серверної частини вебзастосунку, а також необхідно модифікувати дані в залежності від статусу аутентифікації користувача. У ході роботи було розроблено сховище даних яке було реалізовано завдяки React Redux Toolkit. Створений код приведено нижче.

```
import {createAsyncThunk, createSlice} from "@reduxjs/toolkit";
import {AuthState, EditUserDto, LoginDto} from "../../types";
import {AuthService} from "../../services/AuthService";
import {UserService} from "../../services/UserService";

const rejectionReducer = (state: AuthState) => {
    state.isLoaded = true;
}

export const refresh = createAsyncThunk(
    'auth/refresh',
    async (_, {dispatch}) => {
        try {
            const {data} = await AuthService.refresh();
            const {accessToken} = data.data.tokens;

            localStorage.setItem('accessToken', accessToken);

            dispatch(authenticate(data.data.user))
        } catch (e) {
            throw e;
        }
    }
);

export const login = createAsyncThunk(
    'auth/login',
    async (loginData: LoginDto, {dispatch}) => {
        try {
            const {data} = await AuthService.login(loginData);
            const {accessToken} = data.data.tokens;

            localStorage.setItem('accessToken', accessToken);

            dispatch(authenticate(data.data.user))
        } catch (e) {
            throw e;
        }
    }
);

export const logout = createAsyncThunk(
```

```
    'auth/logout',
    async (_, {dispatch}) => {
        try {
            await AuthService.logout();
            dispatch(exit())
        } catch (e) {
            throw e;
        }
    }
);

export const editPersonalData = createAsyncThunk(
    'auth/editPersonalData',
    async (fd: FormData, {dispatch}) => {
        try {
            const {data} = await UserService.edit(fd);
            dispatch(setUser(data.data));
        } catch (e) {
            throw e;
        }
    }
)

const initialState: AuthState = {
    user: null,
    isLogged: false,
    isLoaded: false,
}

const authSlice = createSlice({
    name: 'auth',
    initialState,
    reducers: {

        authenticate(state, {payload}) {
            state.user = payload;
            state.isLogged = true;
            state.isLoaded = true;
        },

        setUser(state, {payload}) {
            state.user = payload;
        },

        exit(state) {
            state.user = null;
            state.isLogged = false;
        },

    },
    extraReducers: (builder => {
```

```
        builder.addCase(refresh.rejected, rejectionReducer);
        builder.addCase(login.rejected, rejectionReducer);
    })
})

export const {authenticate, setUser, exit} = authSlice.actions;

export default authSlice.reducer;
```

Як було проілюстровано вище, робота з аутентифікацією потребує багатьох дій для її реалізації. Для того щоб мати зв'язок з серверною частиною вебзастосунку було створено сервіс з так званими "ендпоінтами".

```
import $api from "../http";
import {RegisterDto, LoginDto} from "../types";

export class AuthService {

    static async register(registerDto: RegisterDto) {
        return await $api.post('/auth/register', registerDto);
    }

    static async login(loginDto: LoginDto) {
        return await $api.post('/auth/login', loginDto);
    }

    static async refresh() {
        return await $api.get('/auth/refresh');
    }

    static async logout() {
        return await $api.get('/auth/logout');
    }

}
```

Головною особливістю проекту є вебсокети завдяки яким вдалось реалізувати можливість отримувати повідомлення у режиму реального часу. Код реалізації приведено нижче.

```
import React, {createContext, FC, ReactNode, useEffect,
useState} from 'react';
import {io, Socket} from "socket.io-client";
import {ChatDto, MessageDto, UserDto} from "../../types";
import {
    addChat,
    addMessage,
    changeChat,
```

```
    changeMessage,
    changeUser,
    setChat
} from "../../store/slices/chat";
import {SoundService} from "../../services/SoundService";
import SnackbarMessage from "../partials/SnackbarMessage";
import {useTypedSelector} from "../../hooks/useTypedSelector";
import {useAppDispatch} from "../../store";
import {useSnackbar} from "../../hooks/useSnackbar";

interface SocketProviderProps {
    children: ReactNode;
}

export const SocketContext = createContext<null | Socket<any,
any>>(null);

const SocketProvider: FC<SocketProviderProps> = ({children}) =>
{
    const [lastMessage, setLastMessage] = useState<null |
MessageDto>(null);
    const [socket, setSocket] = useState<null | Socket<any,
any>>(null);

    const {chats} = useTypedSelector(state => state.chat);
    const {user} = useTypedSelector(state => state.auth);

    const dispatch = useAppDispatch();

    const {snackbar} = useSnackbar();

    useEffect(() => {
        if (!socket) {
            setSocket(io('localhost:5000', {
                transportOptions: {
                    polling: {
                        extraHeaders: {
                            Authorization: `Bearer
${localStorage.getItem('accessToken')}`,
                        }
                    }
                }
            }));
        }
    }, []);

    useEffect(() => {
        if (socket) {

            socket.on('new-chat', (chat: ChatDto) => {
                dispatch(setChat(chat));
```

```
                dispatch(addChat(chat));

                setLastMessage(chat.messages[0]);
            });

            socket.on('chat-message', (message: MessageDto) => {
                dispatch(addMessage(message));
                setLastMessage(message);
            });

            socket.on('read-message', (message: MessageDto) => {
                dispatch(changeMessage(message));
            });

            socket.on('logout', (user: UserDto) => {
                dispatch(changeUser(user));
            });

            socket.on('login', (user: UserDto) => {
                dispatch(changeUser(user));
            });

            socket.on('block-unblock', (chat: ChatDto) => {
                dispatch(changeChat(chat));
            });

            socket.on('mute-unmute', (chat: ChatDto) => {
                dispatch(changeChat(chat));
            });

            return () => {
                socket.disconnect();
            }
        }
    }, [socket]);

    useEffect(() => {
        if (lastMessage) {
            const chat = chats.find(({id}) => id ===
lastMessage.chatId);

            if (!chat?.isMuted && lastMessage?.user.id !==
user?.id) {
                SoundService.playSound();
                snackbar(<SnackbarMessage
user={lastMessage.user} message={lastMessage}/>);
            }
        }
    }, [lastMessage]);

    return (
```

```
        <SocketContext.Provider value={socket}>
            {children}
        </SocketContext.Provider>
    );
};

export default SocketProvider;
```