НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Соціальна мережа для пошуку друзів та спілкування

Виконав студент IV курсу, групи ІП-92
(шифр групи)
Харенко Олексій Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Зенів І.О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент професор каф. ІПЗЕ, д.т.н., доц., Федорова Н. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ –2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Харенку Олексію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Соціальна мережа для пошуку друзів та спілкування
керівник проєкту Зенів Ірина Онуфріївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту « 17 » червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв'язання.
4) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
5) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	03.05.2023	
3	Постановка та формалізація задачі	03.05.2023	
4	Розробка інформаційного забезпечення	07.05.2023	
5	Алгоритмізація задачі	10.05.2023	
6	Обґрунтування вибору використаних технічних засобів	11.05.2023	
7	Розробка програмного забезпечення	13.05.2023	
8	Налагодження програми	21.05.2023	
9	Виконання графічних документів	24.05.2023	
10	Оформлення пояснювальної записки	24.05.2023	
11	Подання ДП на попередній захист	02.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Олексій ХАРЕНКО
(підпис) (ініціали, прізвище)

Керівник _____ Ірина ЗЕНІВ
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 79 таблиць, 21 рисунок та 15 джерел – загалом 84 сторінки.

Дипломний проєкт присвячений розробці соціальної мережі, що дозволить шукати однодумців за певними параметрами та спілкуватись з ними без використання сторонніх засобів.

Мета: покращення способу пошуку друзів за інтересами та спрощення комунікації між ними завдяки зручному вбудованому чату з функцією аудіо та відео дзвінка.

Об'єкт дослідження: програмне забезпечення для спілкування та пошуку друзів.

Предмет дослідження: процес розроблення програмного забезпечення.

У розділі 1 було наведено загальні положення, проаналізовано та описано предметну область, розглянули аналоги.

Другий розділ присвячений моделюванню та конструюванню програмного забезпечення.

У третьому розділі було проведено аналіз засобів тестування та тестування веб-застосунку.

Четвертий розділ описує розгортання програмного забезпечення та його підтримку на платформі Render.

КЛЮЧОВІ СЛОВА: СОЦІАЛЬНА МЕРЕЖА, ДРУЗІ, СПІЛКУВАННЯ, ЧАТ, ВІДЕОДЗВІНОК, БАЗА ДАНИХ.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 79 tables, 21 figures and 15 sources – in total 84 pages.

The diploma project aims to enhance the development of a social network that enables users to search for like-minded individuals based on specific parameters and communicate with them without relying on external tools.

Objective: To improve the method of finding friends based on interests and facilitate communication among them through a user-friendly built-in chat feature with audio and video calling capabilities.

Research Object: Communication and friend-search software.

Research Subject: The software development process.

Chapter 1 presents general principles, analyzes and describes the subject area, and examines similar systems.

Chapter 2 focuses on software modeling and construction.

Chapter 3 includes an analysis of testing tools and the testing of the web application.

Chapter 4 describes the deployment of the software and its support on the Render platform.

KEYWORDS: SOCIAL NETWORK, FRIENDS, COMMUNICATION, CHAT, VIDEO CALL, DATABASE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-9226.045440.01.91	Технічне завдання	25	
3	А4	КПІ.ІП-9226.045440.02.81	Пояснювальна записка	84	
4	А4	КПІ.ІП-9226.045440.03.12	Текст програми	91	
5	А4	КПІ.ІП-9226.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-9226.045440.05.34	Керівництво користувача	18	
7	А3	КПІ.ІП-9226.045440.06.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІП-9226.045440.07.99	Схема бази даних	1	
9	А3	КПІ.ІП-9226.045440.08.99	Креслення вигляду екранних форм	1	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-9226.045440.00.90		
Розроб.		Харенко О.С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Зенів І.О.					
Н.контр.		Ліщук К.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-92	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

СОЦІАЛЬНА МЕРЕЖА ДЛЯ ПОШУКУ ДРУЗІВ ТА СПІЛКУВАННЯ

Технічне завдання

КПІ.ІП-9226.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_Ірина ЗЕНІВ___

Нормоконтроль:
Катерина ЛІЩУК

Виконавець:
Олексій ХАРЕНКО

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Соціальна мережа для пошуку друзів та спілкування.

