**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** *Брежнєва Кирила Миколайовича*
(ПІБ)

**академічної групи** *122-18-2*
(шифр)

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

**освітньої програми** *Комп'ютерні науки*
(назва освітньої програми)

**на тему:** *Розробка клієнтської частини соціального месенджера на основі JavaScript/React*

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

**Дніпро**
**2022**

Міністерство освіти і науки України
**НТУ «Дніпровська політехніка»**

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

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

# ЗАВДАННЯ
## на кваліфікаційну роботу

*бакалавра*
(назва освітньо-кваліфікаційного рівня)

**студента** *122-18-2* *Брежнєва Кирила Миколайовича*
(група) (прізвище та ініціали)

**тема кваліфікаційної роботи** *Розробка клієнтської частини соціального месенджера на основі JavaScript/React*

затверджена наказом ректора НТУ «ДП» від «» червня 2022 р. №

| Розділ | Зміст виконання | Термін виконання |
|---|---|---|
| ***Спеціальний*** | ***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми*** | ***13.05.2022 р.*** |
| ***Економічний*** | ***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки*** | ***27.05.2022 р.*** |

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

Завдання прийняв до виконання _____ *Брежнєв К.М.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

# РЕФЕРАТ

Пояснювальна записка: 99 с., 35 рис., 3 дод., 26 джерел.

Об’єкт розробки: клієнтська частина системи соціального месенджеру.

Мета кваліфікаційної роботи: створення frontend частини системи соціального месенджеру для полегшення комунікації в сучасних умовах, що дозволяють використання новітніх технологій, такі як смартфони, планшети, ноутбуки та ПК для безкоштовного текстового спілкування без використання SMS.

У вступі розглядається аналіз, та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється 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 платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на її створення.

Практичне значення полягає у створенні додатка, що надає можливість спілкування між людьми, зберігання контактів для швидкого пошуку необхідної людини, можливість редагування власного профілю, імені користувача, зміні паролю, що забезпечує комфортне користування інформаційною системою.

Актуальність інформаційної системи визначається великим попитом на системи, що спрощують комунікацію між людьми у сімейних, ділових чи інших відносинах, скорочують зусилля, що повинні бути прикладені для обговорення тих чи інших питань, та підвищують ефективність розподілу часу на спілкування.

Список ключових слів: МЕСЕНДЖЕР, ЧАТ, ПРОФІЛЬ, МЕНЮ, МОБІЛЬНА ВЕРСІЯ, ВЕБ-ДОДАТОК, ФРЕЙМВОРК, ФРОНТЕНД, БРАУЗЕР.

## ABSTRACT

Explanatory note: 99 p., 35 figs, 3 apps, 26 sources.

Object of development: the client part of the social messenger system.

The purpose of the qualification work: to create a frontend part of the social messenger system to facilitate communication in modern conditions that allow the use of the latest technologies such as smartphones, tablets, laptops and PCs for free text communication without SMS.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the task.

In the first section the subject branch is analyzed, the urgency of the task and the purpose of development are determined, the statement of the task is formulated, the requirements to the software implementation, technologies and software are specified.

The second section analyzes the available solutions, selected platforms for development, designed and developed the program, describes the program, algorithm and structure of its operation, as well as calling and loading the program, determines the input and output data, describes the parameters of hardware.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The relevance of the information system is determined by the high demand for systems that facilitate communication between people in family, business or other relationships, reduce the effort that must be made to discuss certain issues, and increase the efficiency of time allocation for communication.

List of keywords: MESSENGER, CHAT, PROFILE, MENU, MOBILE VERSION, WEB APPLICATION, FRAMEWORK, FRONTEND, BROWSER.

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

СУБД – система управління базами даних;

ПК – персональний комп’ютер;

ІТ – інформаційні технології;

SPA – Single Page Application;

FE – FrontEnd;

BE – BackEnd

JS – JavaScript;

JSON – JavaScript

CSS – Cascading Style Sheets;

HTML – Hypertext Markup Language;

HTTP – Hypertext Transfer Protocol;

CMS – Content Management System;

PHP – PHP Hypertext Preprocessor;

API – Application Programming Interface;

SMS – Short Message Service;

GSM – Global System for Mobile Communications;

XSS – Cross-site Scripting;

SQL – Structured Query Language;

DDOS – Distributed Denial of Service

DOM – Document Object Model;

CORS – Cross-origin Resource Sharing;

SSR – Server-Side Rendering;

CSR – Client-Side Rendering;

REST – Representational State Transfer;

STOMP – Streaming Text Oriented Messaging Protocol;

XML – Extended Markup Language;

WS – WebSocket.

# ВСТУП

Розроблена інформаційна система призначена для полегшення комунікації між людьми з будь якої точки світу.

На сьогодні, велика частина людей може спілкуватися не лише за допомогою мобільної телефонії та SMS, а також з використанням інших інформаційних технологій. Інтернет є вдома, та у будь якому закладі загального користування, будь то кафе, торговельно-розважальний центр, чи школа.

Потреба у спілкуванні для людства є однією з найважливіших, адже комунікація це шлях до реалізації та поліпшення стосунків з іншими людьми, задоволення потреб особистістю, та спосіб досягнення соціально важливої мети.

Інтернет спілкування вигідніше у контексті часу, бо написати декілька повідомлень простіше, аніж дзвонити, щоб сказати якусь коротку та нетермінову фразу, тим більше, якщо хтось з співбесідників зайнятий або за кермом. Також є переваги у ціні повідомлення, адже відправляти повідомлення через інтернет дешевше, ніж SMS через GSM зв’язок.

Також месенджер може слугувати єдиним зв’язком у критичних ситуаціях, коли інші види зв’язку недоступні. В наш час, коли в кожну хвилину щось може трапитись та ситуація навколо змінюється миттєво, тримання в курсі важливих людей щодо навколишньої ситуації може приймати державний масштаб.

Тож месенджер – це невід’ємний додаток у сучасному світі, що підтверджується словами вище і актуалізує проблему. Практичне значення підкреслюється тим, що деякі популярні месенджери мають великі проблеми з захистом даних для зовнішніх атак. А інші можуть мати проблеми навіть всередині компанії, що розробляє такий месенджер.

Зазвичай, весь світ цінує якісні складові системи, що використовує. Це не обов’язково повинна бути інформаційна система, також це відноситься до інших звичних нам речей, від системи охорони здоров’я до власного автомобіля чи смартфону.

Отже метою цієї дипломної роботи є розробка клієнтської частини месенджеру, що вирішує всі перераховані проблеми:

- спрощення комунікації між будь якими людьми, що знаходяться будь якій відстані один від одного географічно;
- безкоштовність користування, що дозволяє використати месенджер як для сімейного спілкування, так і для роботи та ведення бізнесу;
- безпеку збереження особистих даних, історії повідомлень та контактів;
- швидкість роботи застосунка, що дає користувачеві відчуття максимального комфорту;
- можливість редагування профілю користувача, та забезпечення анонімності спілкуванні.

Розробленою інформаційною системою зможе користуватися будь яка особа, без вікового обмеження.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1. Загальні відомості з предметної галузі

На сьогодні, у світі є велика кількість різних сайтів та веб-додатків. Вони використовуються для безлічі цілей:

- спілкуванні у різного типу месенджерах, в тому числі відео дзвінки;
- купівлі товарів в інтернет магазинах;
- замовленні квитків на транспорт;
- бронювання готелів;
- переглядання та обговорення новин;
- відстежування на карті поширення епідемій або навіть військ противника у режимі реального часу.

Сайти можна класифікувати як односторінкові та багатосторінкові. Кожна з цих класифікацій в свою чергу поділяється на статичний сайт, та динамічний.

Односторінковий сайт – це сайт, як можна визначити з назви, який має одну сторінку, на відміну від багатосторінкового сайту.

Статичний сайт – це такий сайт, на якому контент не змінюється у режимі реального часу, а генерується сервером з кожним запитом до сторінки. У динамічному веб додатку контент завантажується частинами.

Статичний односторінковий сайт, це зазвичай сайт, що має мету афішувати щось, блог, портфоліо або резюме автора. Такі сайти дуже легко створити та швидко розгорнути. Це прості сайти, що не мають ніякої логіки роботи на сервері, БД, тощо. Вони відображають інформацію, яка може помістится на одну сторінку. Такі сайти робляться за допомогою мови розмітки HTML, та дуже рідко PHP.

Статичний багатосторінковий сайт – це сайт, що має багато сторінок, на яких контент згенерований сервером. Такі сайти зазвичай робляться на чистому HTML іноді з використанням препроцесорів, наприклад таких як Pug, або

серверною мовою програмування PHP. Також, можуть використовуватися CMS системи для управління і налаштування контенту. Ці сайти можуть мати базу даних, але це не є обов'язковим.

Користувач заходить за посиланням на сайт та йому відображається сторінка з розширенням .html або .php. Контент на цій сторінці є статичним та вже не буде редагуватися. Тобто, при зміні якихось елементів, для відображення змін потрібно перезавантажити сторінку. Такий спосіб створення сайтів вже не актуальний у зв'язку з появою нових та сучасних технологій. Як приклади можна навести інтернет форуми, що мали популярність до появи месенджерів, приблизно у 2000-2009 роках.

Динамічний односторінковий веб додаток – це на теперішній момент один з найпоширеніших форматів створення веб додатку. Такі сайти робляться за допомогою фреймворків для створення SPA, таких наприклад як React. Такі додатки майже завжди мають свою базу даних, та серверну (BE) частину.

Контент у додатках такого типу генерується у JSON форматі на бекенді (сервері), та відправляється у такому ж вигляді на фронтенд (клієнт), де підставляється у необхідне місце, та відповідним чином форматується. При специфічних діях на сайті, відправляється запит на бекенд після чого сервер витягує необхідну інформацію з БД, та відправляє у зворотному напрямку, на фронтенд. HTML сторінка у сайті всього одна, тобто користувач попадає на цю сторінку при переході на сайт, де за допомогою JS та клієнтських потужностей браузеру відбувається рендер, та форматування. Перехід на інший екран відбувається непомітно, також завдяки JS.

Насправді, користувач весь час користування додатком лишається на тій самій сторінці, та нікуди з неї не дівається. Саме тому, це називається SPA – Single Page Application, тобто додаток, що працює на одній сторінці. Прикладом такого додатка можна навести месенджер, що розробляється у цієї кваліфікаційної роботі.

Динамічний багатосторінковий веб додаток – теж саме, що і односторінковий, але з відмінністю у кількості точок входу. Frontend

фреймворки такі як React можна використовувати не тільки як повноцінну заміну HTML/PHP, а й як допоміжний засіб. Тобто сайт може мати багато HTML сторінок, але лише на деяких з них буде вмонтований фронтенд фреймворк. Такі додатки складні у реалізації та підтримці, та не є популярними.

Переваги сайтів на основі мови розмітки HTML:

- мова розмітки проста в осягненні та не потребує знання програмування;
- сайт може працювати локально без встановлення додаткових плагінів та додатків для підняття серверів;
- хостинг, на якому розміщено такий сайт не потребує спеціальних налаштувань;
- відсутність реальних програмних компонентів майже унеможливлює злам системи, окрім системи самого хостингу, або підбору пароля до адміністраторської системи.

Недоліки сайтів на основі мови розмітки HTML:

- складність зміни структури сайту через монолітність системи;
- неможливість розбиття повторюваних блоків коду на компоненти, що імпортуються у файли (за виключенням використання препроцесору Pug, Handlebars, тощо або PHP;
- неможливість використання функціональних блоків, такі як коментарі, оцінки статей, чати та інше;
- неможливість використання CMS, тобто щоб змінити контент на сайті, потрібно вручну правити HTML код.

Більшість цих недоліків компенсується використанням різного роду фронтенд фреймворків, такі як React, Vue.js, Angular. Кожен з них створений щоб вирішувати одну й ту саму задачу: створення веб додатків, але кожен робить це різним чином, через що у різних ситуаціях використання того або іншого фреймворку є більш доречним.

SPA фреймворки мають достатньо переваг, щоб почати їх використовувати:

- швидкість завантаження сторінки на девайс користувача дуже висока, навіть при слабкому підключенні до мережі;

– швидкість переходу між екранами у додатку також є надвисокою;
– використовує менше трафіку, через те, що завантажує сторінку тільки один раз, та потім лише приймає відповіді на запити у легковажному JSON форматі.

Також, фронтенд фреймворки мають свої недоліки, такі як:

– відсутність нормальної індексації сторінки у пошукових системах (Google, Bing, тощо);
– необережне ставлення до програмування може призвести до слабкої продуктивності системи у користувачів з старими або дешевими гаджетами, з яких вони використовують додаток;
– значно більш велика можливість ураження XSS атаками, ніж багатосторінкові сайти.

Для подолання зазначених вище недоліків SPA додатків, було створене SSR рішення. Звичайні динамічні односторінкові застосунки використовують CSR, тобто рендеринг на стороні клієнта, використовуючи потужності браузера відвідувача сайту. Такі сайти мають усі зазначені вище SPA недоліки.

SSR означає рендеринг SPA сторінки на сервері, тобто готову сторінку яку згенерував сервер, відправляють на запит клієнта.

Це може здаватися, як той випадок, якого технології намагалися уникнути, адже SPA фреймворки та CSR було вигадано спеціально для того, щоб зменшити навантаження на сервер. Але з появою SSR, здається, що це анульовано, бо для генерації контенту, серверу потрібно більше потужності. Насправді так і є, бо різниця розміру між відправленим контентом HTML та JSON може складати у сотні разів.

SSR вирішує проблему пошукових запитів, через те, що відправляє готову сторінку, та пошуковому двигуну не потрібно компілювати JS код, для розуміння того, що відображається на сторінці. Деякі сучасні пошукові системи вже можуть розуміти SPA додатки, але далеко не всі. Саме для цього було створено SSR.

Також, з недоліків серверного рендерингу сторінки є те, що такі сторінки ще більш уразливі до JS хакерських атак, ніж SPA додатки.

З цього випливає те, що під кожен випадок є свій інструмент, та не можна зупиняти вибір з  SPA CSR, або SPA SSR на одному з них, без достатнього розгляду обох варіантів.

## 1.2. Призначення розробки та галузь застосування

Об’єктом розробки є клієнтська частина інформаційної системи «Messenger Memessenger», у якому користувач може знаходити інших користувачів за електронною адресою та іменем профілю, та писати їм особисті повідомлення.

Так як месенджер є важливою частиною соціального життя особи в сучасному світі, та є часто використовуваним додатком, необхідно, щоб він був достатньо простим для осягнення, та комфортним у використанні.

Розроблена frontend частина месенджеру призначена для:

- спрощення комунікації між двома особами, що використовують даний месенджер;
- забезпечення користувачам месенджеру комфорту у використанні додатку, відправленні та редагуванні повідомлень;
- допомога у спілкуванні між людьми, що знаходяться на будь-якій відстані, у будь який час доби.

## 1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки «виконання кваліфікаційної роботи» є:

- освітня програма 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка»
– завдання на кваліфікаційну роботу на тему «Розробка клієнтської частини соціального месенджера на основі JavaScript/React».

## 1.4. Постановка завдання

Завданням кваліфікаційної роботи є розробка клієнтської частини соціального месенджера на основі JavaScript/React.

Система призначена для надання можливості комунікації між користувачами посередньо чатів у веб додатку.

Додаток повинен надавати можливості наступного функціоналу:

– можливість реєстрації та логіну у додаток з допомогою особистих даних;
– користувачам відправляти особисті повідомлення до інших користувачів;
– редагувати раніше відправлені повідомлення, або видаляти їх, ті, що були відправлені активним користувачем
– шукати інших користувачів, щоб відправити їм повідомлення та/або додати до списку контактів;
– додавати користувачів до списку контактів, та видаляти звідти;
– редагувати власний профіль користувача, змінювати особисті дані у ньому;
– змінювати пароль для входу у застосунок;
– переглядати веб додаток з девайсу з будь-якою роздільною здатністю екрану, між 375 та 2800px.

Для досягнення мети у створенні готового додатку, необхідно:

– дослідити предметну галузь поставленої задачі;
– створити алгоритм реалізації завдання, у тому числі для окремих компонент веб додатку, таких як БД, BE, FE частини;

– створити базу даних, що буде використовуватися серверною частиною програми;
– створити серверну частини застосунку;
– створити клієнтську частину застосунку.

## 1.5. Вимоги до програми або програмного виробу

### 1.5.1. Вимоги до функціональних характеристик

Для досягнення цілей розробки, готовий програмний продукт повинен виконувати наступні дії:

– надавати доступ до додатку через веб-браузер на пристрої користувача: ПК, ноутбуці, планшеті, смартфоні, тощо;
– зберігати історію повідомлень, список контакті, та особисті дані профіля користувачів;
– за запитом клієнту зчитувати вищенаведені дані з БД та відображати їх у браузері користувача;
– надавати можливість забезпечити цілісність та інтегральність даних від фізичних пошкоджень серверу БД або зламу / випадкового видалення;
– надавати можливість записувати нові дані у БД, через відправку нових повідомлень, створення нових користувачі, тощо.

Для реалізації зазначених функцій веб-додатку, повинні бути реалізовані такі дії:

– наявність спеціального конфігураційного файлу для швидкого розгортання та введення в експлуатацію застосунку;
– автоматичне створення копії бази даних, що фізично знаходиться на іншому сервері;
– можливість інтеграції з основними сучасними хмарними сервісами.

### 1.5.2. Вимоги до інформаційної безпеки

Інформаційна система повинна бути добре захищеною від зовнішніх та внутрішніх інформаційних загроз, фізичного пошкодження та мати регламент позаштатних ситуацій.

База даних повинна мати функцію резервного копіювання даних, що робиться на віддалений захищений сервер принаймні раз на тиждень та при виникненні будь-якої позаштатної ситуації.

До позаштатних ситуацій відносяться:

- позапланові відключення електропостачання;
- вихід з ладу компонентів системи, як програмних, так и апаратних;
- помилки розробника або адміністратора, що працюють з базою даних напряму у цей момент (міграція, оновлення, клонування, об’єднання);
- навмисна фізична шкода даним або системі у цілому (пошкодження серверу БД);
- стійкість до хакерських атак типу SQL ін’єкції;
- природні або техногенні катастрофи (пожежа, землетрус, ракетний удар);

Серверна та клієнтські частини додатку також повинні бути захищені, та можливість забезпечувати автономність роботи використовуючи власні резервні джерела потужності.

Серверна частина повинна забезпечувати захист від різного виду хакерських атак, та обмежувати доступ до точок запитів (endpoint) за допомогою CORS налаштувань.

Клієнтська частина також повинна мати захист від різного роду спроб злами сайту, таких як XSS, та мати сильну валідацію активних текстових полів проти SQL ін’єкцій.

Необхідно, щоб клієнт мав можливість фільтрувати доступ користувачів до сайту за допомогою DDOS-firewall утиліти, щоб уникнути атаки, що спрямована на тимчасове виведення з ладу сайту шляхом підвищення навантаження серверу відвідуванням великою кількістю фейкових користувачів.

### 1.5.3. Вимоги до складу та параметрів технічних засобів

Для використання інтернету і в тому числі інтернет спілкування більшість людей використовують переважно мобільні комп'ютерні системи, до яких відносяться смартфони та планшети. Через це, система повинна бути сумісна з якомога більшою кількістю мобільних пристроїв, що є в наявності у суспільстві.

При використання соціального месенджеру «Messenger Memessenger» з ПК, чи ноутбуку, веб-браузер користувача повинен бути версії не нижче зазначеної, в залежності від тієї, що експлуатується:

- Internet Explorer 11;
- Edge Browser 12;
- Mozilla Firefox 52;
- Google Chrome 57;
- Safari 10.1;
- Opera 44.

Відповідно, комп'ютер користувача повинен підтримувати версію Windows не нижче ніж 7, до якої є офіційні вимоги [1]:

- 1 гігагерц (ГГц) або швидше 32-розрядний (x86) або 64-розрядний (x64) процесор
- 1 гігабайт (ГБ) RAM (32-розрядна) або 2 ГБ RAM (64-розрядна)
- 16 ГБ вільного місця на жорсткому диску (32-розрядний) або 20 ГБ (64-розрядний)
- Графічний пристрій DirectX 9 з драйвером WDDM 1.0 або вище

При використанні месенджеру на мобільних платформах, таких як Android чи iOS, основною вимогою є версія браузера, що не є нижче ніж зазначені:

- Safari on iOS 10.3;
- Android Browser (після Android 5), Chrome for Android, Firefox for Android з версією ядра Chromium не нижче 58;
- Opera Mobile 64;
- Samsung Internet 6.2.

Другою вимогою є обсяг оперативної пам’яті, не менше ніж 1 ГБ.

За наявності апаратного та програмного забезпечення, що має характеристики, що зазначені вище, або краще, розроблений веб-додаток буде функціонувати належним чином.

### 1.5.4. Вимоги до інформаційної та програмної сумісності

Для розробки клієнтської частини системи соціального месенджеру «Messenger Memessenger» використовується мова програмування JavaScript та фреймворк React. Проект створюється через офіційний шаблон від Facebook – Create React App [2].

Розробка ведеться компонентним підходом, з використанням компонент функціонального типу, та дотриманням основних принципів SOLID архітектури [3].

# РОЗДІЛ 2
# ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1. Функціональне призначення системи

Завданням кваліфікаційної роботи є реалізація клієнтської частини соціального месенджера на основі JavaScript/React.

Призначення розробленої системи:

- надати користувачам спосіб реєструватися та заходити у додаток під своїми особистими даними (логін, пароль)
- надати користувачам спосіб обмінюватися повідомленнями;
- надати користувачам можливість додавати (видаляти) один одного у список контактів;
- надати користувачам можливість редагування власних особистих даних.

Розроблена система реалізує наступний функціонал:

- зручний та простий інтерфейс для спілкування;
- формування контенту сторінки згідно даних, що приходять до клієнтської частини з серверу, унікально для кожного користувача;
- пошук інших користувачів в базі даних;

Для підтримання усіх пунктів поставленої задачі, готовий веб-додаток забезпечує виконання таких операцій:

- надання рівноправного доступу до сайту користувачам на ПК, смартфоні або іншому пристрої;
- зчитування даних для входу в застосунок з бази даних;
- пошук користувачів у базі даних;
- відправка, прийняття, редагування та видалення повідомлень у режимі реального часу, без оновлення сторінки за допомогою протоколу STOMP Over WebSocket;
- зміна даних користувача у базі даних

## 2.2. Опис застосованих математичних методів

Через особливості предметної галузі розв'язуваної задачі, що не передбачає застосування математичних методів, при розробці клієнтської частини системи соціального месенджеру, жоден математичний метод не використовувався.