Галузь застосування:

Наведене технічне завдання поширюється на розробку соціальної мережі для пошуку друзів та спілкування КПІ.ІП-9226.045440, котра використовується для пошуку друзів та спілкування з ними у режимі реального часу та призначена для покращення комунікації між людьми.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки соціальної мережі для пошуку друзів та спілкування є завдання на дипломне проектування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для звичайних користувачів, що бажають знайти друзів та спілкуватись з ними у режимі реального часу, без використання сторонніх застосунків.

Метою розробки є покращення способу пошуку друзів за інтересами та спрощення комунікації між ними завдяки зручному вбудованому чату з функцією аудіо та відео дзвінка.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Вимоги до користувацького інтерфейсу

– при натисканні посилання “SignIn” на екрані Home, відбувається перехід до екрану SignIn1;

– після вводу даних для входу (екран SignIn2) та натисканні кнопки “Sign In”, відбувається перехід на екран Profile;

– у разі відсутності облікового запису, після натисканні на посилання “SignUp” на екрані Home, відбувається перехід на екран SignUp1;

– після вводу даних для реєстрації облікового запису (екран SignUp2) та натисканні кнопки “Sign Up”, відбувається перехід на екран SignIn1.

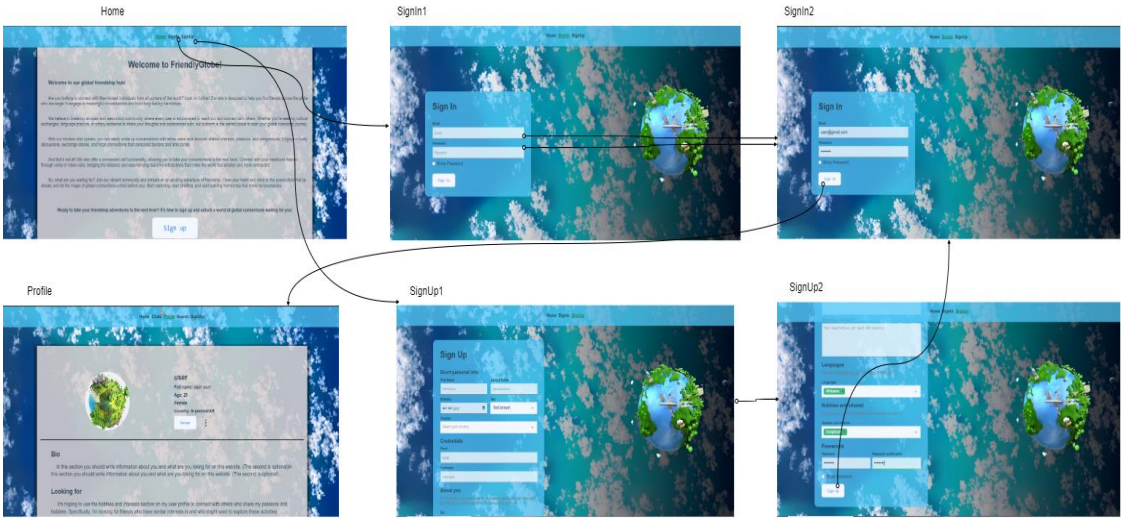


Рисунок 4.1 – Прототип дій неавторизованого користувача

– при натисканні кнопку “Manage” на екрані Profile, відбувається перехід до екрану Update Profile;

– після натискання перемикача режиму прихованості (екран Update Profile), з’являється повідомлення, що режим увімкнений (екран Toggle Hidden Mode);

– після натискання на посилання “Blacklist”, відбувається перехід на сторінку BlackList;

– після видалення останнього користувача з чорного списку буде продемонстровано напис про те, що список пустий (екран Blacklist Empty);

– після натискання на кнопку “Update” на екрані Update Data Profile чи натискання на кнопку “Delete” на екрані Update Password and Delete Account, відбувається перехід на сторінку авторизації;

– натискання на кнопку “Delete” на екрані Update Password and Delete Account, поле паролю очищається.

Рисунок 4.2 – Прототип дій користувача по керуванню обліковим записом

– при переході на сторінку пошуку демонструються 10 людей, що зареєструвались нещодавно (екран Search);

– після введення потрібних фільтрів та натискання кнопки “Filter” демонструється відфільтрований список користувачів (екран Search Filtred);

– після натискання на посилання з логіном користувача, відбувається перехід на сторінку User`s Profile;

– після натискання на кнопку “Send message” на екрані User`s Profile відбудеться перехід на екран New Chat, якщо користувачі не мають спільного чату, в іншому випадку відбудеться перехід на екран Chat Exist;

– після натискання на кнопку з трьома вертикальними крапочками на екрані New Chat чи Chat Exist, відбувається перехід на екран Manage Chat.

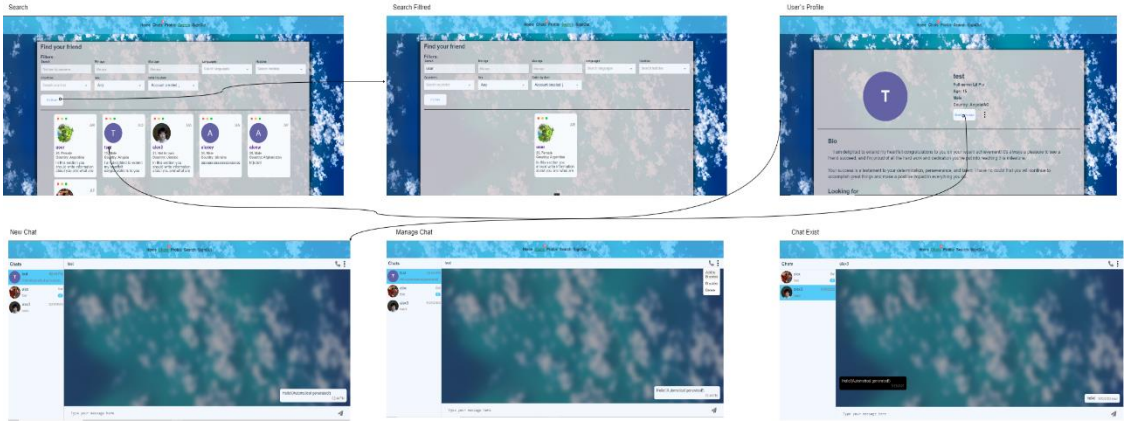


Рисунок 4.3 – Прототип дій користувача з пошуку друзів та створення чату

– якщо обидва учасники чату онлайн, то після натискання на кнопку дзвінкв на екрані Initiator Screen Start відбудеться перехід на екран Initiator Screen Call Initiated. У отримувача, який знаходився на екрані Receiver Screen Start відбудеться перехід на екран Receiver Screen Call Initiated;

– після натискання кнопки підтвердження дзвінка відбудеться перехід на екран Receiver Screen Call In Progress у отримувача, та на Initiator Screen Call In Progress у ініціатора розмови;

– після натискання кнопки відхилення дзвінка відбудеться перехід на екран Receiver Screen Start у отримувача, та на Call Rejected Screen у ініціатора розмови;

– після натискання кнопки завершення дзвінка відбудеться перехід на екран Receiver Screen Start чи Initiator Screen Start в залежності від того хто завершив розмову. Інший учасник розмови буде направлений на екран Call Ended Screen.