## 2.3. Опис використаних технологій та мов програмування

### 2.3.1. Опис мови JavaScript

Клієнтська частина системи соціального месенджеру виконана за допомогою мови JavaScript та фронтенд фреймворку React. При виконанні роботі було використано функціональний компонентний підхід, та основні принципи SOLID архітектури [3].

JavaScript – це клієнтська мова програмування, яку використовують для створення динамічної та складної поведінки веб сторінки, яку неможливо реалізувати у HTML або CSS. Сучасні веб сайти складно уявити без використання JavaScript, адже це єдиний спосіб зв’язку між клієнтом (власне веб сторінкою), та сервером, на якому опрацьовано основну логіку програми. JavaScript також можна використовувати для створення анімацій на сайті, підтримці відеоконтенту та навігацією між сторінками (екранами).

«JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв вебсторінок, що надає можливість на боці клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб сторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні

архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу. [4]»

Робочою назвою нової мови була Mocha, яка була змінена на LiveScript в перших двох бета-версіях браузера Netscape 2.0. А дещо пізніше, користуючись популярністю бренду Java, LiveScript був перейменований на JavaScript і третя бета-версія (2.0B3) Netscape 2.0 вже вийшла з сучасною назвою. Для цього була придбана відповідна ліцензія у компанії Sun Microsystems, що володіла брендом Java.

Мова JavaScript використовується для програмування клієнтської частини не тільки через те, що вона популярна, а й через те, що вона стандартизована.

Мова має повну підтримку HTML та CSS, що дозволяє використати її для створення анімацій, роботи з DOM та підключення різного роду бібліотек, яких написано багато як до фронтенду, так і для бекенду.

JavaScript може бути використаним не тільки як мова для клієнтської частини, але також як мова для створення серверу. Такий JavaScript називається Node.js.

Мова JavaScript при правильному та доречному використанні на клієнті допомагає зменшити навантаження на сервер, завдяки обробці деяких логічних вузлів на стороні користувача. Цими вузлами можуть бути як валідація форм, так і форматування контенту у форматах JSON та XML.

Але потрібно бути обережним, щоб не навантажувати пристрій користувача занадто сильно. Надмірне використання об’ємних калькуляцій на стороні клієнта може негативно вплинути на швидкість відкриття сайту на слабких пристроях, зручність користування сайтом, і як наслідок, зменшення потоку користувачів.

Мова вважається простою для початківців, та має низький поріг входження, що дозволяє не дуже досвідченим розробниками приєднатись до великого ком’юніті JavaScript програмістів.

Безліч бібліотек та фреймворків надають можливість використати готове рішення для проблеми або такий варіант вирішення, що легко налаштовується.

Ті речі, для яких неможливо створити бібліотеку дуже чітко та зрозуміло описані у дизайн шаблонах (patterns).

Усе це створює бездоганну екосистему програмування, яка робить розробку додатків легше та якісніше.

Найпопулярнішими фреймворками для frontend JS є:

- React;
- Angular та застаріла версія Angular.js;
- Vue.js.

Для backend JS:

- Node.js;
- Express.js.

У серпні 2021, команда Stack Overflow – популярної системи питань і відповідей для програмістів, провела щорічне опитування серед розробників про мови програмування. У опитуванні взяло участь більше ніж 82 тисячі професійних програмістів, та ентузіастів розробників.

На питання про найпопулярніший веб фреймворк відповіли 67,593 учасника, з яких 49,941 професійних програмістів [5]. Результати наведені на рис. 2.1.

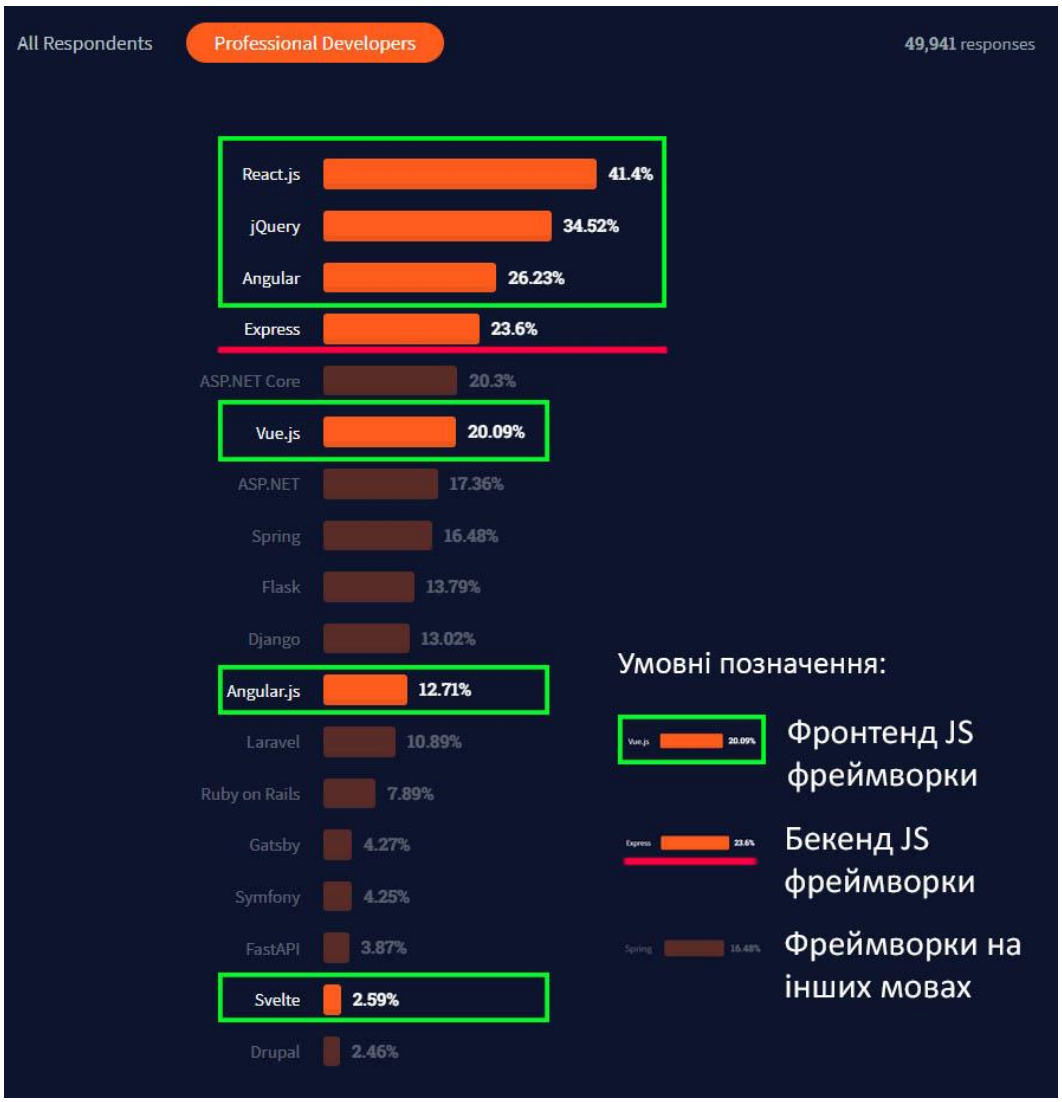


Рис. 2.1. Результати опитування програмістів

Якщо виділити тільки фреймворки, що відносяться до JavaScript, то найпопулярнішим є React, за яким йде jQuery. Це два лідери рейтингу JS фреймворків, що довго на висоті. jQuery існує з 2006 року, та був дуже популярним з початку існування і до сьогодні. За популярністю, React та jQuery зрівнялися у середині 2018. [6]

Наступними йдуть Angular, Express, та Vue.js. Наявність бекенд фреймворку у списку на такій високій позиції дуже гарно демонструє, що JavaScript широко використовується як серверна мова розробки.

Angular.js, що є старою версією сучасного Angular знаходиться посередині списку, а на передостанньому місці досить новий, та ще не дуже знайомий в масах фреймворк Svelte. Реакції розробників показують, що цей фреймворк дуже

сподобався респондентам, що його використовували, тож він має шанси стати популярним [7].

### 2.3.2. Опис бібліотеки React

Найпопулярніший на сьогодні фреймворк React – це JavaScript бібліотека, яка була створена Facebook (нині Meta), для використання в однойменній соціальній мережі.

Бібліотека дозволяє змінювати контент на сторінці, без необхідності перезавантаження. Це дуже зручно у контексті використання соціальної мережі, яка має складний інтерфейс та багато елементів взаємодії. Якби при кожному переході по соціальної мережі, відкриттю профіля інших користувачів або написання коментаря система перезавантажувала сторінку, такою системою неможливо було б користуватися.

Принцип роботи React, полягає у тому, що при завантаженні сторінки користувач отримує весь JS код додатку, та не довантажує його при взаємодії з застосунком.

Натомість, коли користувач проводить дії з сайтом, такі як відправка форми, повідомлення, зміна кольору теми, тощо, додаток отримує тільки дані у JSON форматі, а не цілком нову сторінку (рис. 2.2). При написанні програми, розробник створює шаблон, куди потім підставляться дані з серверу. Ці дані підставляються у шаблони, та відображаються однаково для усіх даних, без необхідності створення окремої сторінки для кожного запису в базі даних.

Це логічно, бо дані в базі даних для прикладу мають вагу 1кб, а згенерована або навіть написана вручну сторінка з тими ж даними може важити 1мб, або більше. Тобто сервер відправляє мінімум даних, а клієнт його споживає.

Такий підхід пришвидшує роботу серверу, через зняття навантаження, за причиною описаною вище. Також це пришвидшує роботу додатку на пристрої користувача і економить йому інтернет трафік.

Саме тому перехід між екранами у SPA додатках такий швидкий і непомітний [8].

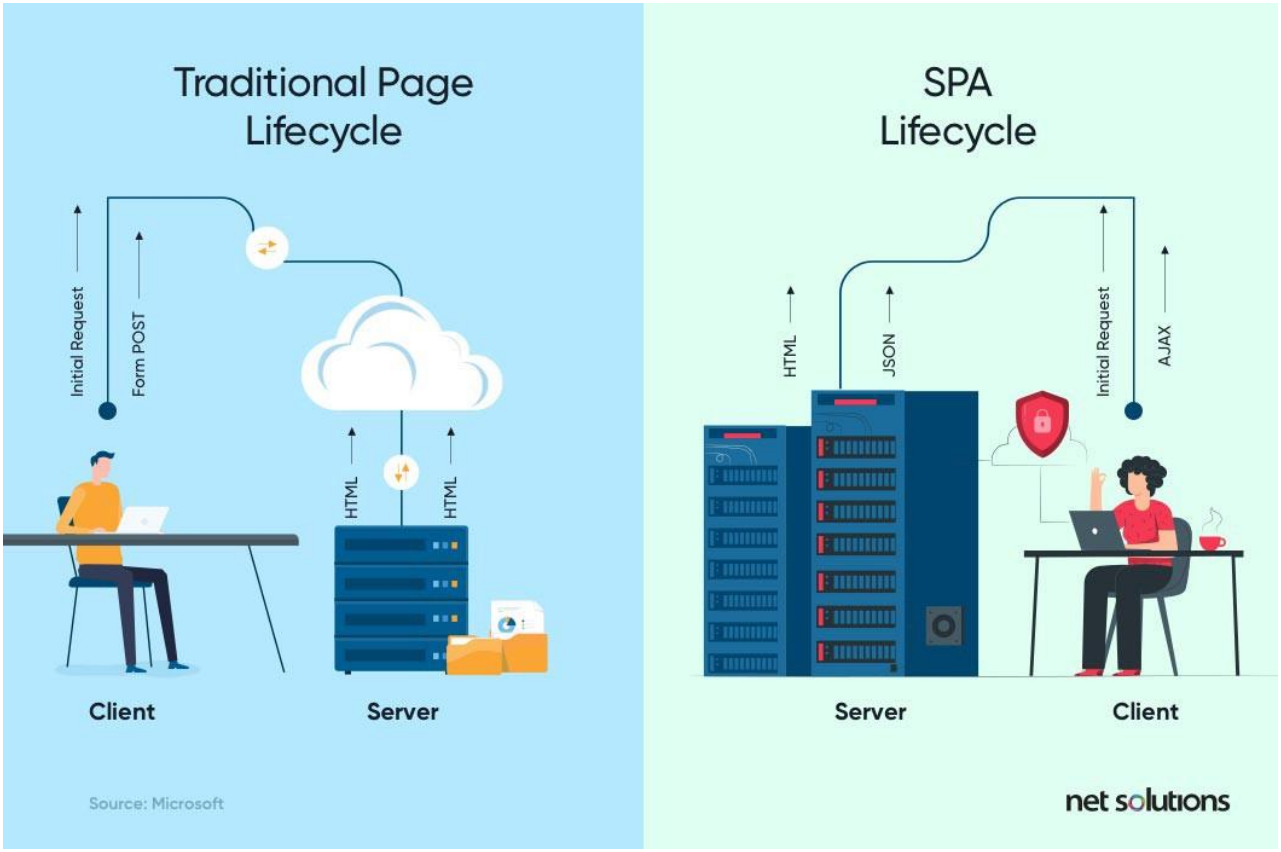


Рис. 2.2. Порівняння принципів роботи багатосторінкового сайту з SPA додатком

Великим плюсом для бібліотеки React, що також підвищує її популярність є універсальність. Після опанування React для створення веб додатків у браузері, можна писати мобільні додатки за допомогою React Native – офіційної бібліотеки для створення мобільних застосунків.

Усі великі SPA фреймворки, такі як React, Vue, Angular засновані з використанням компонентного підходу. Це спрощує процес розробки, через зменшення повторюваного коду, фрагментацію, та структуризацію файлової системи проекту.

Компонентний підхід розкривається у поділенні цілого блоку інтерфейсу на частини – незалежні одна від одної компоненти, що утворюють цей блок.

Наприклад необхідно зробити модальне вікно з можливістю редагування профілю користувача. У цьому вікні є основна частина – власне вікно, у якому знаходиться інший контент. Скоріш за все, за правилами дизайну, знизу будуть знаходитись кнопки для підтвердження зберігання оновлених даних, та відміни. Десь посередині знаходиться блок полів з особистими даними, такі як ім’я, прізвище, електронна пошта, тощо. І зверху знаходиться заставка профілю, яку також можна редагувати.

Якщо використовувати компонентний підхід, одразу видно, що буде компонентами та як вони будуть пов’язані.

Модальне вікно – це батьківська компонента, яка містить у собі дочірні.

Ці компоненти – блок редагування картинки, блок полів з даними, блок кнопок управління вікном. При чому, кожна з цих компонент може містити підкомпоненти, наприклад оригінально стилізоване поле для імені користувача, що має своє API з доданими методами.

Кожна з компонент має свій інкапсульований стан, що знаходиться окремо від DOM. Також компоненти можуть приймати дані з батьківських компонент, та повертати їх у зворотному боці.

Наприклад компонента модального вікна має прапор isTouched, що відповідає за логіку опрацьовування кнопки збереження. Тобто, якщо користувач зробив зміни в будь яких дочірніх компонентах, наприклад змінив ім’я, ця інформація повинна відобразитись на екрані, та дозволити користувачу натискати на кнопку збереження даних. Для цього прапорець, що відповідає за цей функціонал передається у дочірні компоненти, де може змінюватися в залежності від дій користувача, та передаватися назад, для опрацювання своєї логіки.

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно (diff) зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином бібліотека самостійно вирішує які компоненти сторінки треба оновити [10].