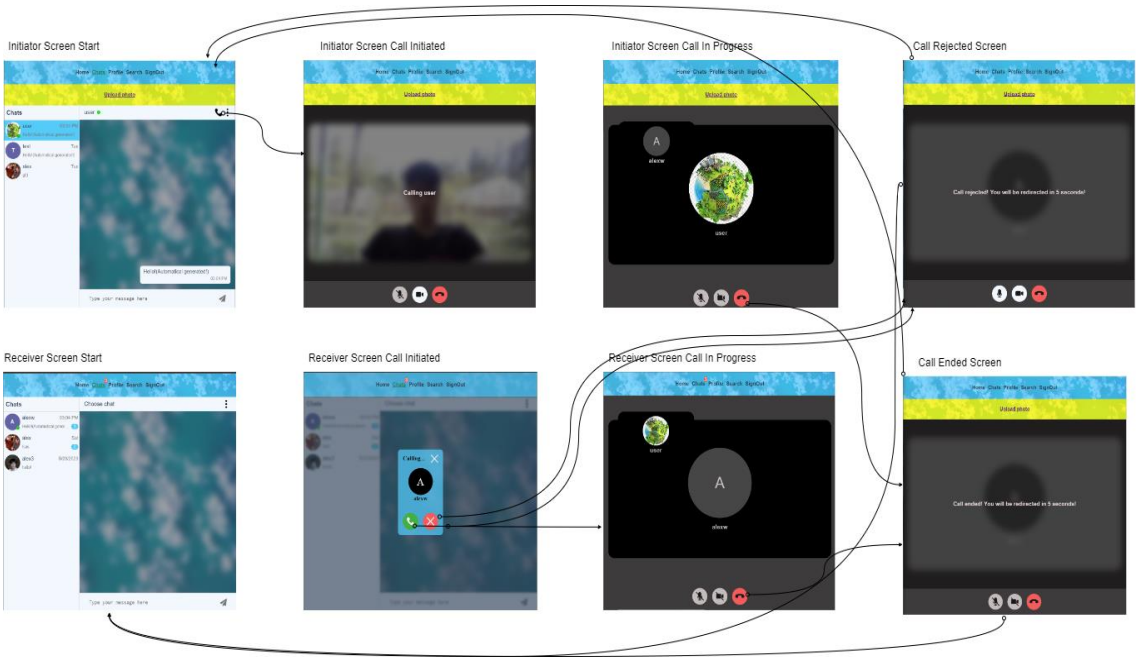


Рисунок 4.4 – Прототип дій користувача під час дзвінка

– після введення необхідних фільтрів на екрані Admin та натискання кнопки “Filter” відбудеться перехід на екран Admin Filtred;

– після натискання кнопки “Verify” відбудеться перехід на екран Admin Verified;

– після натискання кнопки “Block” на екрані Admin відбудеться перехід на екран Admin Block1;

– після введення причини блокування та натискання кнопки “Block” на екрані Admin Block2 відбудеться перехід на екран Admin Block3.

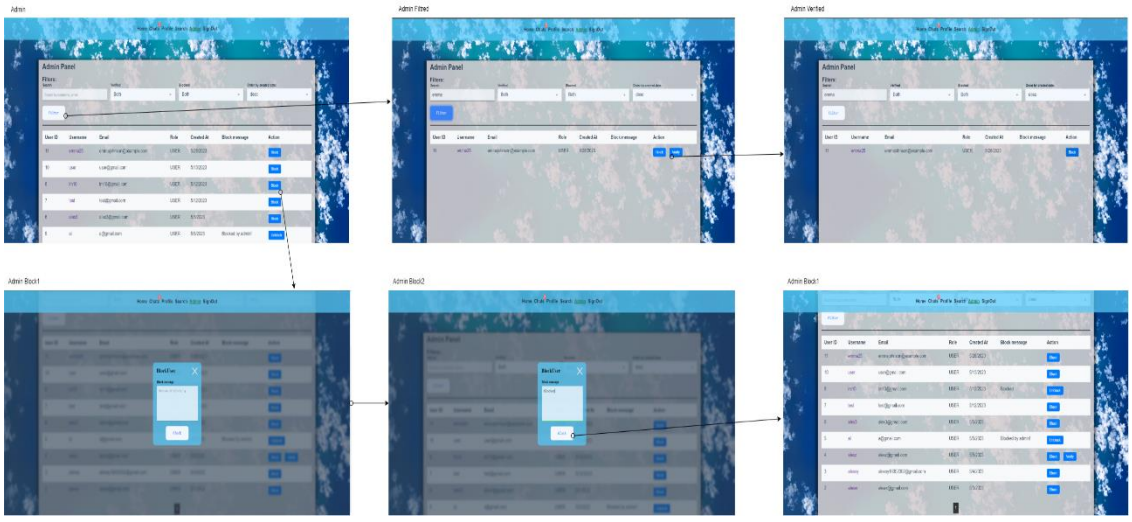


Рисунок 4.5 – Прототип дій адміністратора

4.1.2 Вимоги до користувача

4.1.2.1 Забезпечити можливість реєстрації користувача

4.1.2.1.1 Забезпечити можливість введення імені

4.1.2.1.1.1 Мінімальна кількість символів повинна дорівнювати 2

4.1.2.1.1.2 Максимальна кількість символів не повинна перевищувати 15

4.1.2.1.1.3 Поле не може бути порожнім

4.1.2.1.2 Забезпечити можливість введення прізвища

4.1.2.1.2.1 Мінімальна кількість символів повинна дорівнювати 2

4.1.2.1.2.2 Максимальна кількість символів не повинна перевищувати 15

4.1.2.1.2.3 Поле не може бути порожнім

4.1.2.1.3 Забезпечити можливість введення дати народження

4.1.2.1.3.1 Мінімальна значення 12 років

4.1.2.1.3.2 Максимальне значення 120 років

4.1.2.1.4 Забезпечити можливість вибору статі зі списку

4.1.2.1.4.1 Поле не може бути порожнім

4.1.2.1.4.2 Можна вибрати тільки одне значення

4.1.2.1.5 Забезпечити можливість вибору країни зі списку

4.1.2.1.5.1 Поле не може бути порожнім

4.1.2.1.5.2 Можна вибрати тільки одне значення

4.1.2.1.6 Забезпечити можливість введення електронної адреси

4.1.2.1.6.1 Поле не може бути порожнім

4.1.2.1.6.2 Поле повинно відповідати патерну

4.1.2.1.6.3 Перевірка унікальності логіну

4.1.2.1.7 Забезпечити можливість введення логіну користувача

4.1.2.1.7.1 Поле не може бути порожнім

4.1.2.1.7.2 Мінімальна кількість символів повинна дорівнювати 2

4.1.2.1.7.3 Максимальна кількість символів не повинна перевищувати 15

4.1.2.1.7.4 Перевірка унікальності логіну

4.1.2.1.8 Забезпечити можливість введення інформації про користувача

4.1.2.1.8.1 Мінімальна кількість символів повинна дорівнювати 100

4.1.2.1.8.2 Максимальна кількість символів не повинна перевищувати 1000

4.1.2.1.8.3 Поле не може бути порожнім

4.1.2.1.9 Забезпечити можливість введення причини реєстрації у веб-застосунку

4.1.2.1.9.1 Мінімальна кількість символів повинна дорівнювати 100

4.1.2.1.9.2 Максимальна кількість символів не повинна перевищувати 1000

4.1.2.1.9.3 Поле не може бути порожнім

4.1.2.1.10 Надати можливість користувачу обрати мови зі списку

4.1.2.1.10.1 Надати можливість вибору декількох мов

4.1.2.1.10.2 Надати можливість пошуку мови у списку

4.1.2.1.10.3 Список мов користувача не може бути порожнім

4.1.2.1.11 Надати можливість користувачу обрати хобі зі списку

4.1.2.1.11.1 Надати можливість вибору декількох хобі

4.1.2.1.11.2 Надати можливість пошуку хобі у списку

4.1.2.1.11.3 Список хобі користувача не може бути порожнім

4.1.2.1.12 Надати можливість користувачу введення паролю

4.1.2.1.12.1 Мінімальна кількість символів повинна дорівнювати 6

4.1.2.1.12.2 Максимальна кількість символів не повинна перевищувати 18

4.1.2.1.12.3 Поле не може бути порожнім

4.1.2.1.12.4 Застосувати шифрування

4.1.2.1.13 Надати можливість користувачу введення повторного паролю

4.1.2.1.13.1 Мінімальна кількість символів повинна дорівнювати 6

4.1.2.1.13.2 Максимальна кількість символів не повинна перевищувати 18

4.1.2.1.13.3 Поле не може бути порожнім

4.1.2.1.13.4 Повинен співпадати з введеним паролем

4.1.2.1.14 У разі введення некоректних даних кнопка не активна

4.1.2.1.15 У разі введення некоректних даних біля поля з’являється помилка

4.1.2.2 Надати можливість користувачу увійти у систему

4.1.2.2.1 Надати можливість введення електронної адреси

4.1.2.2.1.1 Поле не може бути порожнім

4.1.2.2.1.2 Поле повинно відповідати патерну

4.1.2.2.2 Надати можливість введення паролю

4.1.2.2.2.1 Мінімальна кількість символів повинна дорівнювати 6

4.1.2.2.2.2 Максимальна кількість символів не повинна перевищувати 18

4.1.2.2.2.3 Поле не може бути порожнім

4.1.2.2.3 Пошук користувача за введеною електронною поштою

4.1.2.2.3.1 Вивести повідомлення у разі введення некоректних даних

4.1.2.2.4 Порівняння введеного пароля та пароля знайденого користувача

4.1.2.2.4.1 Вивести повідомлення у разі введення некоректних даних

4.1.2.2.5 Перевірити чи користувач заблокований

4.1.2.2.5.1 Вивести повідомлення, якщо користувач заблокований

4.1.2.2.5.2 Не надати доступ користувачу до системи, якщо користувач заблокований