Оригінальною особливістю React, є використання JSX [11, 12]. JSX - JavaScript Syntax Extension, це розширення синтаксису JavaScript, яке виглядає звичним чином для багатьох розробників. Простота осягнення JSX є чинником низького порогу входу у фреймворк React. JSX схожий на HTML, але насправді це JavaScript (рис. 2.3).

```jsx
import React, { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Рис. 2.3. Приклад синтаксису JSX

React має велику екосистему, що включає у себе багато офіційних бібліотек:

- React – власне сама бібліотека, та похідна бібліотека, для швидкого створення та розгортання останньої офіційної збірки create-react-app;
- React Native – використовується для створення мобільних додатків, підтримує усі бібліотеки нижче;
- React Router – інструмент навігації між екранами;
- Redux – бібліотека управління станами додатку;
- Next.js – бібліотека для переходу від CSR до SSR, та похідна від неї бібліотека create-next-app, що створює додаток на React, який оптимізовано для рендерингу на сервері.

При виконанні кваліфікаційної роботи було надано перевагу саме React, а не Angular, або Vue.js, через наступні причини:

- Angular має надвелику вагу, через те, що усі компоненти фреймворку поєднані в одній бібліотеці, в той час як React або Vue.js це власне одна маловажна бібліотека та кілька додаткових, що загалом складають великий фреймворк. Якщо не потрібно використовувати бібліотеку для управління станам або маршрутизацією – від них можна відмовитись, просто не встановлюючи. Вагу готового проекту це зменшить в багато разів, відносно кількості бібліотек, що не були встановлені. Натомість тільки один Angular важить близько 20мб;
- Angular має досить високий поріг входу, та потребує знання TypeScript, що завищує вартість та складність проекту;
- Vue.js має досить невелике, порівняно з React або Angular суспільство користувачів, що зменшує шанси на пошук необхідного рішення при складній задачі, або помилці.

Саме тому ці фреймворки не розглядаються, як ті, які підходять для виконання поставленої задачі.

Системою контролю версій було обрано Git, як найпопулярнішу та найбільш використовувану.

Система управління репозиторієм – GitHub, що є безкоштовною системою для розміщення проектів з відкритими вихідними кодами.

## 2.4. Опис структури системи та алгоритмів її функціонування

### 2.4.1. Файлова структура системи

Проект знаходиться у головній папці «messenger», що містить 3 основних папки першого рівня (рис 2.4.):

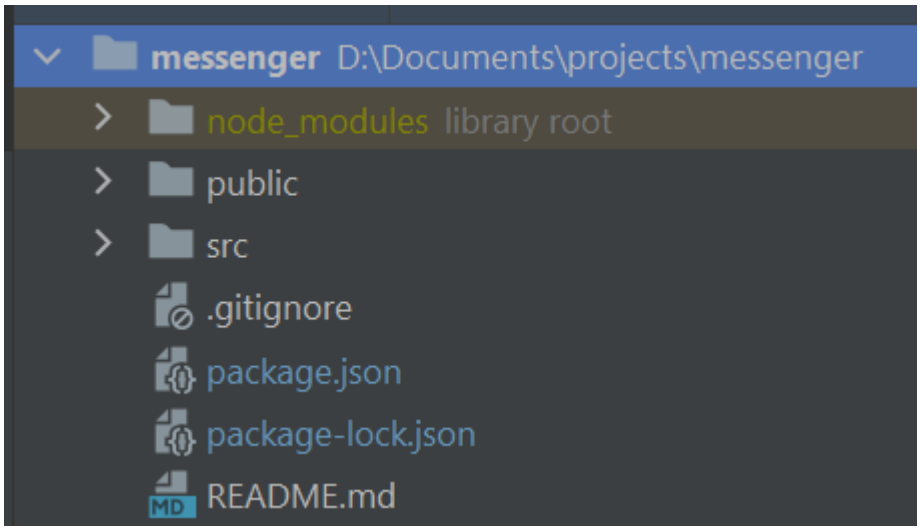


Рис. 2.4. Файлова структура кваліфікаційної роботи

– Каталог node_modules відповідає за встановлені залежності та бібліотеки, що використовуються у додатку явно (самим проектом), та неявно. Неявне використання бібліотек означає, що наприклад React використовує більше 200 сторонніх залежностей, що повинні бути встановлені для коректної роботи фреймворку. Папка node_modules створюється автоматично пакетною системою npm, що є частиною Node.js.

  Для того, щоб файлова система зрозуміла, які пакети необхідно встановити використовується конфігураційний файл package.json, що містить назви пакетів. Усі інші пакети, що мають встановлюватися для роботи, також мають такий файл. Таким чином забезпечується ланцюжок з пакетів та їх залежностей. Після встановлення пакетів, генерується файл package-lock.json, що містить логи встановлення модулів у каталог

модулів. Частину внутрішньої структури папки node_modules зазначено на рис. 2.5:

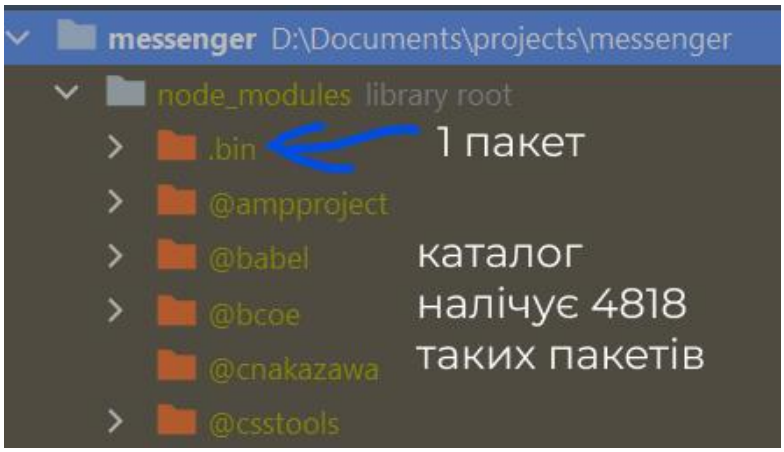


Рис. 2.5. Пакети в каталозі node_modules

– Папка public містить єдиний HTML файл, що є точкою входу у застосунок. Саме ця сторінка завантажується користувачеві при відкритті сайту, та саме в неї додаються згенеровані шаблони React компонент. Також ця папка містить іконку для сайту, що відображається у браузері, та автоматично створені файли конфігурації React (рис. 2.6.):

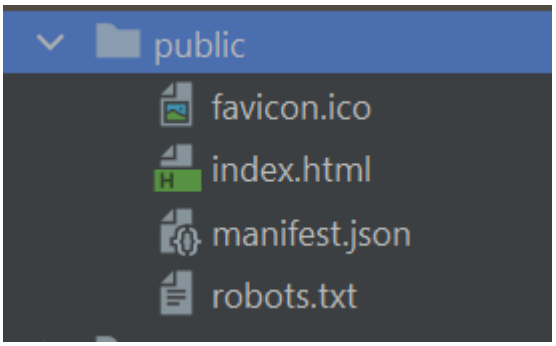


Рис. 2.6. Вміст папки public

– Папка src є головною складовою проекту, та містить у собі функціональну частину платформи (рис 2.7.).

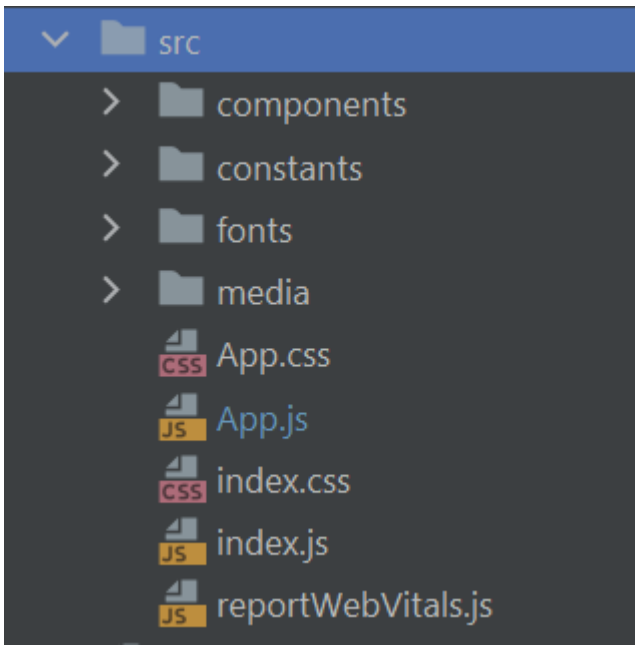


Рис. 2.7. Вміст папки src

У папці знаходяться папки другого рівня:

1. components, що вміщує усі React компоненти (рис. 2.8):

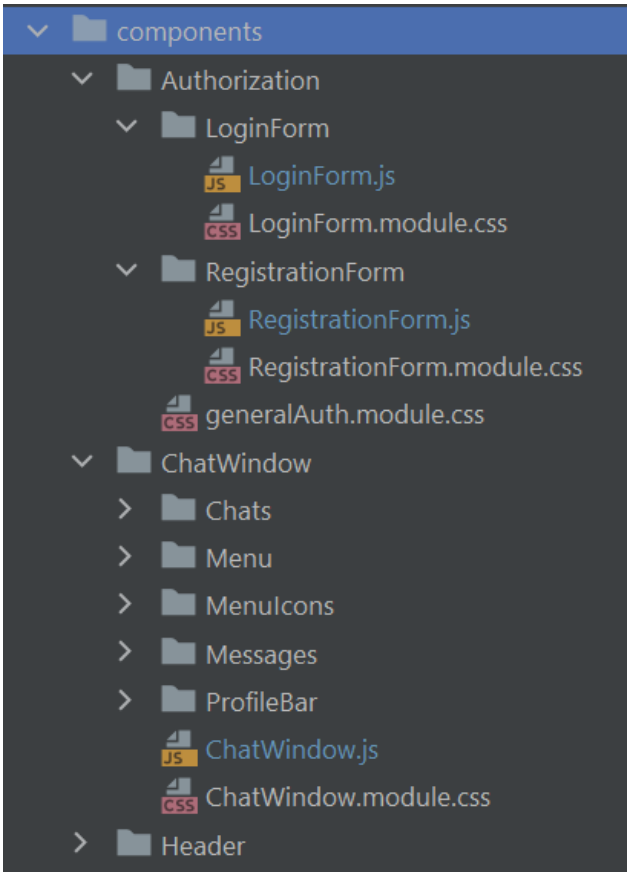


Рис. 2.8. Папка components та приклад вигляду компоненти React

Каталог компонент містить підкаталоги, що в свою чергу можуть містити підкаталоги ще нижчого рівня. У найнижчому рівні знаходиться файл компонент, це JavaScript файл у форматі .js, та модуль стилів для компоненти.

Перевага модульного підходу в форматуванні компоненту полягає у тому, що не обов'язково придумувати оригінальний css клас до кожного елементу, який потрібно стилізувати. Модульність розподіляє стилі таким чином, що кожна таблиця стилів, що відноситься до компоненти форматує тільки ті файли у які вона імпортована.

2. папку constants, у якій є константи, що використовуються додатком, а також деякі методи, що їх створюють (рис. 2.9):

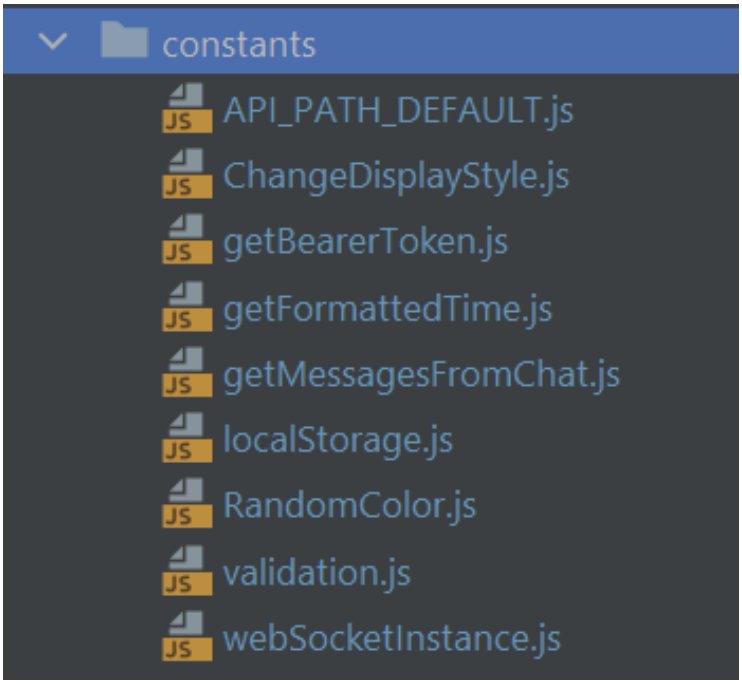


Рис. 2.9. Список констант у папці constants

3. fonts, яка містить шрифти, що використовуються у додатку, та згенерований файл stylesheet.css для підключення цих шрифтів (рис. 2.10):

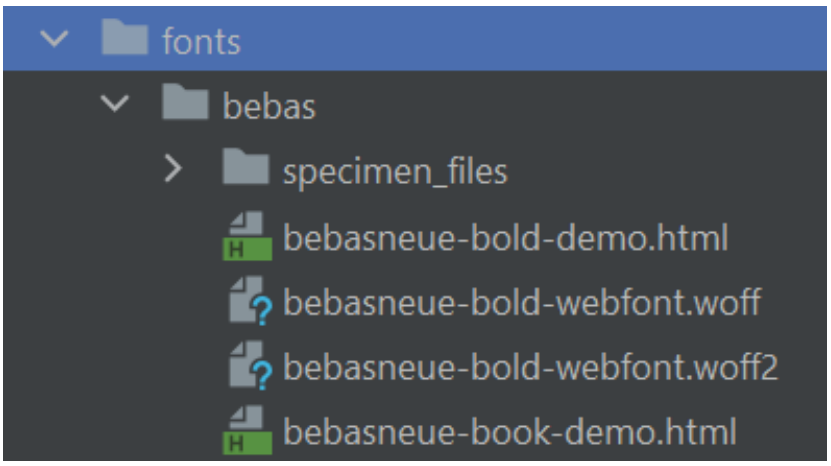


Рис. 2.10. Частковий вміст папки fonts

4. media, у якій знаходяться іконки та звукові сигнали, що подаються під час надходження повідомлення (рис. 2.11):

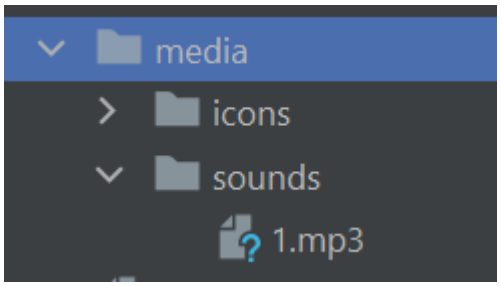


Рис. 2.11. Частковий вміст папки media

Також папка src містить допоміжні файли, що не відносяться до зазначених категорій:

- App.css – використовується для стилізування глобальних класів, що не пов’язані з окремими компонентами, окрім головної;
- App.js – головна компонента програми, яка вміщує у собі усі інші компоненти другого і нижче рівня, та маршрутизацію між екранами;
- index.css – що слугує для стилізації єдиного HTML файлу;
- index.js – автоматично згенерований файл з компонентом обгортки React, у який передається головна компонента App;
- reportWebVitals.js - автоматично згенерований файл, що допомагає відлагодженню додатку у процесі розробки;

### 2.4.2. Алгоритм функціонування сервісу обміну повідомленнями

Як було описано вище, при виконанні кваліфікаційної роботи використовується підхід SPA додатку, що не потребує перезавантаження сторінки для оновлення даних [8, 9].

Але месенджер, як система обміну повідомленнями, це у першу чергу відображення інформації у реальному часі. Тобто коли один користувач посилає повідомлення іншому, що чекає на нього з відкритим вікном чату, воно повинне з'явитися на екрані одразу, без перезавантаження сторінки. Так, технології сучасних SPA фреймворків це дозволяють, але тільки у випадку, якщо клієнт має необхідні дані.

Щоб у клієнта були дані, він повинен отримати їх з серверу. Щоб отримати дані з серверу, потрібно зробити запит на сервер з спеціальною командою. Такі запити можуть робитися при завантаженні сторінки, або при іншій взаємодії з сайтом. Наприклад, якщо юзер натискає на спеціальну кнопку «Fetch» (Обновити). Це застарілий підхід, що використовувався в інтернет форумах у 2000х роках. Якщо використовувати такий підхід, це нівелює усі плюси та переваги SPA фреймворку, на якому створено додаток, та власне самої бібліотеки, тому що бібліотеки створюються щоб вирішувати якісь проблеми. При поєднанні старого підходу з ручним оновленням даних і сучасного SPA додатку, на виході отримується додаток, який нікому не потрібен через свій застарілий та неактуальний функціонал.

Для вирішення цієї проблеми, на заміну REST, було винайдено протокол WebSocket, або скорочено WS.

REST – це архітектурний стиль програмування, що використовує принципи функціонування інтернету, та зокрема методи HTTP.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від

мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи і дозволяє їй еволюціонувати з новими вимогами [13].

Основними принципами REST архітектури є:

- Клієнт-серверна архітектура, від якої успадковується REST. Центральний вузол, який обслуговує інші вузли в системі, є вузлом сервера, а всі інші вузли є вузлами клієнта. Ця архітектура вимагає розділення відповідальності (separation of concerns), де компоненти, що зберігають і оновлюють дані (сервер), строго відокремлені від компонент, що відображають ці дані користувачеві. Це дозволяє незалежно розробляти ці частини системи;
- Відсутність стану, диктує, що кожен запит клієнту на сервер, повинен містити всю інформацію, що необхідна для розуміння та виконання сервером запиту. Тобто сервер не може використовувати інформацію з попередніх запитів. Через це, клієнт повинен утримувати стан сесії [14];
- Кешування. Системи, що написані у цьому стилі, повинні підтримувати кешування, тобто дані , які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність, уникаючи зайвих запитів, але також зменшує надійність системи через те, що дані в кеші можуть застаріти [15].
- Однорідний інтерфейс. Використання принципу однорідності до компонентів, можна суттєво спростити загальну архітектуру системи та покращити візуалізацію взаємодій [14].

WebSocket — це протокол, що призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Він забезпечує двонаправлений канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веб-браузерах та веб-серверах, але може також використовуватись будь-яким клієнт-серверним застосунком.

Доцільно використовувати протокол за необхідності відображення інформації в real-time [16].

Основні особливості WebSocket наступні:

- Протокол веб-сокета стандартизований [17], що означає, що за допомогою цього протоколу можливий зв'язок між веб-серверами і клієнтами в режимі реального часу.
- Веб-сокети трансформуються в кроссплатформовий стандарт для обміну даними між клієнтом і сервером [18].
- Найбільша перевага – це двосторонній зв'язок (повний дуплекс) по одному TCP-з'єднанню [19].
- WS - це незалежний протокол на основі TCP, що призначений для підтримки будь-якого іншого протоколу, який традиційно працює тільки поверх чистого TCP-з'єднання.
- WS – працює на транспортному рівні, поверх якого може працювати будь-який інший протокол. WebSocket API підтримує можливість визначення суб-протоколів: бібліотек протоколів, які можуть інтерпретувати певні протоколи.
- Єдина вимога від браузера - це запуск бібліотеки JavaScript, яка може інтерпретувати WebSocket, встановлювати і підтримувати з'єднання. У сучасних браузерах та версіях після 2015 року, підтримка протоколу є вбудованою, та не потребує додаткових бібліотек [20].

WebSocket - це найбільш кардинальне розширення протоколу HTTP з моменту його появи. Даний протокол змінює початково синхронний протокол, побудований за моделлю «запит-відповідь» на повністю асинхронний і симетричний. Для передачі даних ролі змінилися з клієнта і сервера з фіксованими ролями на два рівноправних учасника обміну даними. Кожен працює сам по собі, і коли виникає потреба відправляє дані іншому. Найкращим вибором для створення месенджера є саме протокол WebSocket, адже виникає потреба у динамічному оновленні даних, наявності постійно відкритого з’єднання клієнтів з сервером для підтримання комунікації, також можливість

використання інших протоколів поверх транспортного рівня. Для виконання обміну даними у кваліфікаційній роботі зі створення інтерактивного месенджеру було прийнято рішення вибрати WebSocket [21].

Нижче наведено відмінності між протоколами HTTP і WS:

– WS заснований на протоколі TCP і працює на транспортному рівні. HTTP працює на прикладному рівні;
– WS забезпечує двосторонній (full-duplex) зв'язок поверх одного TCP з'єднання, в той час як HTTP передбачає тільки односторонній (half-duplex) зв'язок. Двосторонній клієнт-серверний зв'язок означає, що сервер може повертати дані клієнта, чого не дозволяє HTTP;
– Коли клієнт відправляє HTTP запит на сервер, відкривається з'єднання TCP між клієнтом і сервером і після отримання відповіді TCP з'єднання закривається. Кожен HTTP запит відкриває окреме TCP з'єднання, наприклад, якщо клієнт відправить 10 запитів, відкриється 10 з'єднань і закриються після отримання відповіді або у результаті відмови відповіді серверу (fallback), коли TCP порти сервера не доступні клієнту (рис. 2.12).

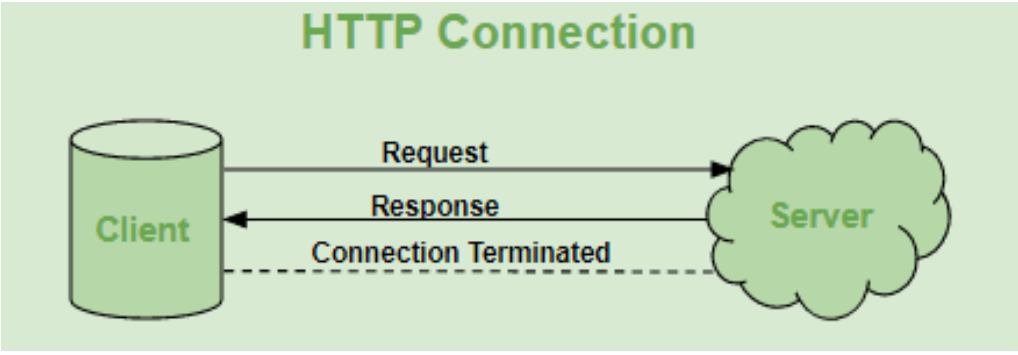


Рис. 2.12. Принцип роботи зв’язку через протокол HTTP

WebSocket являє собою «stateful protocol», що означає, що з'єднання між клієнтом і сервером буде залишатися активним до тих пір, поки воно не буде розірвано одною зі сторін (клієнтом або сервером). Після закриття з'єднання однією стороною, воно припиняється для обох сторін (рис. 2.13).


Рис. 2.13. Принцип роботи зв’язку через протокол WS

Для виконання поставленої задачі кваліфікаційної роботи було використано обидва протоколи.

Протокол HTTP використовується для усіх запитів на сервер, окрім власне листування. Сервер приймає запити за декількома точками доступу, що виконують такі функції як:

- реєстрація нового користувача;
- логін існуючого користувача;
- дії з книгою контактів (додавання та видалення користувача, відображення усього списку);
- дії з чатами (повернути, видалити);
- дії над користувачем (отримати профіль, редагувати профіль, редагувати пароль, шукати користувача)
- дії з повідомленнями (редагувати, видалити повідомлення, отримати останнє повідомлення, отримати частину повідомлень у чаті).

Так як протокол WebSocket розуміє тільки передачу даних, що відбувається просто зараз, необхідно отримати список повідомлень відправлених раніше. Саме для цього і створено точку запиту, яка повертає останні 200 повідомлень.

Протокол WS підключається одразу після завантаження сторінки з вікном чату. Тобто, як тільки юзер зміг авторизуватися у системі та потрапив на основну сторінку месенджера, встановлюється зв’язок WebSocket, що прослуховує сервер на наявність нових повідомлень. Коли з’являється нове повідомлення,

клієнт відображає його в інтерфейсі. Таким чином користувач може отримувати одночасно багато повідомлень з різних чатів, та встановлений зв’язок оброблює їх усі.

## 2.5. Обґрунтування та організація вхідних та вихідних даних програми

Під час проектування системи, було проаналізовано вхідні та вихідні дані, призначення додатку, користувачів, та прораховано спрощення інтерфейсу для взаємодії користувача та системи.

Вхідними даними є ті, які вводить користувач у формах, та текстових полях, які згодом записуються у базу даних.

Вихідними даними є ті, що витягуються з бази даних для відтворення правильного вигляду сторінки.

Вхідні дані отримуються під час:

- заповнення форми реєстрації, логіну;
- редагування профілю;
- листування у чатах;
- редагування/видалення повідомлень;
- додавання/видалення контактів.

Вихідні дані відображаються як:

- профіль користувача;
- профілі інших користувачів;
- завантажені повідомлення у чатах;
- список контактів.

Дані у додатку знаходяться на рівні компонент, тобто є головна компонента, яка отримує необхідну їй частину даних, а потім розсилає далі, йдучи по дереву компонентів від головної, до інших вниз. Деякі компоненти отримують дані самостійно, без участі головної компоненти, та можуть розподіляти їх у собі, або передавати вниз, до дочірніх компонентів.

Нижче наведено моделі даних у форматі JSON, що відправляються на сервер, та отримання відповідей на них, на прикладі трьох частин додатку – реєстрації, контактів, повідомлень.

1. Реєстрація користувача (sign-up). Це POST запит, на який клієнт відправляє наступні дані:

```
{
 "username": "string",
 "firstname": "string",
 "lastname": "string",
 "email": "string",
 "password": "string",
 "phoneNumber": "string"
}
```

Це ім’я, прізвище, ім’я користувача, електронна адреса, пароль, та номер телефону. У відповідь сервер надсилає код 200 OK [22].

Треба зауважити, що не всі поля є обов’язковими. Так, поля для прізвища та номеру телефону можна пропустити (рис. 2.14).


Рис 2.14. Схема даних для створення користувача

2. Для контактів є 4 точки запиту:
   - додати у контакти;
   - видалити з контактів;
   - повернути список контактів;
   - перевірити, чи є людина у контактах;

Якщо відправити на захищену точку запиту /api/v1/contacts/{id} де id це унікальний ідентифікатор користувача в БД у POST запиті – цей користувач буде додано до контактів. При цьому сервер повертає код 201 Created [22].

Якщо на цю саме адресу відправити id користувача у DELETE запиті - користувача буде видалено з контактів. Сервер повертає код 202 Accepted [22].

Для того, щоб отримати частковий список контактів, потрібно відправити GET запит на /api/v1/contacts з тілом запиту

```
{
  "size": 200,
  "page": 0
}
```

Де size означає розмір сторінки з кількістю користувачів, від 10 до 500, а page означає номер сторінки. Якщо у користувача більше ніж 200 контактів, ті, що йдуть після номеру 200, не будуть попадати в першу сторінку, якщо її розмір 200.

Сервер повертає код 200 ОК [22] та масив об’єктів, що містять дані про контакт:

```
[
  {
    "username": "string",
    "email": "string",
    "firstname": "string",
    "lastname": "string",
    "id": 0
  }
]
```

Щоб дізнатися, чи є у користувача контакт з цим ідентифікатором, потрібно у GET запиті на /api/v1/contacts/present/{id}, відправити цей ідентифікатор. У відповідь сервер відправляє код 200 ОК, та значення

– true – якщо користувач є контактом

– false – у протилежному випадку.

3. Повідомлення мають 6 точок запиту, але для скорочення звіту наведено тільки 2.

Для редагування повідомлення потрібно відправити PUT запит на /api/v1/messages/{id}, де id – унікальний ідентифікатор повідомлення з тілом повідомлення:

```
{
  "text": "string"
}
```

Де text – новий текст повідомлення.

Для видалення повідомлення, потрібно на ту ж саму адресу відправити DELETE запит, без тіла. У обох випадках сервер повертає код 202 Accepted.

## 2.6. Опис роботи розробленої системи

### 2.6.1. Використані технічні засоби

Для серверних систем забезпечення функціонування додатку необхідні апаратні характеристики, не нижче наступних:

- операційна система Windows 10;
- оперативна пам’ять 8 ГБ;
- процесор AMD Ryzen 7 5800X 3.8 GHz / 32 MB;
- пам’ять відеокарти ніж 64 МБ;
- швидкість інтернет зв’язку 200 мб/с;
- SSD M.2 об’ємом 1 ТБ.

При наявності серверного обладнання, що має характеристики не нижче наведених, можливо розгорнути клієнтську частину соціального месенджеру.

Мається на увазі сервер, на якому знаходиться клієнтська частина, а не сервер, який обробляє запити. Але характеристики апаратної складової серверу, що опрацьовує запити відрізняється тільки пам’яттю жорсткого диску, та кількістю таких готових серверів.

Для масштабування додатку, при зростанні популярності та кількості користувачів, необхідно збільшувати кількість серверів, що обробляють запити.

За потреби їх можна встановлювати у різних географічних локаціях, для покращення маршрутизації для користувачів з різних країн.

Детальні специфікації для пристроїв користувачів наведено у підпункті 1.5.3, першого розділу цієї роботи.

### 2.6.2. Використані програмні засоби

Проект створений на основі мови програмування JavaScript з використанням SPA бібліотеки React.

Також використовувалися бібліотеки:

- axios – для створення HTTP запитів на сервер;

– socksJS – імплементація протоколу WebSocket для ранніх версій JavaScript, що допомагає забезпечити підтримку протоколу навіть для старих браузерів;
– stompjs – бібліотека для обгортки протоколу WS протоколом STOMP;
– seedrandom – бібліотека генерації випадкових чисел, що використовуються для створення кольорів, що відображаються у профілі користувачів;
– use-sound – бібліотека-доповнення для React, що дозволяє використовувати звукові ефекти у React додатку;

У якості способу стилізування використовувалися каскадні таблиці стилів (CSS), з модульним підходом, властивим для React застосунків.

Під час розробки була використана IDE WebStorm від компанії Jet Brains, що має вбудований вебсервер. Перевагу саме цьому інструменту розробки було надано через великий функціонал, який впроваджено у системі, та легкість його осягнення, а також те, що університет надає можливість безоплатно користуватися цим редактором завдяки університетській ліцензії.

### 2.6.3. Виклик та завантаження програми

Завантаження розробленої системи на сервер залежить від серверу та хостингової системи, які було обрано для утримання додатку.

Типова практика визначає 2 способи – розгортання сайту за допомогою FTP підключення, або автоматичне розгортання на сервісах з репозиторію, на якому знаходиться розроблений додаток.

Перший спосіб, FTP з’єднання з хостингом передбачає:

– орендування або придбання серверного обладнання;
– налаштування цього обладнання;
– приєднання до серверу за допомогою FTP клієнта, та ручне переміщення скомпільованих файлів програми;
– підключення DNS та SSL сертифікату.

Другий спосіб, розгортання з репозиторію – який було обрано наприкінці розробки передбачає:

- обрати сервіс, що займається хостингом, для цього було обрано Heroku;
- обрати контент план з необхідною потужністю обладнання;
- підключити акаунт та репозиторій GitHub, на якому знаходиться проект месенджеру;
- обрати гілку, з якої буде відбуватися розгортання;
- розгорнути проект за допомогою кнопки «Deploy»;
- за необхідністю поставити галочку у пункті автоматичного розгортання при зміні обраної гілки.

Для завантаження програми користувачу потрібно перейти за адресою, на якій розташовано месенджер [25]. Також можна відвідати розгорнуту серверну частину додатку для розуміння функціонуючого API [26].

### 2.6.4. Опис інтерфейсу користувача

Структуру додатку наведено у схемах, що відображають способи використання додатку, та частково показують елементи інтерфейсу.

1. Для того, щоб потрапити в додаток і почати листування від користувача потрібно 2 умови:
    a. користувач повинен бути зареєстрований
    b. вхід в обліковий запис проведено (логін)

    При виконанні цих умов, користувач отримує доступ до системи месенджеру (рис. 2.14).

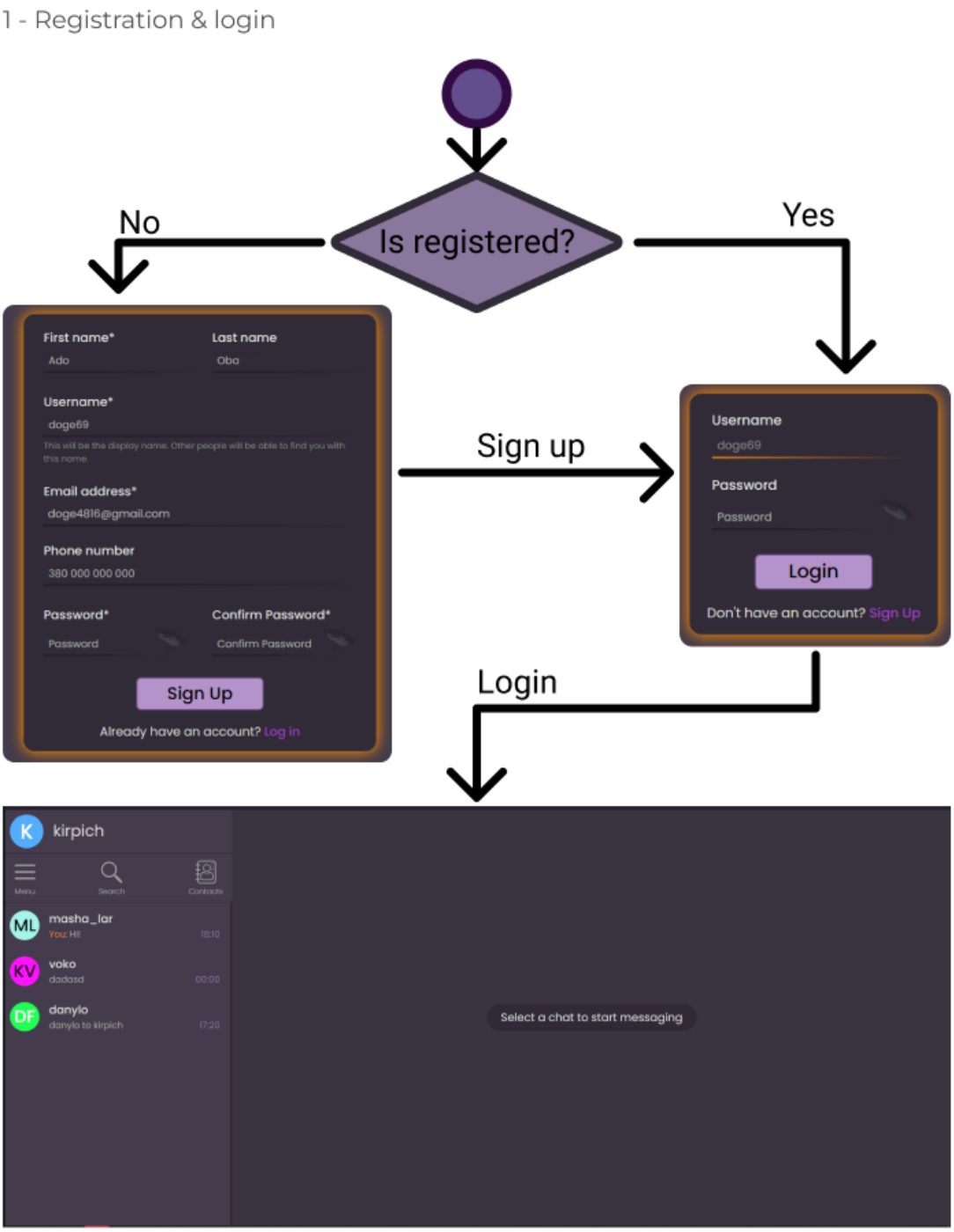


Рис. 2.15. Схема використання системи реєстрації та логіну

Якщо користувач ще не створив обліковий запис – йому потрібно це зробити. Він вводить особисті дані, необхідні для реєстрації: ім’я, прізвище, електронну адресу, телефон, пароль та повторення паролю.

Після цього користувач переходить на сторінку логіну, де вводить дані створеного аккаунту, та підключається до власне месенджеру.

2. Припустимо, що користувач, при вході у додаток, має один або більше діалогів з іншими юзерами. В такому випадку, він може відкрити чат, натиснувши по рядку з іменем користувача (рис. 2.15). Закрити чат можна натисканням клавіші «Escape», на клавіатурі. Мобільна версія не передбачає закриття чату.

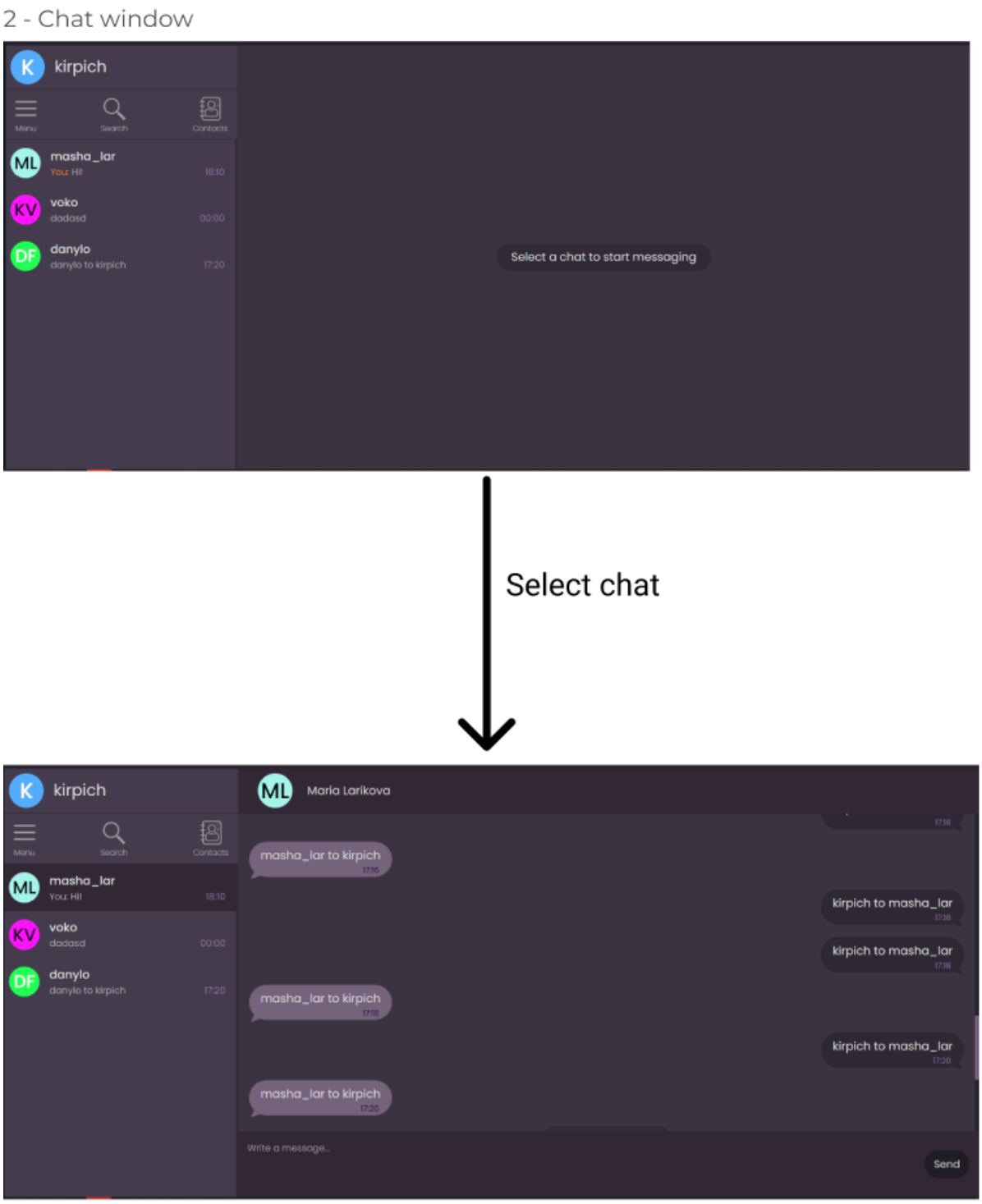


Рис. 2.16. Схема порядку дій для відкриття існуючого чату

3. Під час використання месенджером, користувач може відкривати профіль співрозмовника, натиснувши на іконку з ініціалами користувача, або його ім’я (рис. 2.16). У результаті відкриється модальне вікно, що показує базову інформацію про користувача, та дозволяє робити певні дії з ним. Можна додати користувача у список контактів, або забрати його звідти. Також можна видалити чат із співрозмовником. Ця кнопка видаляє чат для обох користувачів.

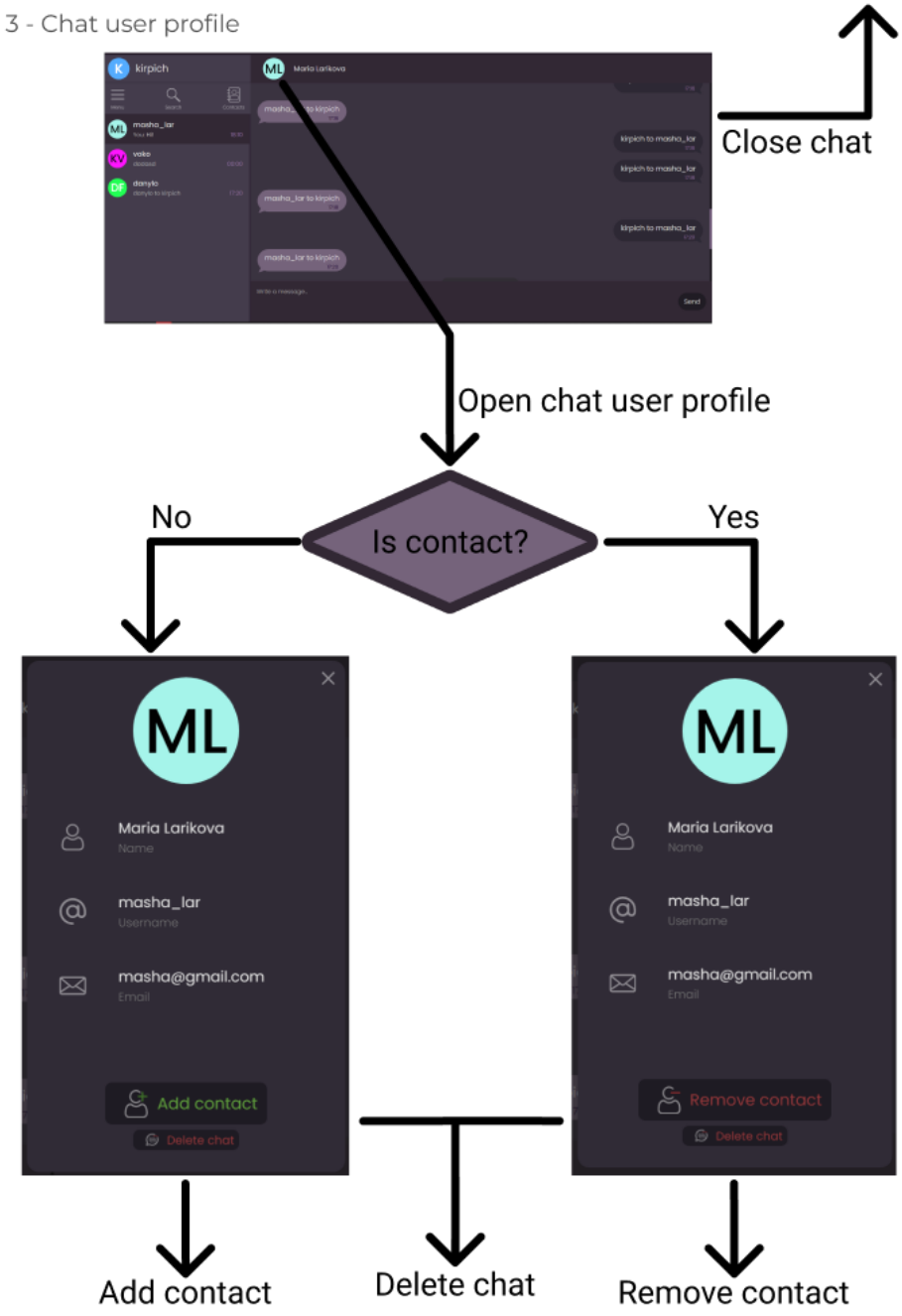


Рис. 2.17. Схема дій з профілем співбесідника

4. Користувач має можливість змінювати власний профіль, натиснувши на іконку зі своїми ініціалами або ім’ям у верхній лівій частині екрану, одразу під логотипом месенджеру. Відкриється модальне вікно, яке дозволяє редагувати ім’я, псевдонім, електронну адресу, номер телефону (рис. 2.17) та пароль (рис. 2.18). Для кожного з полів відкриється додаткове модальне вікно, де можна ввести нові дані.

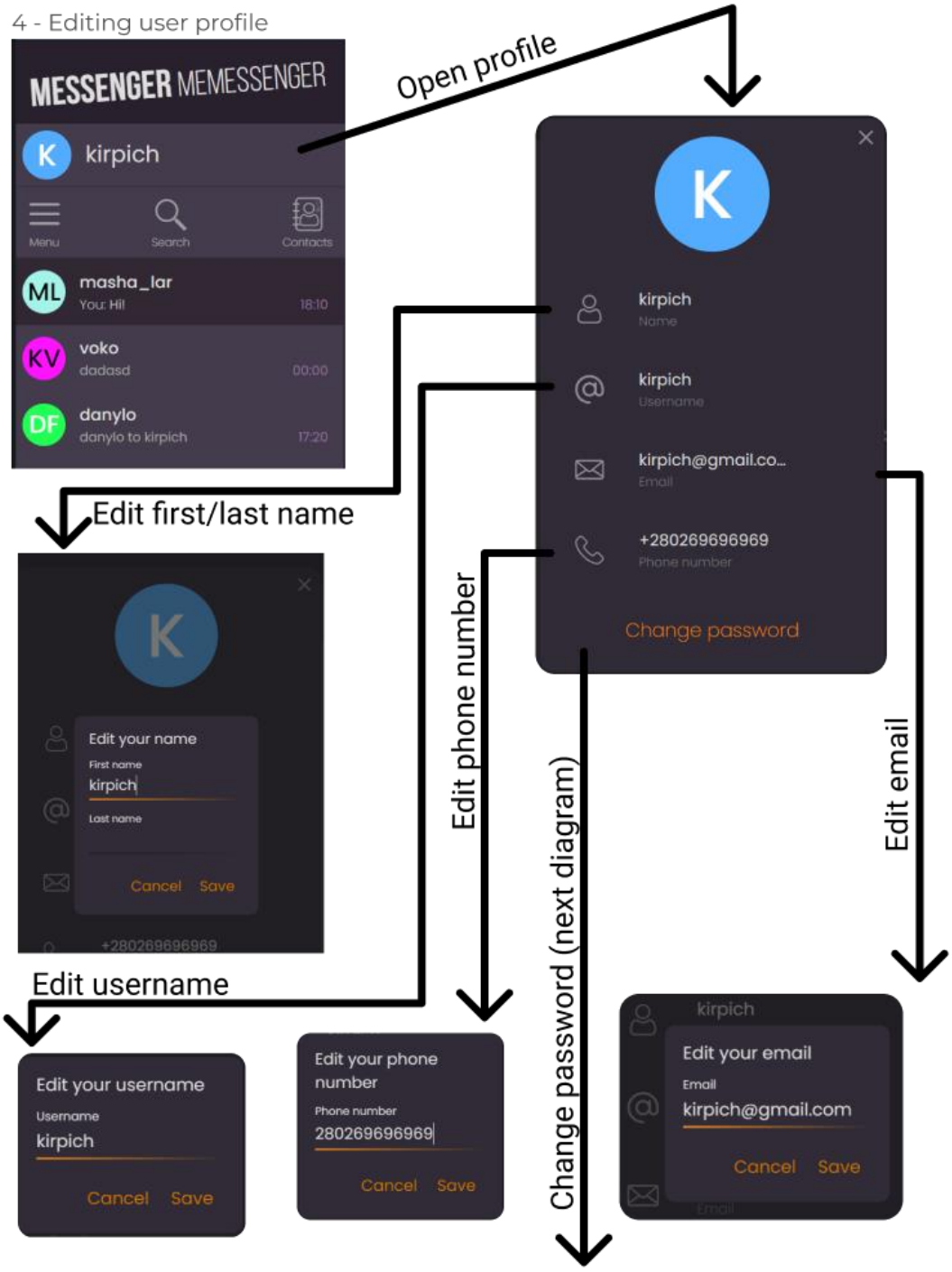


Рис. 2.18. Схема можливостей редагування основних полів профілю

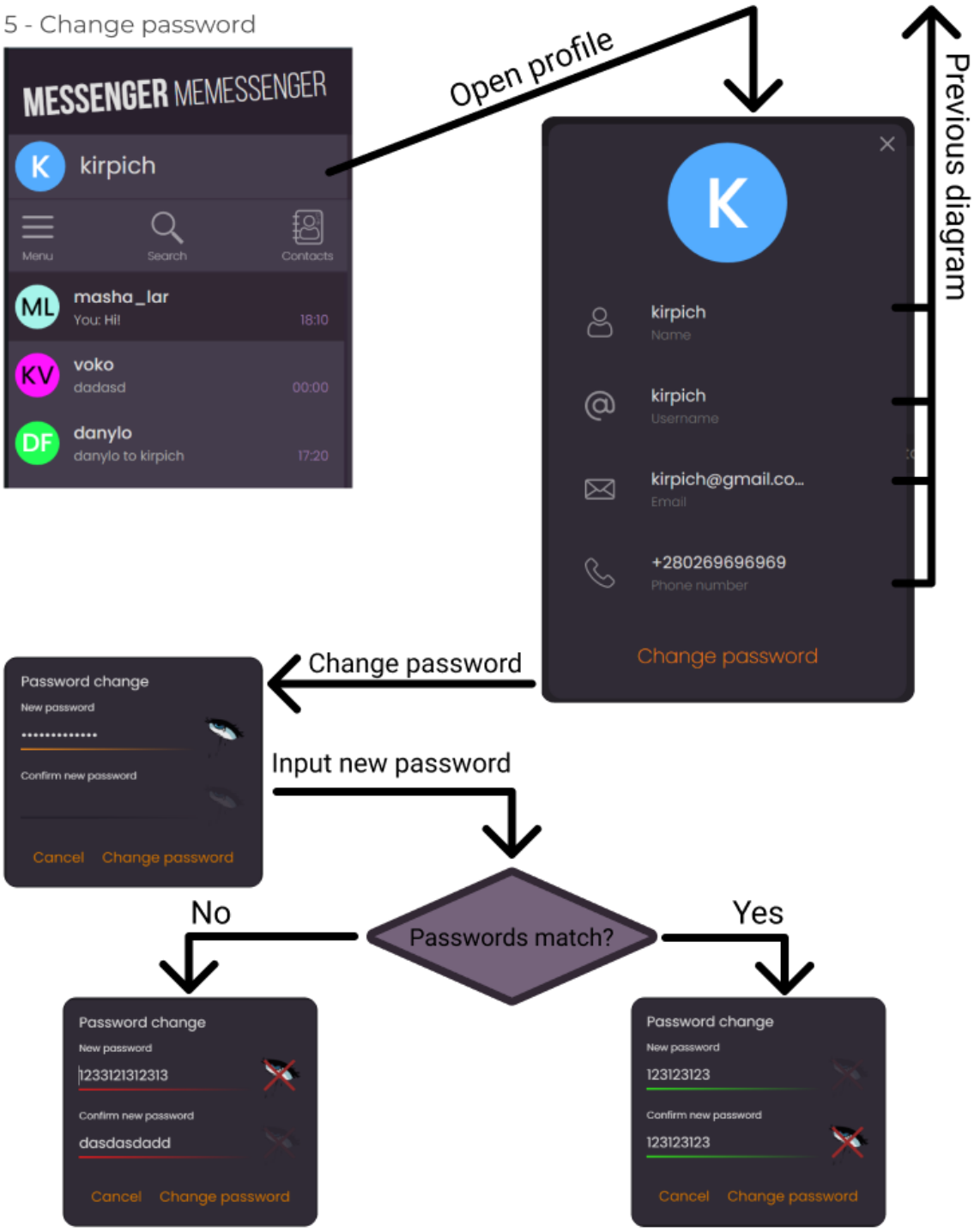


Рис. 2.19. Порядок дій користувача під час редагування паролю

5. Модальне вікно редагування паролю потребує ввести 2 поля. Новий пароль, та підтвердження нового паролю. Якщо введені паролі співпадають, поля підкреслюються зеленим кольором – користувач може натиснути на кнопку для збереження нового паролю. В іншому випадку,

коли паролі не співпадають, поле з паролем підкреслюється червоним, та натискання кнопки зміни паролю не працює.

6. Користувач має змогу шукати інших користувачів, для додавання їх у контакти, або для того, щоб написати їм повідомлення. Для відкриття віконця пошуку, потрібно натиснути на відповідну іконку, що знаходиться між профілем користувача, та списком чатів (рис. 2.19).

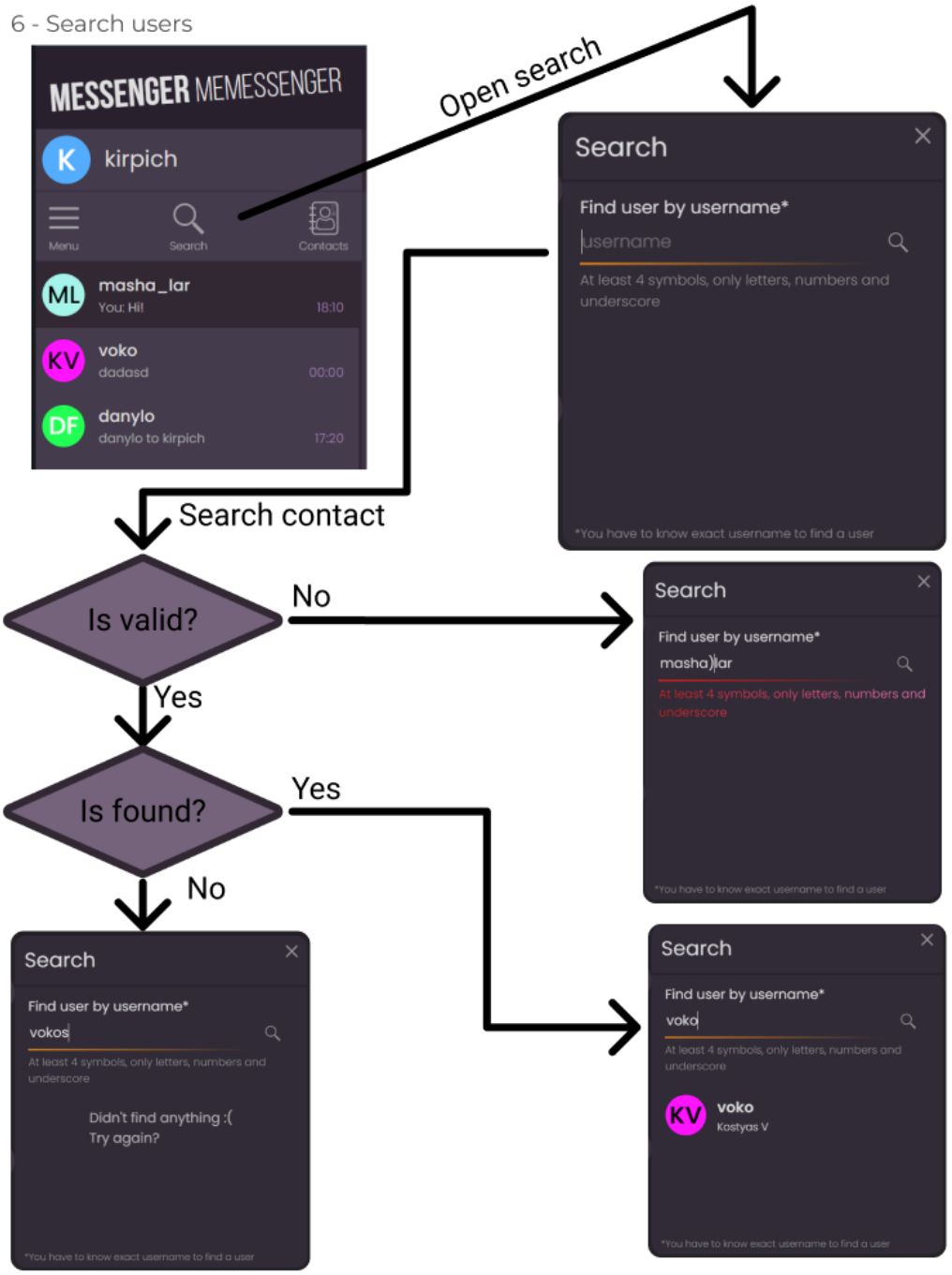


Рис. 2.20. Схема дій користувача для пошуку інших юзерів

Якщо введене значення у поле відповідає дозволеному, проходить пошук. В іншому випадку, юзер отримує повідомлення про схиблену валідацію імені користувача.

Якщо система знайшла користувача за його ім’ям, то цей користувач відображається у вікні пошуку.

7. З користувачем можна виконати спеціальні дії, відправити повідомлення або додати (видалити) до контактів (рис. 2.20)

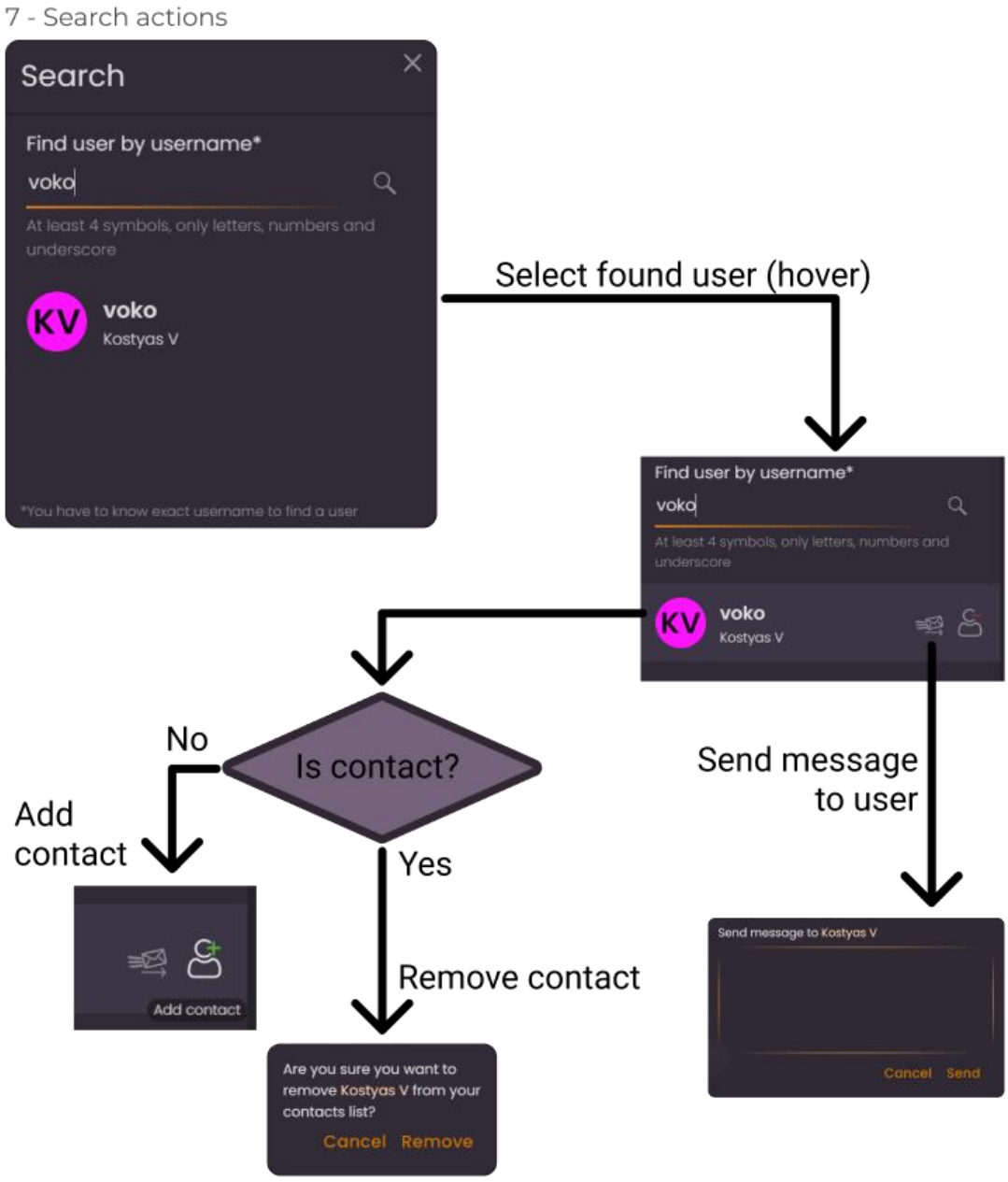


Рис. 2.21. Можливі дії над знайденим користувачем

Якщо навести на знайденого користувача, він підсвічується світлішим кольором, та з правої частини з’явиться 2 іконки:

- відправити повідомлення;
- додати / видалити до/з контактів.

При натисканні на кнопку для відправки повідомлення відкривається модальне вікно з текстовим полем, у якому можна написати повідомлення. Вікно можна закрити натиснувши на кнопку відміни, або на клавішу «Escape» на клавіатурі.

Якщо користувач, якого шукають не знаходиться у контактах користувача, що його шукає, іконка дій з контактом має вигляд людини з зеленим знаком «+» зверху. При натисканні на кнопку, користувач додається до книги контактів.

Якщо користувача, якого шукають вже є доданим контактом, іконка виглядає так само, але замість зеленого знаку «+» відображається червоний знак «–». При натисканні на кнопку, відкривається модальне віконце з підтвердженням видалення користувача із контактів.

8. Користувач має можливість заходити в книгу контактів, для перегляду списку своїх контактів, та проводити наступні дії над ними (рис. 2.21).:

- перегляд профілю користувача;
- видалення контакту;
- надсилання повідомлення користувачу

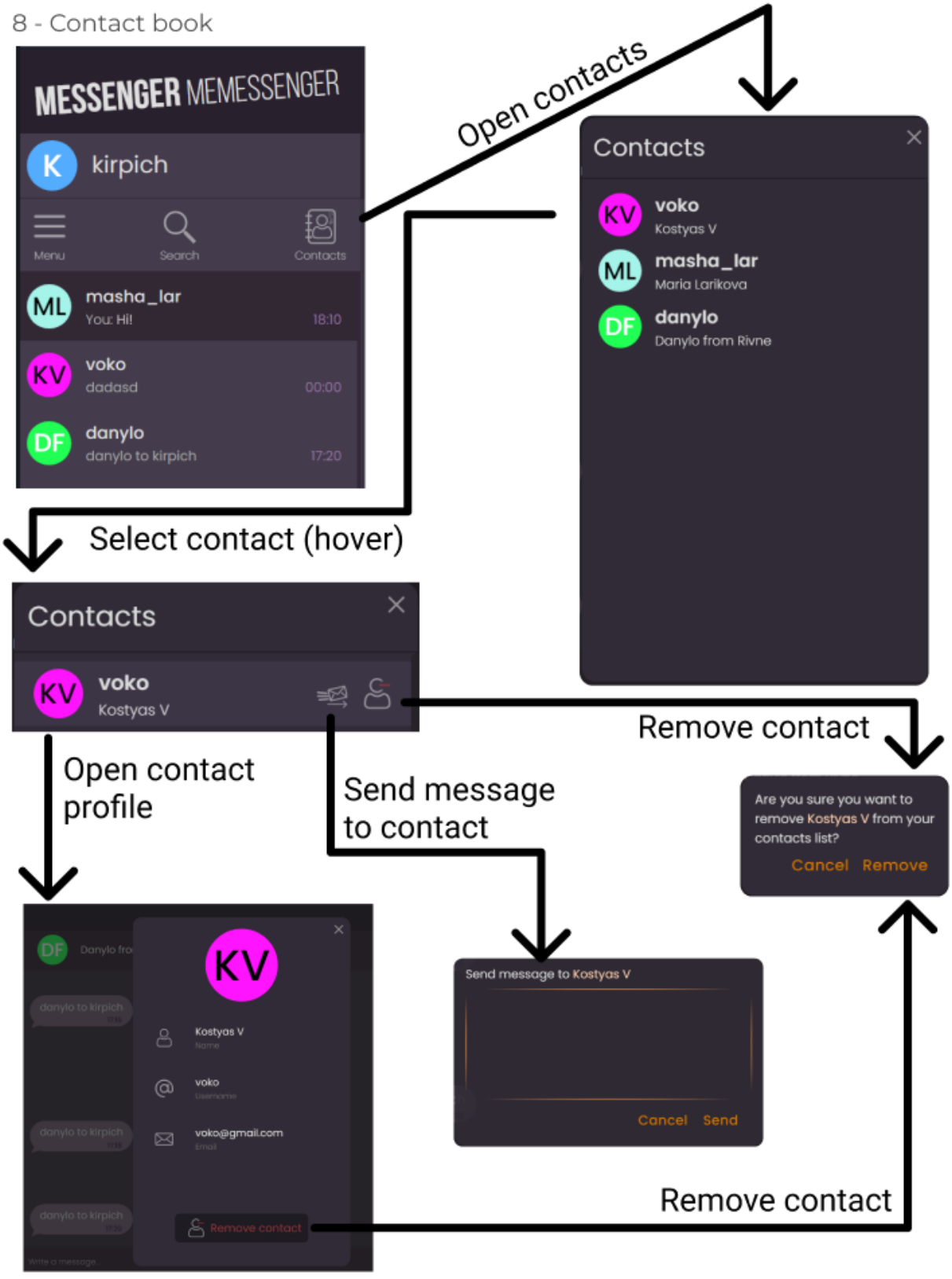


Рис. 2.22. Схема вигляду книги контактів та дій над ними

При відкритті списку контактів, користувач бачить усі контакти, що були додані до книги. Якщо відображається більше ніж вміщує екран, користувач прокручує список униз.

При наведені на користувача, відбувається теж саме, що і у вікні пошуку користувача. Також можна виконати такі самі дії, та додатково зайти у профіль користувача, звідки також можна видалити його з контактів.

9. Користувач може вийти з облікового запису на сторінку логіну, натиснувши на відповідну опцію в меню, що можна відкрити крайньою лівою функціональною кнопкою, що знаходиться між рядком профілю та списком чаті (рис 2.22).

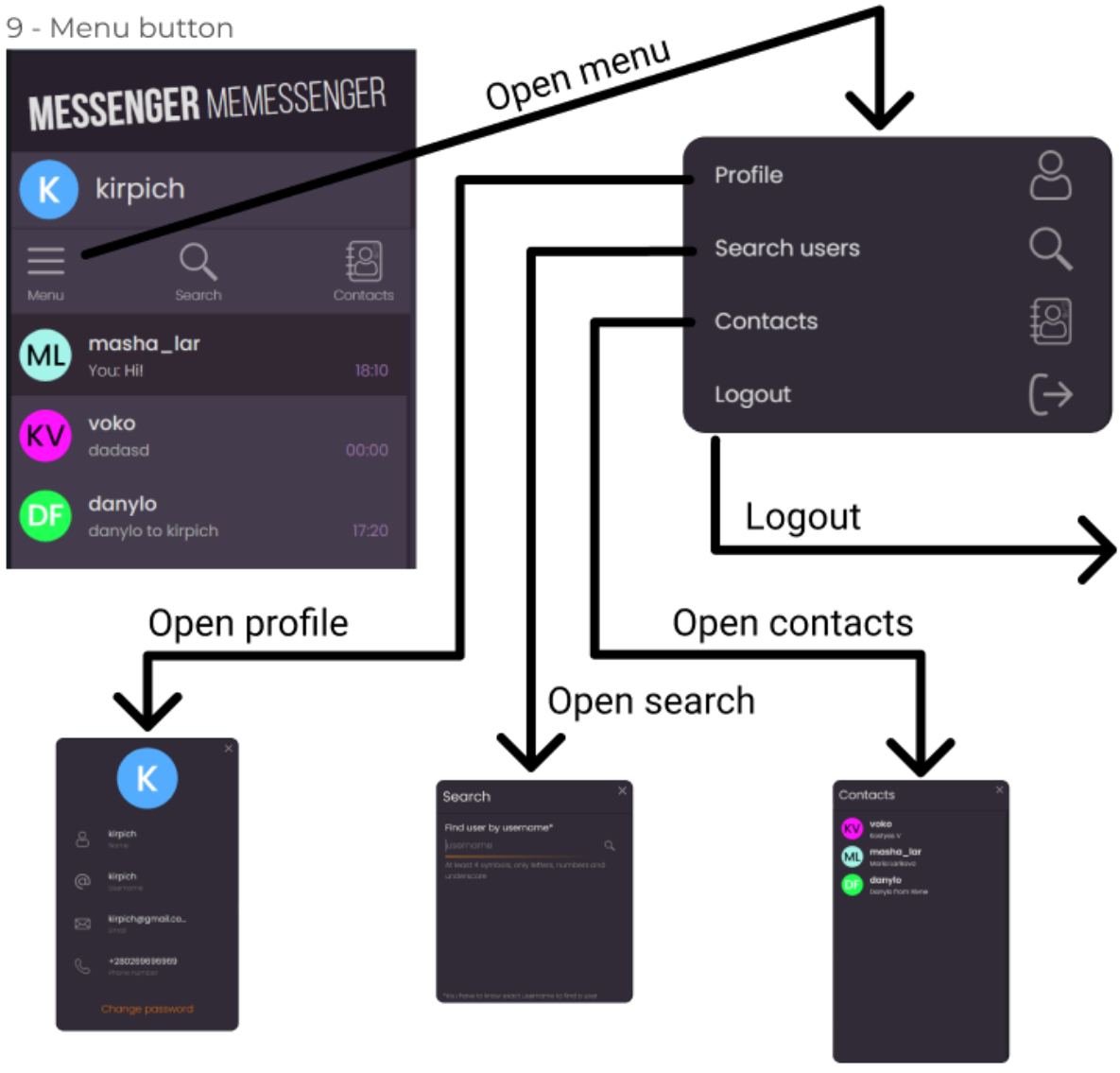


Рис. 2.23. Можливі дії для використання меню

Кнопка меню, при натисканні відкриває модальне вікно, у якому є можливість:

– відкрити профіль користувача;
– шукати інших користувачів;
– відкрити книгу контактів;
– вийти з облікового запису.

В мобільній версії додатку, ця кнопка присутня, та працює так само як і в версії для ПК.

10. Користувач може проводити дії над повідомленнями. Над своїми повідомленнями, тобто тими, що користувач відправив сам, він має більше прав, ніж над повідомленнями співрозмовника (рис. 2.23).

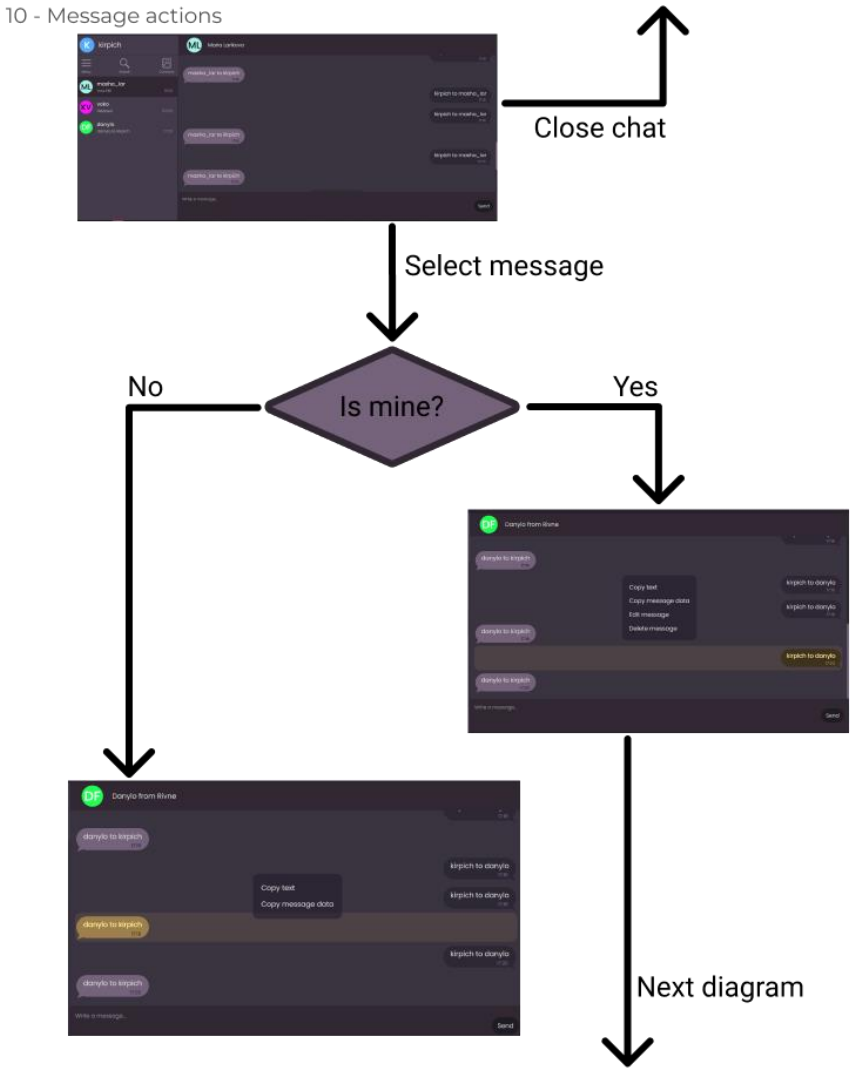


Рис. 2.24. Схема використання дій над повідомленнями

Для відкриття спливаючого віконця з повідомленнями, потрібно клацнути правою кнопкою миші по повідомленню. Воно виділиться жовтуватим кольором, та з’явиться вікно зі списком доступних дій.

Якщо повідомлення було відправлене користувачем, він може проводити над ним усі можливі дії, які включають:

- копіювання тексту повідомлення;
- копіювання даних повідомлення;
- редагування повідомлення;
- видалення повідомлення.

Якщо користувач виділив повідомлення, що було надіслане іншим користувачем, він не може його редагувати або видаляти, тільки копіювати текст та дані.

При натисканні на опцію редагування повідомлення, з’являється модальне вікно з текстовим полем, куди можна ввести новий текст повідомлення, та дві кнопки, що відповідають за збереження нового повідомлення, або відбій редагування.

При натисканні на опцію видалення повідомлення, з’являється модальне вікно з попередження про незворотність цієї дії та того, що повідомлення видаляється для обох користувачів (рис. 2.24).

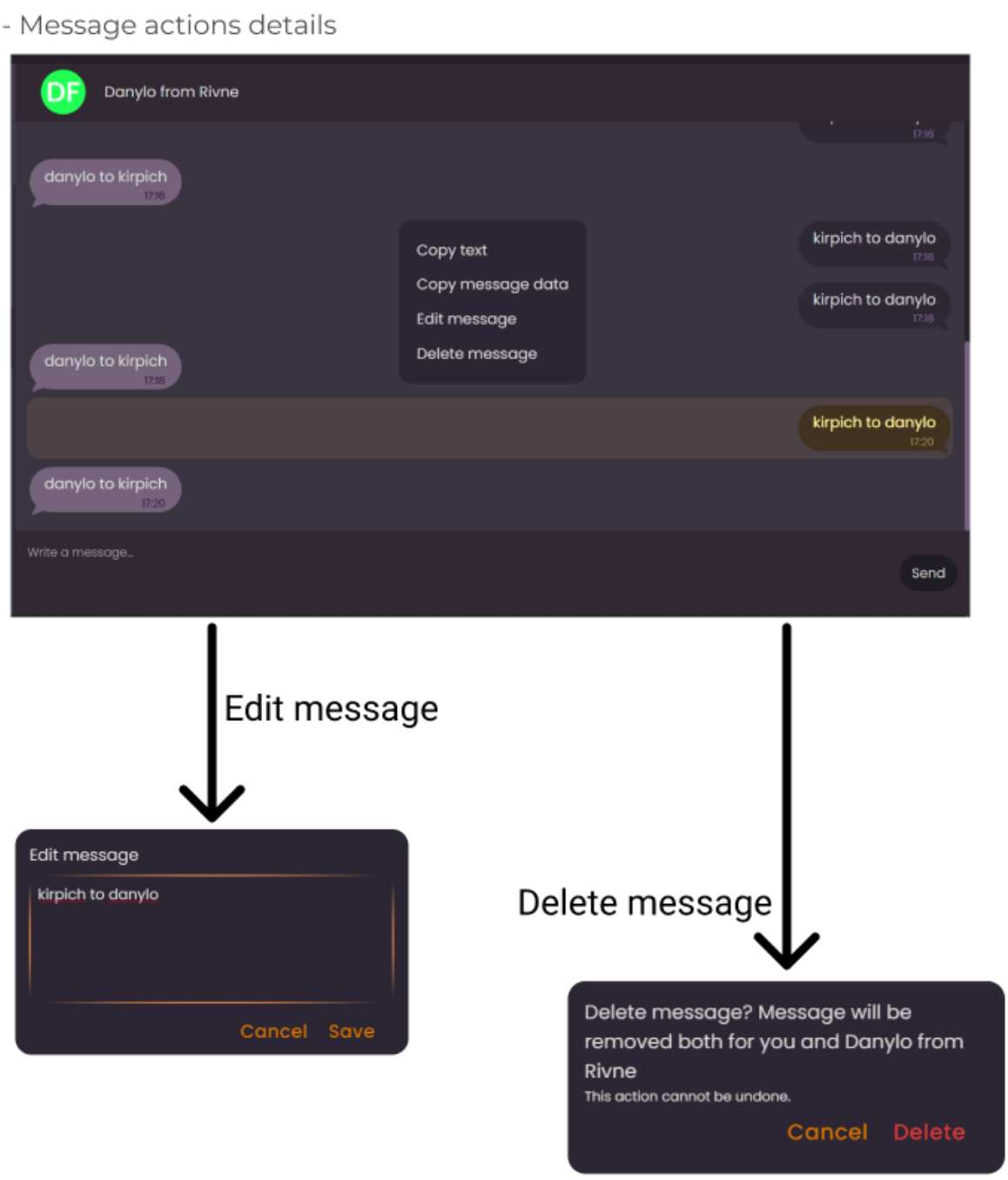


Рис. 2.25. Схема опцій над повідомлення та порядок дій для редагування та видалення повідомлень

Загальний вигляд інтерфейсу наведено у знімках екрану. Сторінка реєстрації має форму для реєстрації з такими полями:

– ім’я;
– прізвище;
– юзернейм;

– електронна пошта;

– номер телефону;

– пароль;

– повтор паролю.

Також на сторінці є кнопка реєстрації, що відсилає дані на сервер, та посилання на форму логіну, якщо користувач вже має акаунт (рис 2.25).

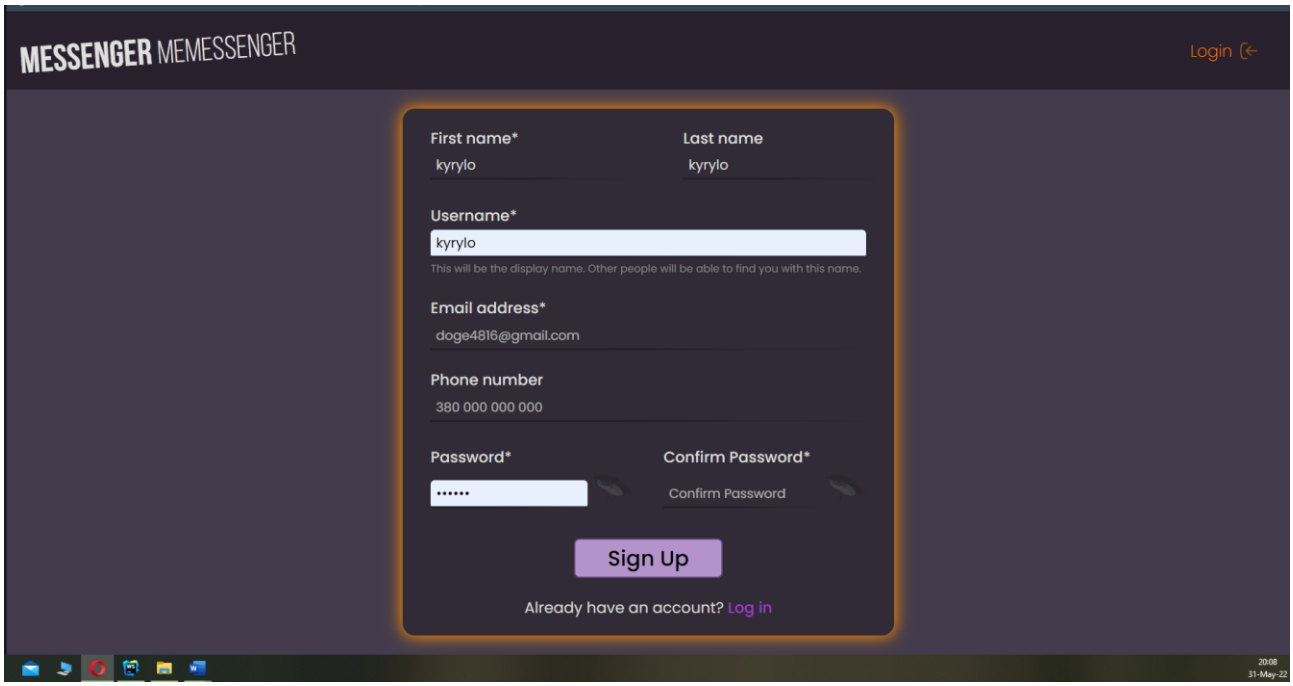


Рис. 2.26. Вигляд сторінки реєстрації

Сторінка входу в обліковий запис має невелику форму з двома полями – юзернейм та пароль. Під формою знаходяться кнопка з посиланням на форму реєстрації, на випадок, якщо користувач немає акаунту в додатку та випадково потрапив на сторінку логіну (рис. 2.26).

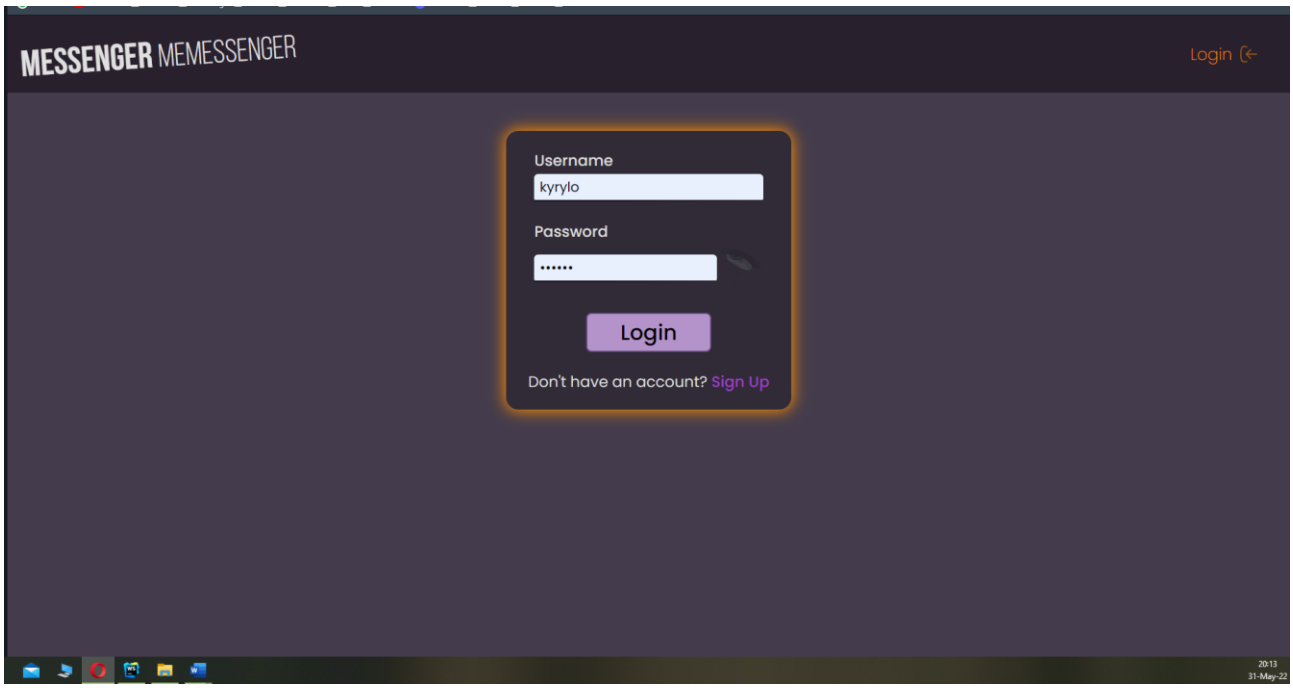


Рис. 2.27. Сторінка входу в обліковий запис

Усі сторінки мають хедер (шапку), що містить логотип месенджеру та кнопку для входу (виходу) у обліковий запис.

Якщо юзер не увійшов до облікового запису, в хедері відображається посилання на сторінку логіну (рис. 2.27).


Рис. 2.28. Вигляд шапки сайту для користувачів, що не увійшли в додаток

Якщо користувач увійшов до акаунту, в хедері відображається юзернейм користувача та посилання для виходу з додатку, що переводить його на сторінку логіну (рис. 2.28).


Рис. 2.29. Вигляд шапки сайту при вході у обліковий запис

Після входу на сайт, користувач бачить загальний інтерфейс, та список чатів. У вікні чату відображається повідомлення, що закликає користувача вибрати чат, щоб почати переписуватись (рис. 2.29). Чати сортуються за часом відправки повідомлення, тобто діалоги, в яких повідомлення було відправлене раніше знаходяться знизу, а ті, де повідомлення новіше – зверху.

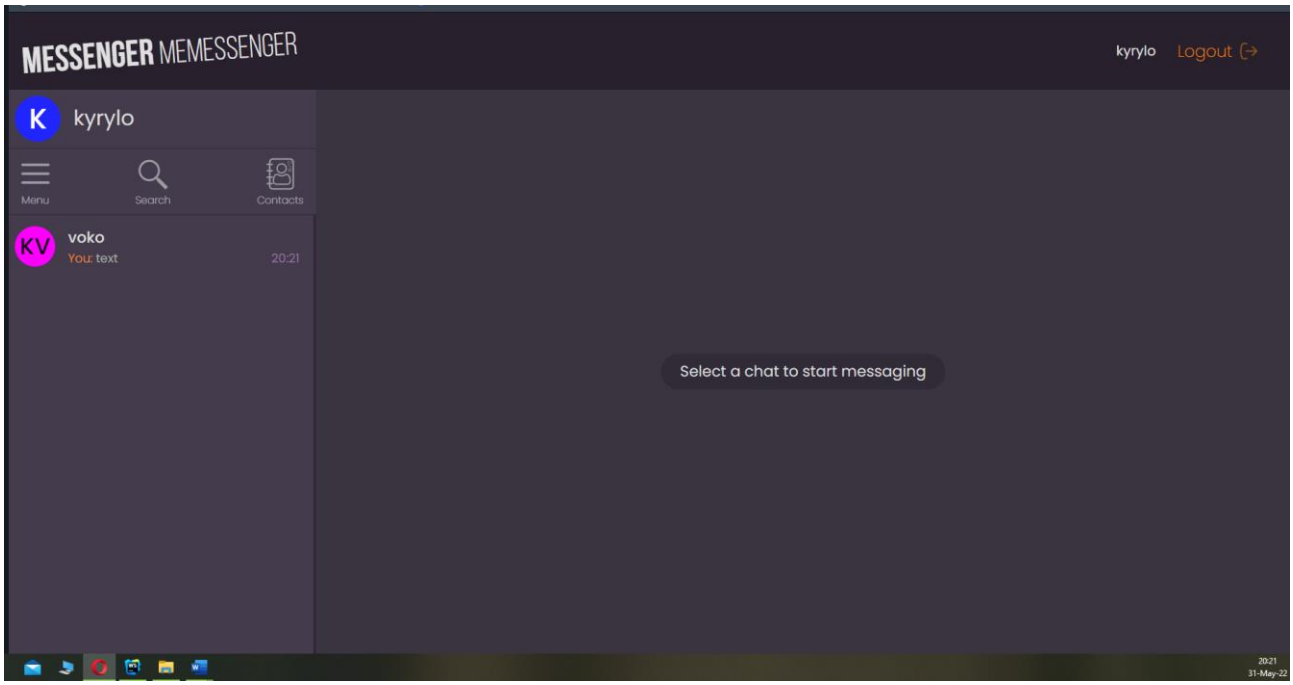


Рис. 2.30. Вікно чату одразу після логіну

При виборі діалогу, він відкривається на все вікно чату, та відображає історію повідомлень з цим користувачем. Зверху можна побачити ім’я та прізвище користувача, та його ініціали (рис. 2.30). Після відкриття чату, список чатів не пропадає.

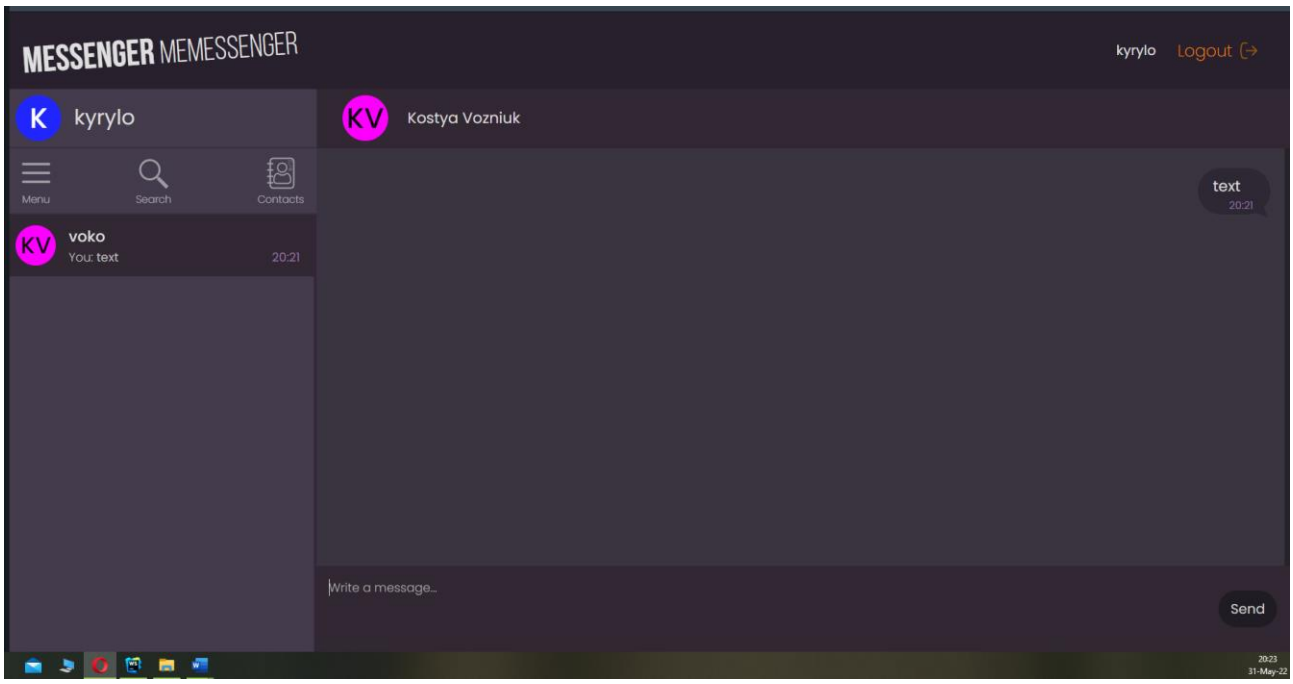


Рис. 2.31. Вікно діалогу з відкритим чатом

Якщо натиснути на кружечок з ініціалами або ім’я, відкриється профіль цього користувача (рис. 2.31).

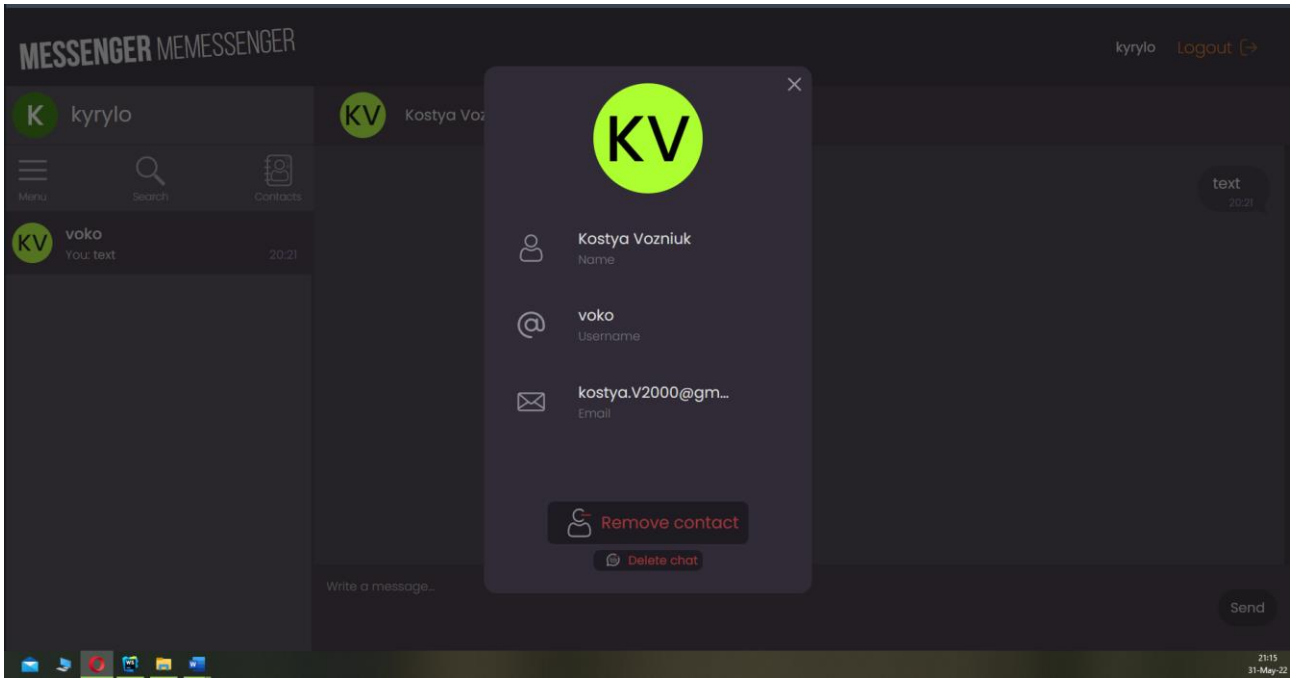


Рис. 2.32. Вікно діалогу з відкритим чатом та профілем співрозмовника

Зліва від вікна діалогу розташовано список чатів, над яким знаходяться функціональні кнопки для відкриття меню, пошуку, книги контактів. Над ними вказано юзернейм користувача, натиснувши на який можна відкрити профіль, та редагувати його (рис. 2.32).

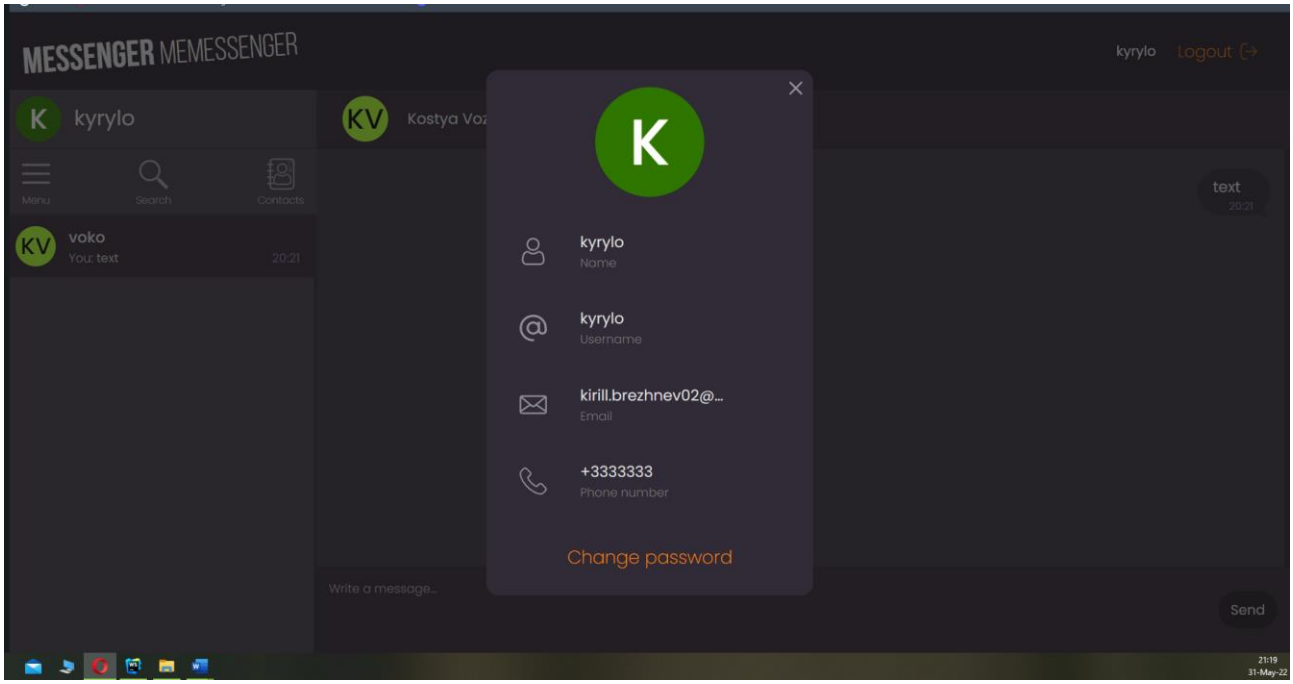


Рис. 2.33. Профіль користувача

При наведені курсором миші на поля, вони підсвічуються та з’являється іконка олівця, що натякає користувачу на можливість редагування (рис 2.33).

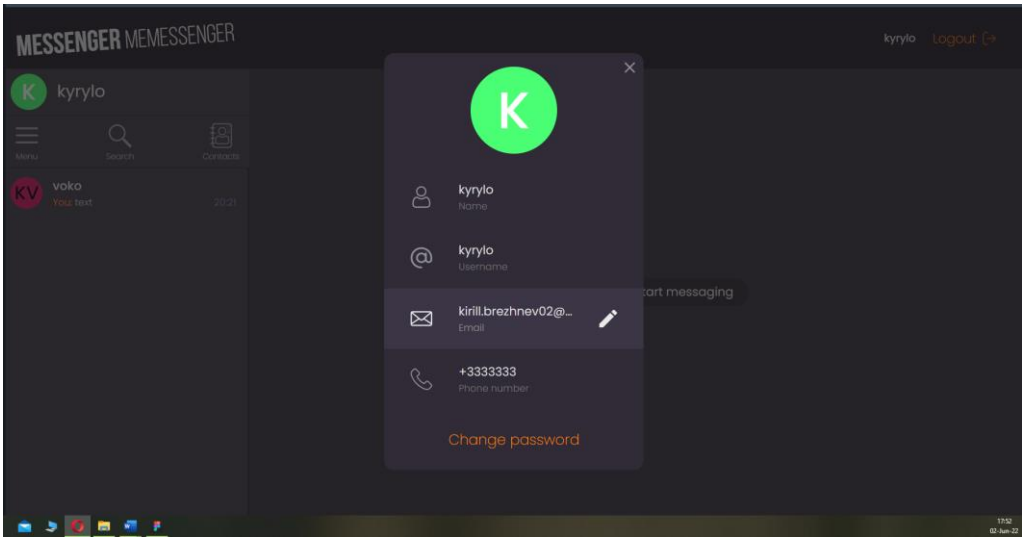


Рис. 2.34. Підсвітка редагування поля

При натисканні на поле, користувач може змінити його значення (рис. 2.34).

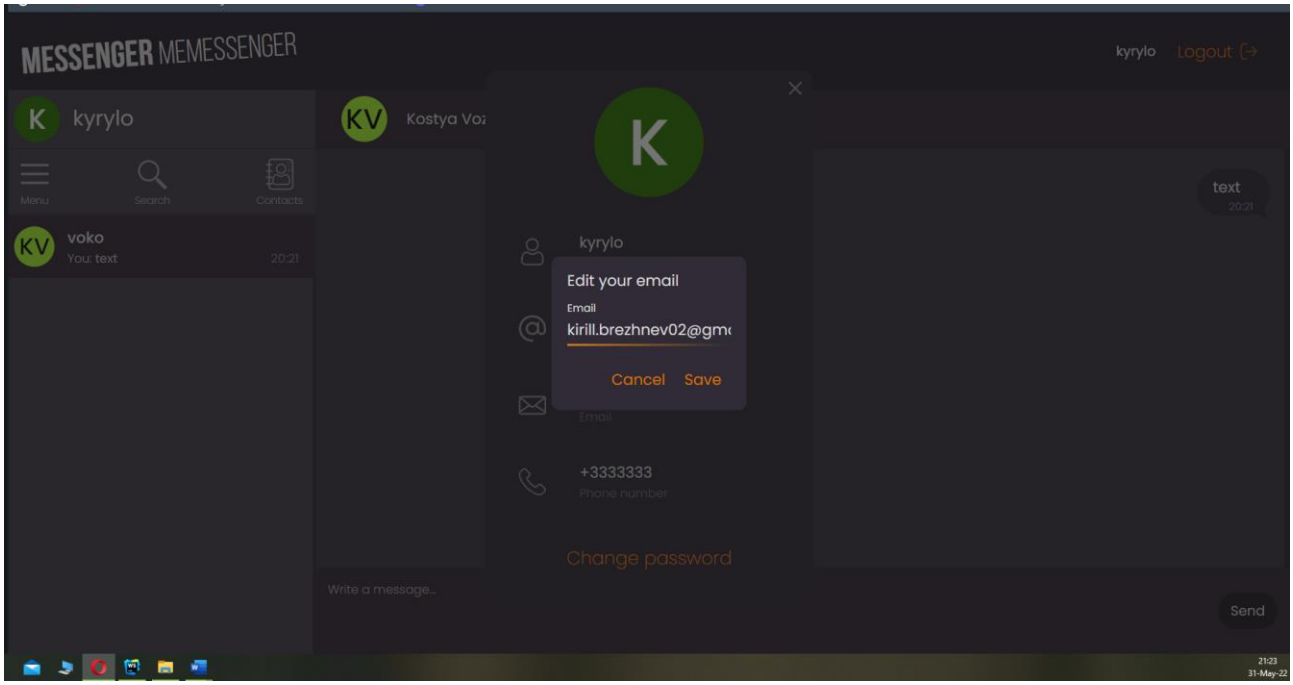


Рис. 2.35. Редагування електронної адреси користувача

# РОЗДІЛ 3
# ЕКОНОМІЧНИЙ РОЗДІЛ

## 3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2030;
2. коефіцієнт кореляції програми в ході її розробки – 0,05;
3. коефіцієнт складності програми – 1,8;
4. годинна заробітна плата програміста– 140 грн/год;

Зарплата програміста (Junior Frontend developer) була обчислена на основі даних «Української спільноти програмістів (DOU)» [23].

Наприкінці 2021 року, інформація за грудень, зарплата Junior FE розробника починається від 650$ до 1200$. За даними тієї ж спільноти, медіанна заробітна плата по Україні становить 900$ у місяць.

Враховуючі курс валют НБУ на початок січня 2022 року один долар США дорівнює 27,27 грн, тому середня зарплата в гривнях дорівнює 24543 грн. При графіку 40 робочих годин на тиждень (176 годин/місяць) зарплата за годину буде становити близько 140 грн.

5. коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,4;
7. вартість машино-години ЕОМ – 12 грн/год.

Оскільки для цього проекту потрібна велика потужність ПК для розгортання на локальному сервері, гарним рішенням буде оренда комп’ютера на час розробки додатку. Вартість ноутбуку, що підходить для виконання поставленої задачі на місяць починається від 20000 грн. Оренда такого пристрою складає 2025 грн. Оренда не передбачає повернення техніки на вихідних або у неробочій час власнику, але робота на ноутбуці проходить тільки у робочій час, тож вартість оренди погодинно вираховується на основі 176 робочих годин на місяць.

Тож вартість ЕОМ за годину роботи буде становити 12 грн. В цю вартість входить ремонт за гарантією [24].

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

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

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

$t_u$ - витрати праці на дослідження алгоритму рішення задачі;

$t_a$- витрати праці на розробку блок-схеми алгоритму;

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

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

$t_д$ - витрати праці на підготовку документації.

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

Умовне число операторів (підпрограм):

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

де $q$ - передбачуване число операторів (2030);

$C$ - коефіцієнт складності програми (1,8);

$p$ - коефіцієнт кореляції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}8 \cdot 2030 \cdot (1 + 0{,}05) = 3836{,}7$$

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

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

де $B$ - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

$k$ - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 4 до 8 років він складає 1,4.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% ($B$ = 1,3). З урахуванням коефіцієнта кваліфікації $k$ = 1,4, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (3836{,}7 \cdot 1{,}3) / (75 \cdot 1{,}4) = 47{,}50 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

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

де $Q$ – умовне число операторів програми;

$k$ – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), отримаємо:

$$t_a = 3836{,}7 / (20 \cdot 1{,}4) = 137{,}03 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

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

$$t_n = 3836{,}7 / (25 \cdot 1{,}4) = 109{,}62 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

–за умови автономного налагодження одного завдання:

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

$$t_{отл} = 3836{,}7 / (5 \cdot 1{,}4) = 548{,}1 \text{ людино-годин.}$$

–за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл} \text{ , людино-годин} \tag{3.7}$$

$$t^k_{отл} = 1{,}5 \cdot 548{,}1 = 822{,}15 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до} \text{ , людино-годин} \tag{3.8}$$

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

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

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

$$t_{до} = 0{,}75 \cdot t_{др} \text{ , людино-годин} \tag{3.10}$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 3836{,}7 / (18 \cdot 1{,}4) = 152{,}25 \text{ людино-годин.}$$

$$t_{до} = 0{,}75 \cdot 152{,}25 = 114{,}18 \text{ людино-годин.}$$

$$t_{д} = 152{,}25 + 114{,}18 = 266{,}43 \text{ людино-годин.}$$

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

$$t = 50 + 47{,}50 + 137{,}03 + 109{,}62 + 548{,}1 + 266{,}43 = 1159 \text{ людино-годин.}$$

### 3.2. Розрахунок витрат на створення програми

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

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн} \tag{3.11}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн} \tag{3.12}$$

де $t$ - загальна трудомісткість, людино-годин,

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

З урахуванням того, що середня годинна зарплата програміста становить 125 грн / год, отримуємо:

$$З_{ЗП} = 1159 \cdot 140 = 162\,260 \text{ грн.}$$

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

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн,} \tag{3.13}$$

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

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (12 грн/год).

Підставивши в формулу (3.14) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 548{,}1 \cdot 12 = 6\,577{,}2 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 162\,260 + 6\,577{,}2 = 168\,837{,}2 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.} \qquad (3.14)$$

де $B_k$- число виконавців (дорівнює 1);

$F_p$ - місячний фонд робочого часу (при 40 годинному робочому тижні $F_p$=176 годин).

Звідси, за формулою (3.14) витрати на створення програмного продукту:

$$T = 1159 / (1 \cdot 176) \approx 6{,}58 \text{ міс.}$$

**Висновок:** додаток розроблено для забезпечення можливості користувачів до спілкування один з одним, за допомогою чатів, що оновлюються у режимі реального часу. Вартість даного програмного забезпечення становить близько 168 837,2 грн. Очікуваний час розробки становить 1159 годин, тобто орієнтовно 6,58 місяців. Цей термін пов'язаний зі значним числом операторів, і включає час

на написання документації-опису, необхідної для створення додатку, дослідження і розробку алгоритму вирішення поставленого завдання, програмування за готовим алгоритмом, налагодження і розгортання програми, та підготовку вторинної документації, що допомагає у підтримці розробленого ПЗ.

# ВИСНОВКИ

Головною задачею кваліфікаційної роботи було розробити клієнтську частину соціального месенджеру на React.

Цей інтернет застосунок призначено для надання можливості користувачам обмінюватися текстовими повідомленнями у режимі реального часу, знаходячись на будь-якій відстані один від одного. Основною вимогою, для використання месенджера є наявність інтернет зв’язку та пристрою, який має вихід в інтернет за допомогою браузера.

Практичне значення системи полягає у забезпеченні користувачів способу підтримання контакту з іншими особами, друзями, рідними у ситуаціях, що не передбачають застосування GSM зв’язку. Це актуальна проблема у наш час, адже мобільний зв’язок може бути відсутнім через різні причини, у тому числи екстрені.

Під час виконання роботи було виконано наступні задачі:

- досліджено предметну галузь поставленого завдання;
- створено алгоритм для реалізації усіх цілей проекту;
- створено базу даних, клієнтську та серверні програми, які працюють разом;
- розгорнуто на реальному хостингу усі частини застосунку – БД, BE, FE;
- протестовано на реальному середовищі на рахунок наявності програмних дефектів.

Розроблений застосунок дозволяє:

- реєструватися у системі новим користувачам, та заходити під своїми особистими даними у додаток;
- шукати інших користувачів;
- писати користувачам приватні повідомлення, що зберігаються у зашифрованому вигляді на сервері;
- редагувати та видаляти власні повідомлення;
- додавати користувачів в контакти, та видаляти звідти;

– змінювати власний профіль користувача.

Клієнтську частину додатку реалізовано на основі фреймворку React з використанням мови JavaScript. Для реалізації функціоналу обміну повідомленнями використовується протокол WebSocket, а для усіх інших функцій – HTTP.

При написанні проекту використовувався компонентний підхід, що дозволяє компактно організувати структуру проекту та повторно використовувати вже існуючий код.

Також проаналізовано час, затрачений на створення програми, що складає 1159 людино-годин, або 6,58 місяців роботи, та вартість розробки додатку.

Включно із зарплатнею для програміста та орендою обладнання для написання застосунку, вартість складе 168 837,2 грн.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Windows 7 system requirements [Електронний ресурс] / Support Microsoft – Режим доступу до ресурсу: https://support.microsoft.com/en-us/windows/windows-7-system-requirements-df0900f2-3513-a851-13e7-0d50bc24e15f.

2. Create a New React App [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/docs/create-a-new-react-app.html.

3. SOLID (об'єктно-орієнтоване програмування) [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/SOLID_(об%27єктно-орієнтоване_програмування).

4. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript.

5. 2021 Developer Survey – Most popular technologies: web frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks.

6. Top Front-end Development Frameworks in 2020 [Електронний ресурс] – Режим доступу до ресурсу: https://scand.com/company/blog/top-5-front-end-development-frameworks/.

7. 2021 Developer Survey – Most loved, dreaded and wanted web frameworks [Електронний ресурс] – Режим доступу до ресурсу: https://insights.stackoverflow.com/survey/2021#section-most-loved-dreaded-and-wanted-web-frameworks.

8. What is Single Page Application (SPA)? Pros and Cons with Examples [Електронний ресурс] – Режим доступу до ресурсу: https://www.netsolutions.com/insights/single-page-application/#single-page-application-pros-and-cons.

9. Single-Page Application Vs Multi-Page Application [Електронний ресурс] – Режим доступу до ресурсу: https://lvivity.com/single-page-app-vs-multi-page-app

10. Virtual DOM and Internals [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/docs/faq-internals.html

11. Вступ до JSX [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/docs/introducing-jsx.html

12. JSX (JavaScript) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JSX_(JavaScript)

13. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST

14. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/

15. Representational state transfer [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Representational_state_transfer

16. The WebSocket API (WebSockets) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

17. The WebSocket Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc6455

18. WebSocket [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.info/websocket. Дата звернення: 12.02.2022.

19. WebSocket [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/WebSocket

20. Can I use WebSocket [Електронний ресурс] – Режим доступу до ресурсу: https://caniuse.com/?search=websocket

21. WebSockets vs. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://ably.com/topic/websockets-vs-http

22. HTTP response status codes [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

23. Зарплатне опитування DOU [Електронний ресурс] – Режим доступу до ресурсу:https://jobs.dou.ua/salaries/?period=2021-12&position=Junior%20SE&technology=JavaScript&experience=1-10

24. Оренда ноутбуків [Електронний ресурс] – Режим доступу до ресурсу:https://a-pc.com.ua/index.php?route=information/information&information_id=12

25. Messenger Memessenger FE [Електронний ресурс] – Режим доступу до ресурсу: https://messenger-memesenger.herokuapp.com/login

26. Messenger Memessenger BE [Електронний ресурс] – Режим доступу до ресурсу: https://cheese-giant.herokuapp.com/swagger-ui/index.html#/

## ДОДАТОК А

## КОД ПРОГРАМИ

**index.html** // головний HTML файл

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8"/>
   <link rel="icon" href="%PUBLIC_URL%/favicon.ico"/>
   <meta name="viewport" content="width=device-width, initial-scale=1"/>
   <meta name="theme-color" content="#000000"/>
   <meta
         name="description"
         content="Web site created using create-react-app"
   />
   <link rel="manifest" href="%PUBLIC_URL%/manifest.json"/>
   <title>Messenger</title>
</head>
<body>
<noscript>You need to enable JavaScript to run this app.</noscript>
<div id="root"></div>
</body>
</html>
```

**index.js** // головний JS файл

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
   <React.StrictMode>
      <App/>
   </React.StrictMode>,
   document.getElementById('root')
);
reportWebVitals();
```

**App.js** // основний JS файл у якому знаходиться власне додаток

```
import Header from "./components/Header/Header";
import './App.css';
import React, {useEffect, useState} from 'react';
import RegistrationForm from "./components/Authorization/RegistrationForm/RegistrationForm.js"
import ChatWindow from "./components/ChatWindow/ChatWindow";
import LoginForm from "./components/Authorization/LoginForm/LoginForm";
import {BrowserRouter as Router, Route, Switch, Redirect} from "react-router-dom";
import {getLocalWithExpiry} from "./constants/localStorage";
```

```
function App() {

    /* this state defines if the user is logged in
    according to value of this state user is redirected to /chat or to /login pages
    */
    const [isLoggedIn, setIsLoggedIn] = useState(false)
    const [registrationSuccess, setRegistrationSuccess] = useState(false)
    const [currentUser, setCurrentUser] = useState({
        id: 0
    })

    useEffect(() => {
        const JWT = getLocalWithExpiry('token')

        if (JWT !== null && JWT !== "") {
            setIsLoggedIn(true)
        } else if (JWT === null || JWT === "") {
            setIsLoggedIn(false)
        }
    }, [isLoggedIn])

    return (
        <Router>
            <div className="App">
                <Header
                    isLoggedIn={isLoggedIn}
                    setIsLoggedIn={setIsLoggedIn}
                    currentUser={currentUser}
                    setCurrentUser={setCurrentUser}/>
                <div className="content">
                    <Switch>
                        {/*if user tries to access / route he will be redirected regardless of isLoggedIn value
                        if the user is logged in /chat opens
                        if the user is not logged in /login page opens
                        */}
                        <Route exact path="/">
                            {isLoggedIn ? <Redirect to="/chat"/> : <Redirect to="/login"/>}
                        </Route>

                        {/* this route forbids user to go to /login page after he had authorized*/}
                        {/*//comment lines marked by 1 to access /login route after authorizing*/}
                        <Route path='/login'>
                            {isLoggedIn === false ?         //1
                                <LoginForm
                                    setIsLoggedIn={setIsLoggedIn}
                                    currentUser={currentUser}
                                    registrationSuccess={registrationSuccess}
                                    // setCurrentUser={setCurrentUser}
                                />
                                : <Redirect to="/chat"/>    //1
                            }
```

```
                </Route>
                {/* this route forbids user to go to /signup page after he had authorized*/}
                <Route path='/signup'>
                  {isLoggedIn ?
                    <Redirect to="/chat"/>
                    : <RegistrationForm
                      currentUser={currentUser}
                      setRegistrationSuccess={setRegistrationSuccess}/>}
                </Route>

                <Route path='/chat'>
                  <ChatWindow
                    isLoggedIn={isLoggedIn}
                    setIsLoggedIn={setIsLoggedIn}
                    currentUser={currentUser}
                    setCurrentUser={setCurrentUser}
                  />
                </Route>
              </Switch>
            </div>
          </div>
        </Router>
    );
}

export default App;
```

**package.json** // JSON файл, що містить усі пакети, необхідні для роботи додатку

```
{
  "name": "messenger",
  "version": "0.1.0",
  "private": true,
  "engines": {
    "node": "14.15.0",
    "npm": "6.14.8"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "dependencies": {
    "@stomp/stompjs": "^6.1.0",
    "@testing-library/jest-dom": "^5.11.9",
    "@testing-library/react": "^11.2.5",
    "@testing-library/user-event": "^12.8.3",
    "axios": "^0.21.1",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-router-dom": "^5.2.0",
    "react-scripts": "^5.0.1",
```

```
    "seedrandom": "^3.0.5",
    "sockjs-client": "^1.5.1",
    "use-sound": "^3.0.1",
    "web-vitals": "^1.1.1"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

**RegistrationForm.js** // компонент форми реєстрації

```
import React, {useEffect, useRef, useState} from 'react';
import style from './RegistrationForm.module.css';
import generalStyle from '../generalAuth.module.css';
import {NavLink, useHistory} from "react-router-dom";
import {API_PATH} from "../../../constants/API_PATH_DEFAULT";
import axios from "axios";
import {
    usernameValidation, firstnameValidation, lastnameValidation,
    emailValidation, phoneValidation, passwordValidation, confirmPasswordValidation
} from "../../../constants/validation";
import eyeIcon from '../../../media/icons/eye-icon.svg'
import noEyeIcon from '../../../media/icons/no-eye-icon.svg'

const initialFormData = {
    firstname: '',
    lastname: '',
    username: '',
    email: '',
    phoneNumber: '',
    password: '',
    passwordConfirm: ''
}

function RegistrationForm(props) {
    const {currentUser, setRegistrationSuccess} = props
```

```
const [formData, setFormData] = useState(initialFormData)
   const {password, passwordConfirm} = formData
const [showPassword, setShowPassword] = useState({
   p: 'password',
   c: 'password',
   pp: eyeIcon,
   cp: eyeIcon
})
const [validatingError, setValidatingError] = useState({})
const [touched, setTouched] = React.useState({});
const defaultPasswordClass = generalStyle.inputField
const [passwordClass, setPasswordClass] = useState(defaultPasswordClass)
let history = useHistory()

const passwordRef = useRef(null)
const passwordConfirmRef = useRef(null)

//sets type for input field to make password visible
const toggleVisibility = (field, pic) => {
   if (showPassword[field] === 'password') {
      setShowPassword(prevShow => ({
         ...prevShow,
         [field]: 'text',
         [pic]: noEyeIcon
      }))
   } else {
      setShowPassword(prevShow => ({
         ...prevShow,
         [field]: 'password',
         [pic]: eyeIcon
      }))
   }
}
const togglePassword = () => {
   toggleVisibility('p', 'pp')
   passwordRef.current.focus()
}
const toggleConfirmPassword = () => {
   toggleVisibility('c', 'cp')
   passwordConfirmRef.current.focus()
}
useEffect(() => {
   if (password !== '' && password === passwordConfirm) {
      setPasswordClass(`${defaultPasswordClass} ${generalStyle.match}`)
   } else if (password !== ''
      && passwordConfirm !== ''
      && password !== passwordConfirm) {
      setPasswordClass(`${defaultPasswordClass} ${generalStyle.noMatch}`)
   } else if (password === ''
      && passwordConfirm === '') {
      setPasswordClass(defaultPasswordClass)
   }
```

```
}, [password, passwordConfirm, defaultPasswordClass])

const handleChange = e => {
   setFormData(prevFormData => ({
      ...prevFormData, [e.target.name]: e.target.value
   }))
   setTouched({
      ...touched,
      [e.target.name]: true,
   });
}
const handleBlur = evt => {
   const {name, value} = evt.target;
   // remove whatever error was there previously
   const {[name]: removedError, ...rest} = validatingError;
   // check for a new error
   const error = validate[name](value);
   // validate the field if the value has been touched
   setValidatingError({
      ...rest,
      ...(error && {[name]: touched[name] && error}),
   });
};

useEffect(() => {
}, [validatingError])

const validate = {
   username: usernameValidation,
   firstname: firstnameValidation,
   lastname: lastnameValidation,
   email: emailValidation,
   phoneNumber: phoneValidation,
   password: passwordValidation,
   passwordConfirm: confirmPasswordValidation
};

const handleSubmit = e => {
   e.preventDefault();
   let isValidated = false

   //validate the form
   const formValidation = Object.keys(formData).reduce(
      (acc, key) => {
         const newError = validate[key](formData[key]);
         const newTouched = {[key]: true};
         return {
            errors: {
               ...acc.errors,
               ...(newError && {[key]: newError}),
            },
            touched: {
```

```
                ...acc.touched,
                ...newTouched,
            },
        };
    },
    {
        errors: {...validatingError},
        touched: {...touched},
    },
);
setValidatingError(formValidation.errors);
setTouched(formValidation.touched);

if (
    !Object.values(formValidation.errors).length && // errors object is empty
    Object.values(formValidation.touched).length ===
    Object.values(formData).length && // all fields were touched
    Object.values(formValidation.touched).every(t => t === true) // every touched field is true
) {
    isValidated = true
}
if (isValidated) {
    sendDataToServer()
}
}
const sendDataToServer = () => {
    const registrationData = {
        firstname: formData.firstname,
        lastname: formData.lastname,
        username: formData.username,
        email: formData.email,
        phoneNumber: formData.phoneNumber,
        password: formData.password
    }
    //
    axios.post(`${API_PATH}/sign-up`, registrationData)
        .then(response => {
            console.log(response.status)
            setRegistrationSuccess(true)
            currentUser.username = registrationData.username
            history.push('/login')
        })
        .catch(error => {
            alert(error)
            console.log(error)
        })
}
return (
    <div className={style.formWrap}>
        <form onSubmit={handleSubmit}
              className={generalStyle.authForm}
              autoComplete='off'
```

```
            spellCheck='false'>
          {/*firstname lastname*/}
          <div className={style.twoFieldsWrap}>

            <div className={generalStyle.fieldLabelWrapper}>
              {touched.firstname && validatingError.firstname ?
              <div className={generalStyle.validationError}>{validatingError.firstname}</div>:
              <label className={generalStyle.fieldLabel} htmlFor="firstName">First
name<span>*</span></label>
              }
              <input type="text"
                  name="firstname"
                  placeholder="Adolf"
                  value={formData.firstname}
                  onChange={handleChange}
                  onBlur={handleBlur}
                  maxLength={30}
                  className={generalStyle.inputField}
              />
            </div>
            <div className={generalStyle.fieldLabelWrapper}>
              {touched.lastname && validatingError.lastname ?
              <div className={generalStyle.validationError}>{validatingError.lastname}</div>:
              <label className={generalStyle.fieldLabel} htmlFor="lastName">Last
name</label>
              }
              <input type="text"
                  name="lastname"
                  id="lastname"
                  placeholder="Obama"
                  value={formData.lastname}
                  onChange={handleChange}
                  onBlur={handleBlur}
                  maxLength={30}
                  className={generalStyle.inputField}
              />
            </div>
          </div>
          {/*//firstname lastname*/}
          <div className={generalStyle.fieldLabelWrapper}>
            {touched.username && validatingError.username ?
            <div className={generalStyle.validationError}>{validatingError.username}</div>:
            <label className={generalStyle.fieldLabel}
htmlFor="username">Username<span>*</span></label>
            }
            <input type="text"
                id='username'
                name="username"
                placeholder="doge69"
                value={formData.username}
                onChange={handleChange}
                onBlur={handleBlur}
```

```
                maxLength={25}
                className={generalStyle.inputField}
            />
            <p className={generalStyle.hint}>This will be the display name. Other people will be
able to find
               you with
               this name.</p>
         </div>
         <div className={generalStyle.fieldLabelWrapper}>
            {touched.email && validatingError.email ?
            <div className={generalStyle.validationError}>{validatingError.email}</div>:
            <label className={generalStyle.fieldLabel} htmlFor="email">
               Email address<span>*</span></label>
            }
            <input type="email"
                id='email'
                name="email"
                autoComplete='email'
                placeholder="doge4816@gmail.com"
                value={formData.email}
                onChange={handleChange}
                onBlur={handleBlur}
                maxLength={40}
                className={generalStyle.inputField}
            />
         </div>
         <div className={generalStyle.fieldLabelWrapper}>
            {touched.phoneNumber && validatingError.phoneNumber ?
               <div
className={generalStyle.validationError}>{validatingError.phoneNumber}</div> :
               <label className={generalStyle.fieldLabel} htmlFor="phoneNumber">Phone
number</label>
            }
            <input type="text"
                name="phoneNumber"
                id="phoneNumber"
                placeholder="380 000 000 000"
                value={formData.phoneNumber}
                onChange={handleChange}
                onBlur={handleBlur}
                maxLength={12}
                className={generalStyle.inputField}
            />
         </div>

         {/*password confirm*/}
         <div className={style.twoFieldsWrap}>
            <div className={generalStyle.fieldLabelWrapper}>
               {touched.password && validatingError.password ?
                  <div
className={generalStyle.validationError}>{validatingError.password}</div> :
                  <label className={generalStyle.fieldLabel} htmlFor="password">
```

```
                    Password<span>*</span></label>
                }
                <div className={generalStyle.passwordWrap}>
                  <input type={showPassword.p}
                      name="password"
                      id="password"
                      ref={passwordRef}
                      placeholder="Password"
                      value={formData.password}
                      onChange={handleChange}
                      onBlur={handleBlur}
                      maxLength={30}
                      className={passwordClass}
                  />
                  <img src={showPassword.pp}
                     alt="Show password"
                     className={generalStyle.passwordIcon}
                     onClick={togglePassword}/>
                </div>
              </div>

              <div className={generalStyle.fieldLabelWrapper}>
                {touched.passwordConfirm && validatingError.passwordConfirm ?
                <div
classNameName={generalStyle.validationError}>{validatingError.passwordConfirm}</div>:
                <label className={generalStyle.fieldLabel} htmlFor="confirmPassword">
                  Confirm Password<span>*</span></label>
                }
                <div className={generalStyle.passwordWrap}>
                  <input type={showPassword.c}
                      name="passwordConfirm"
                      id="passwordConfirm"
                      ref={passwordConfirmRef}
                      placeholder="Confirm Password"
                      value={formData.passwordConfirm}
                      onChange={handleChange}
                      onBlur={handleBlur}
                      maxLength={30}
                      className={passwordClass}
                  />
                  <img src={showPassword.cp}
                     alt="Show password"
                     className={generalStyle.passwordIcon}
                     onClick={toggleConfirmPassword}/>
                </div>
              </div>
            </div>
            <div className={generalStyle.submitBtnWrap}>
              <button type="submit" className={generalStyle.submitBtn}>Sign Up</button>
            </div>
            <p className={generalStyle.loginHint}>Already have an account? <NavLink to='/login'
                                                    className={generalStyle.loginLink}>Log
```

```
                in</NavLink></p>
            </form>
        </div>
    )
}

export default RegistrationForm
```

**generalAuth.module.css** // CSS файл що відповідає за стилі форм реєстрації та логіну

```
.authForm {
    height: fit-content;
    color: #dddddd;
    background-color: #302b36;
    padding: 0 1vw 1vw 1vw;
    font-size: calc(10px + 0.4vw + 0.5vh);
    box-sizing: border-box;
    border-radius: 18px;
    box-shadow: #FF8C00 0 0 22px 2px;
}
.fieldLabel {
    margin: 0;
    padding: 0.8em 0 0.4vmin;
    width: fit-content;
    font-size: 1em;
    font-weight: 500;
}
.fieldLabelWrapper {
    position: relative;
    display: flex;
    flex-direction: column;
    margin: 0.4em 1em;
}
.inputField {
    font-size: 0.9em;
    padding: 4px 8px;
    width: inherit;
    background-color: unset;
    color: #dedede;
    border: none;
    border-bottom: 3px solid;
    border-image: linear-gradient(to right, #24202b 25%, #24202b 40%, #302b36 100%) 1;
    text-overflow: ellipsis;
    border-radius: 6px;
}
.inputField:focus {
    border-image: linear-gradient(to right, rgba(246, 140, 27, 0.88), #302b36 98%) 1;
}
.inputField::placeholder {
    color: #ffffff;
    font-size: 0.94em;
```

```
    opacity: 0.6;
}
.inputField:focus::placeholder {
    opacity: 0.4;
}
.match, .match:focus {
    border-image: linear-gradient(to right, rgba(49, 246, 27, 0.88), #302b36 98%) 1;
}
.noMatch, .noMatch:focus {
    border-image: linear-gradient(to right, rgba(246, 27, 27, 0.88), #302b36 98%) 1;
}
.hint {
    font-size: 0.7em;
    color: #888888;
    margin: 4px 0;
    width: inherit;
    max-width: inherit;
}
.loginHint {
    font-size: 1em;
    padding: 0.2em 0.6em;
    text-align: center;
}
.submitBtnWrap {
    text-align: center;
}
.submitBtn {
    font-size: 1.4em;
    font-weight: 500;
    padding: 0.4vmin 48px;
    margin: 1.2vmax;
    background-color: #b493cb;
    border: 2px solid #5f4579;
    border-radius: 8px;
}
.submitBtn:hover {
    background-color: #bba5ca;
}
.submitBtn:active {
    background-color: #cb81ff;
}
.loginLink, .loginLink:link, .loginLink:visited {
    font-weight: 500;
    color: #9f40c2;
}
.loginLink:hover {
    color: #7100c1;
}
.passwordWrap{
    display: flex;
    align-items: center;
}
```

```
.passwordIcon {
    opacity: 0.1;
    width: 3em;
    right: 0;
    cursor: pointer;
    transition: opacity 1s;
}
.inputField:focus ~ .passwordIcon {
    visibility: visible;
    opacity: 0.9;
}
.validationError {
    top: 0;
    width: 100%;
    max-width: 280px;
    overflow-wrap: break-word;
    white-space: pre-wrap;
    font-size: 0.68em;
    margin: 0;
    padding: 0.8em 0 0.4vmin;
    font-weight: 300;
    color: #e33c3c;
}
.response {
    white-space: pre-wrap;
}
@media screen and (min-width: 320px) {
    .authForm {
        width: calc(300px + 420 * ((100vw - 320px) / 1500));
    }
}
@media screen and (min-width: 1820px) {
    .authForm {
        width: 720px;
    }
}
@media screen and (max-width: 800px) {
    .fieldLabelWrapper{
        margin: 0.9em 1em;
    }
    .fieldLabel{
        display: none;
    }
}
@media screen and (max-width: 812px) and (orientation: landscape){
    .authForm {
        position: absolute;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
    }
    .fieldLabelWrapper{
```

```
    margin: 0.4em 1em;
  }
  .fieldLabel{
    display: none;
  }
  .submitBtn{
    margin: 0.2em;
  }
}
```

**ChatWindow.js** // основний JS файл вікна чату

```
import React, {useEffect, useState} from 'react';
import style from './ChatWindow.module.css'
import Messages from "./Messages/Messages";
import Chats from "./Chats/Chats";
import axios from "axios";
import {SECURED_API_PATH} from "../../constants/API_PATH_DEFAULT";
import {Redirect} from "react-router-dom";
import ProfileBar from "./ProfileBar/ProfileBar";
import DefaultMessageWindow from
"./Messages/DefaultMessageWindow/DefaultMessageWindow";
import {getBearerToken} from "../../constants/getBearerToken";
import getMessagesFromChat from "../../constants/getMessagesFromChat";
import {webSocketInstance} from "../../constants/webSocketInstance";
import MenuIcons from "./MenuIcons/MenuIcons";
import RandomColor from "../../constants/RandomColor";
import useSound from "use-sound";
import messageSound from '../../media/sounds/1.mp3'

function ChatWindow(props) {
  const {isLoggedIn, setIsLoggedIn, currentUser, setCurrentUser} = props

  const [chatsData, setChatsData] = useState([])
  const [messages, setMessages] = useState([])
  const [lastMessages, setLastMessages] = useState([])
  const [selectedChat, setSelectedChat] = useState(0)
  const [secondChatUser, setSecondChatUser] = useState({})
  const [profilePictureColors, setProfilePictureColors] = useState({})
  const [sessionResponse, setSessionResponse] = useState({
    message: '',
    errorMessage: '',
    serverResponse: ''
  })
  const [dataIsFetched, setDataIsFetched] = useState({
    user: false,
    chats: false,
    lastMessages: false
  })
  const [messagesPage, setMessagesPage] = useState(1)
  const [receivedMessage, setReceivedMessage] = useState({
    id: 0,
```

```
    time: 0
})
const [messageIsSent, setMessageIsSent] = useState({
    id: 0,
    time: 0,
    isSent: true
})
const [messageChanged, setMessageChanged] = useState({
    id: 0,
    time: 0,
    edited: false,
    deleted: false
})
const [menuMessageSent, setMenuMessageSent] = useState({
    id: 0,
    time: 0,
    recipientId: 0,
    chatId: 0
})
const [chatIsDeleted, setChatIsDeleted] = useState(0)
// const {message, errorMessage, serverResponse} = sessionResponse
const [playSound] = useSound(messageSound,
    {volume: 0.25})
const cancelToken = axios.CancelToken
const source = cancelToken.source()

//initial render, data fetch
useEffect(() => {
    const JWT_header = getBearerToken('ChatWindow initial')
    if (isLoggedIn === true && JWT_header !== null) {
        getUser(JWT_header)
        getChats(JWT_header)
        getLastMessages(JWT_header)
    } else {
        setIsLoggedIn(false)
    }

    return () => {
        source.cancel("axios fetch cancelled")
        setSessionResponse(prevSessionResponse => ({
            ...prevSessionResponse,
            message: `axios fetch cancelled`,
            requestCancelled: true
        }))
    }

}, [])//useEffect

//sets WebSocket connection after successful user setting
useEffect(() => {
    if (currentUser.id !== 0) {
        webSocketInstance(setReceivedMessage, currentUser.id)
```

```
            setProfilePictureColors(prevColors => ({
                ...prevColors,
                [currentUser.id]: RandomColor()
            }))
        }
    }, [currentUser.id])

    // rerender component to display message preview, based on sent messages
    useEffect(() => {
        const JWT_header = getBearerToken('ChatWindow message sent')
        if (isLoggedIn === true && JWT_header !== null) {
            getChats(JWT_header)
            getLastMessages(JWT_header)
        }

        return () => {
            source.cancel("axios fetch cancelled")
            setSessionResponse(prevSessionResponse => ({
                ...prevSessionResponse,
                message: `axios fetch cancelled`,
                requestCancelled: true
            }))
        }
    }, [messageIsSent])
    /* rerender component to display chats and message preview,
     based on edited or deleted messages */
    useEffect(() => {
        const fetchData = async () => {
            const JWT_header = getBearerToken('ChatWindow message changed')
            //updates lastMessages array if the changed message is last
            const lastMessageChanged =
                lastMessages.find(message => message.id === messageChanged.id)

            if (lastMessageChanged) {
                if (isLoggedIn === true && JWT_header !== null) {
                    getChats(JWT_header)
                    getLastMessages(JWT_header)
                }
            }
            if (JWT_header !== null && messageChanged.id !== 0) {
                try {
                    const messages = await getMessagesFromChat(JWT_header, selectedChat)
                    console.log('her', messages)
                    if (messages !== null) {
                        setMessages(messages)
                    }
                } catch (error) {
                    console.log(error)
                }
            }
        }
        fetchData().then()
```

```
}, [messageChanged])
//rerender component to display messages after receiving a new message
useEffect(() => {
    if (receivedMessage.id !== 0) {
        const JWT_header = getBearerToken('ChatWindow message sent')
        if (isLoggedIn === true && JWT_header !== null) {
            getLastMessages(JWT_header)
            if (chatsData.some(el => el.chatId !== receivedMessage.chatId)) {
                getChats(JWT_header)
            }
        }
        // if (!messageAreaHasFocus) {
        if (selectedChat === 0 ||
            selectedChat !== receivedMessage.chatId) {
            playSound()
        }
        return () => {
            source.cancel("axios fetch cancelled")
            setSessionResponse(prevSessionResponse => ({
                ...prevSessionResponse,
                message: `axios fetch cancelled`,
                requestCancelled: true
            }))
        }
    }
}, [receivedMessage])
//rerender components to display new message sent from menu
useEffect(() => {
    if (menuMessageSent.id !== 0) {
        const JWT_header = getBearerToken('ChatWindow menu message sent')
        if (isLoggedIn === true && JWT_header !== null) {
            getChats(JWT_header)
            getLastMessages(JWT_header)
            if (selectedChat === menuMessageSent.chatId) {
                getMessagesFromChat(JWT_header, selectedChat)
                    .then(response => setMessages(response))
            }
        }
        console.log(menuMessageSent)
    }
}, [menuMessageSent])
useEffect(() => {
    if (chatIsDeleted !== 0) {
        const JWT_header = getBearerToken('ChatWindow menu message sent')
        if (isLoggedIn === true && JWT_header !== null) {
            getChats(JWT_header)
            setSelectedChat(0)
        }
        console.log(chatIsDeleted)
    }
    console.log(sessionResponse)
}, [chatIsDeleted])
```

```
const getUser = (JWT_header) => {
    axios.get(`${SECURED_API_PATH}/user`, {
        headers: {authorization: JWT_header},
        cancelToken: source.token
    })
        .then(response => {
            // console.log('ChatWindow.js getUser response.data', response.data)
            setCurrentUser(response.data)
            setDataIsFetched(prevData => ({
                ...prevData,
                user: true
            }))
        })
        .catch(error => {
            if (axios.isCancel(error)) {
                console.log(`axios fetch cancelled\n${error}`)
            }
            setIsLoggedIn(false)
            localStorage.removeItem('token')
            console.log(error)
        })
}
const getChats = (JWT_header) => {
    axios.get(`${SECURED_API_PATH}/chat`, {
            headers: {authorization: JWT_header},
            cancelToken: source.token
        }
    )
        .then(response => {
            console.log(response.data)
            setChatsData(response.data)
            setSessionResponse(prevSessionResponse => ({
                ...prevSessionResponse,
                message: `success`,
            }))
            setDataIsFetched(prevData => ({
                ...prevData,
                chats: true
            }))
        })
        .catch(error => {
            console.log(error)
            if (typeof error.response === 'undefined') {
                setSessionResponse(prevSessionResponse => ({
                    ...prevSessionResponse,
                    errorMessage: `Cannot log in due to a network error.`,
                    serverResponse: error.toString()
                }))
            } else {
                console.log(error.response)
                if (error.response.status === 401) {
                    setSessionResponse(prevSessionResponse => ({
```

```
                    ...prevSessionResponse,
                    message: `Your session was active for a long time. Please log in again`,
                }))
                setIsLoggedIn(false)
                localStorage.removeItem('token')
            } else {
                setSessionResponse(prevSessionResponse => ({
                    ...prevSessionResponse,
                    errorMessage: `Cannot continue due to an error.`,
                    serverResponse: error.toString()
                }))
                setIsLoggedIn(false)
                localStorage.removeItem('token')
            }
        }
    })
}

const getLastMessages = (JWT_header) => {
    axios.get(`${SECURED_API_PATH}/messages/last/`, {
        headers: {authorization: JWT_header},
        cancelToken: source.token
    })
        .then(response => {
            console.log('last message', response.data)
            setLastMessages(response.data)
            setDataIsFetched(prevData => ({
                ...prevData,
                time: Date.now(),
                lastMessages: true
            }))
        })
        .catch(error => {
            console.log(error, error.response)
        })
}

return (
    isLoggedIn ?
        <div className={style.chatWindow}>
            <div className={style.chatSectionWrap}>
                <ProfileBar currentUser={currentUser}
                            profilePictureColors={profilePictureColors}
                            setCurrentUser={setCurrentUser}/>
                <MenuIcons profilePictureColors={profilePictureColors}
                           setMenuMessageSent={setMenuMessageSent}
                           currentUser={currentUser}
                           setIsLoggedIn={setIsLoggedIn}
                           setCurrentUser={setCurrentUser}/>
                {(dataIsFetched.user && dataIsFetched.chats && dataIsFetched.lastMessages) &&
                <Chats chatsData={chatsData}
                       currentUser={currentUser}
```

```
                setSecondChatUser={setSecondChatUser}
                setIsLoggedIn={setIsLoggedIn}
                setMessages={setMessages}
                lastMessages={lastMessages}
                selectedChat={selectedChat}
                setSelectedChat={setSelectedChat}
                profilePictureColors={profilePictureColors}
                setProfilePictureColors={setProfilePictureColors}
                receivedMessage={receivedMessage}/>
            }
          </div>
          <div className={style.messageSectionWrap}>
            {selectedChat !== 0 ?
              <Messages selectedChat={selectedChat}
                    currentUser={currentUser}
                    secondChatUser={secondChatUser}
                    profilePictureColors={profilePictureColors}
                    messages={messages}
                    messageIsSent={messageIsSent}
                    setMessageIsSent={setMessageIsSent}
                    messagesPage={messagesPage}
                    setMessagesPage={setMessagesPage}
                    receivedMessage={receivedMessage}
                    setChatIsDeleted={setChatIsDeleted}
                    setMessageChanged={setMessageChanged}/>
              :
              <DefaultMessageWindow/>
            }
          </div>
        </div>
        : <Redirect to="/login"/>
    )
}

export default ChatWindow;
```

У цьому переліку наведено тільки частину файлів у демонстраційних цілях. Інші файли можна знайти на оптичному диску.

**ДОДАТОК Б**

**ВІДГУК**
**керівника економічного розділу**
**на кваліфікаційну роботу бакалавра**
**на тему:**
**«Розробка клієнтської частини соціального месенджера на основі JavaScript/React»**
**студента групи 122-18-2 Брежнєва Кирила Миколайовича**

**Керівник економічного розділу** **Л. В. Касьяненко**

**доцент каф. ПЕП та ПУ, к.е.н**

**ДОДАТОК В**

## ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

| **Ім’я файлу** | **Опис** |
|---|---|
| Пояснювальні документи | |
| Дипломна-робота-Брежнєв.docx | Пояснювальна записка до кваліфікаційної роботи. Документ Word. |
| Дипломна-робота-Брежнєв.pdf | Пояснювальна записка до кваліфікаційної роботи в форматі PDF. |
| Програма | |
| diploma.zip | Архів. Містить коди програми. |
| Презентація | |
| Презентація Брежнєв.pptx | Презентація кваліфікаційної роботи. |