4.1.2.2.6 У разі введення некоректних даних кнопка не активна

4.1.2.3 Надати користувачу можливість редагувати інформацію про себе

4.1.2.3.1 Забезпечити можливість редагування імені

4.1.2.3.1.1 Поле повинно містити актульне ім’я користувача

4.1.2.3.1.2 Мінімальна кількість символів повинна дорівнювати 2

4.1.2.3.1.3 Максимальна кількість символів не повинна перевищувати 15

4.1.2.3.1.4 Поле не може бути порожнім

4.1.2.3.2 Забезпечити можливість редагування прізвища

4.1.2.3.2.1 Поле повинно містити актуальне прізвище користувача

4.1.2.3.2.2 Мінімальна кількість символів повинна дорівнювати 2

4.1.2.3.2.3 Максимальна кількість символів не повинна перевищувати 15

4.1.2.3.2.4 Поле не може бути порожнім

4.1.2.3.3 Забезпечити можливість редагування дати народження

4.1.2.3.3.1 Поле повинно містити актуальну дату народження користувача

4.1.2.3.3.2 Мінімальна значення 12 років

4.1.2.3.3.3 Максимальне значення 120 років

4.1.2.3.4 Забезпечити можливість вибору статі зі списку

4.1.2.3.4.1 Поле повинно містити актуальну стать користувача

4.1.2.3.4.2 Поле не може бути порожнім

4.1.2.3.4.3 Можна вибрати тільки одне значення

4.1.2.3.5 Забезпечити можливість вибору країни зі списку

4.1.2.3.5.1 Поле повинно містити актуальну країну користувача

4.1.2.3.5.2 Поле не може бути порожнім

4.1.2.3.5.3 Можна вибрати тільки одне значення

4.1.2.3.6 Забезпечити можливість редагування інформації про користувача

4.1.2.3.6.1 Поле повинно містити актуальну інформацію про користувача

4.1.2.3.6.2 Мінімальна кількість символів повинна дорівнювати 100

4.1.2.3.6.3 Максимальна кількість символів не повинна перевищувати 1000

4.1.2.3.6.4 Поле не може бути порожнім

4.1.2.3.7 Забезпечити можливість редагування причини реєстрації у веб-застосунку

4.1.2.3.7.1 Поле повинно містити актуальну інформацію про користувача

4.1.2.3.7.2 Мінімальна кількість символів повинна дорівнювати 100

4.1.2.3.7.3 Максимальна кількість символів не повинна перевищувати 1000

4.1.2.3.7.4 Поле не може бути порожнім

4.1.2.3.8 Надати можливість користувачу редагувати список мов:

4.1.2.3.8.1 Мови повинні бути обрані зі списку

4.1.2.3.8.2 Поле повинно містити обрані користувачем мови раніше

4.1.2.3.8.3 Надати можливість вибору декількох мов

4.1.2.3.8.4 Надати можливість пошуку мови у списку

4.1.2.3.8.5 Список мов користувача не може бути порожнім

4.1.2.3.9 Надати можливість користувачу редагувати список хобі:

4.1.2.3.9.1 Хобі повинні бути обрані зі списку

4.1.2.3.9.2 Поле повинно містити обрані користувачем хобі раніше

4.1.2.3.9.3 Надати можливість вибору декількох хобі

4.1.2.3.9.4 Надати можливість пошуку хобі у списку

4.1.2.3.9.5 Список хобі користувача не може бути порожнім

4.1.2.3.10 Встановити користувача не верифікованим у разі успіху

4.1.2.3.11 У разі введення некоректних даних кнопка не активна

4.1.2.3.12 У разі введення некоректних даних біля поля з’являється помилка

4.1.2.4 Надати користувачу можливість змінити пароль

4.1.2.4.1 Надати можливість користувачу введення паролю

4.1.2.4.1.1 Мінімальна кількість символів повинна дорівнювати 6

4.1.2.4.1.2 Максимальна кількість символів не повинна перевищувати 18

4.1.2.4.1.3 Поле не може бути порожнім

4.1.2.4.1.4 Застосувати шифрування

4.1.2.4.2 Надати можливість користувачу введення повторного паролю

4.1.2.4.2.1 Мінімальна кількість символів повинна дорівнювати 6

4.1.2.4.2.2 Максимальна кількість символів не повинна перевищувати 18

4.1.2.4.2.3 Поле не може бути порожнім

4.1.2.4.2.4 Повинен співпадати з введеним паролем

4.1.2.4.3 У разі введення некоректних даних кнопка не активна

4.1.2.4.4 У разі введення некоректних даних біля поля з’являється помилка

4.1.2.5 Надати користувачу можливість видалити обліковий запис

4.1.2.5.1 Продемонструвати вікно з вибором

4.1.2.5.1.1 Надати можливість підтвердити видалення

4.1.2.5.1.2 Надати можливість скасувати видалення

4.1.2.6 Надати користувачу можливість керувати режимом прихованості

4.1.2.6.1 Надати можливість користувачу ввімкнути режим, якщо він вимкнений

4.1.2.6.1.1 Продемонструвати повідомлення про відповідний режим

4.1.2.6.1.2 Приховати користувача з результатів пошуку.

4.1.2.6.2 Надати користувачу вимкнути режим, якщо він ввімкнений

4.1.2.6.2.1 Додати користувача до результатів пошуку

4.1.2.7 Надати користувачу можливість керувати фото профілю

4.1.2.7.1 Надати можливість додати фото профілю

4.1.2.7.1.1 Встановити користувача не верифікованим.

4.1.2.7.1.2 Зберегти фото

4.1.2.7.2 Надати можливість користувачу видалити фото профілю

4.1.2.7.3 Надати користувачу змінити фото профілю

4.1.2.7.3.1 Продемонструвати користувачу старе фото

4.1.2.7.3.2 Надати користувачу додати нове фото профілю

4.1.2.7.3.3 Встановити користувача не верифікованим

4.1.2.7.3.4 Зберегти нове фото

4.1.2.7.3.5 Видалити старе фото

4.1.2.8 Надати користувачу можливість пошуку інших користувачів цієї мережі

4.1.2.8.1 Надати можливість фільтрації користувачів

4.1.2.8.1.1 Надати можливість ввести логін користувача

4.1.2.8.1.2 Надати можливість обрати країни зі списку, що цікавлять користувача

4.1.2.8.1.3 Надати можливість обрати хобі зі списку, що цікавлять користувача

4.1.2.8.1.4 Надати можливість вказати мінімальний вік користувача

4.1.2.8.1.4.1 Заборонити введення від’ємних чисел

4.1.2.8.1.4.2 Введення тільки числових значень

4.1.2.8.1.5 Надати можливість вказати максимальний вік користувача

4.1.2.8.1.5.1 Заборонити введення від’ємних чисел

4.1.2.8.1.5.2 Введення тільки числових значень

4.1.2.8.1.6 Надати можливість обрати стать користувача зі списку

4.1.2.8.2 Надати користувачу можливість відсортувати користувачів за датою створення аккаунту

4.1.2.9 Надати користувачу можливість передивитись інформацію про іншого користувача

4.1.2.10 Надати користувачу можливість керувати чорним списком

4.1.2.10.1 Надати можливість користувачу додати іншого користувача у чорний список

4.1.2.10.1.1 Прибрати доданого користувача з результатів пошуку власника списку

4.1.2.10.1.2 Прибрати у доданого користувача з результатів пошуку власника списку

4.1.2.10.1.3 Видалити всі повідомлення з між користувачем та доданим користувачем до чорного списку

4.1.2.10.1.4 Видалити чат між користувачем та доданим користувачем

4.1.2.10.1.5 Прибрати можливість створити новий чат з власником списку

4.1.2.10.1.6 Приховати користувачу сторінку заблокованого користувача

4.1.2.10.1.7 Приховати заблокованому користувачу сторінку власника списку

4.1.2.10.2 Надати користувачу можливість прибрати іншого користувача з чорного списку:

4.1.2.10.2.1 Додати розблокованого користувача до результатів пошуку власника списку

4.1.2.10.2.2 Додати у розблокованого користувача до результатів пошуку власника списку

4.1.2.10.2.3 Надати можливість користувачу створити новий чат з власником списку

4.1.2.10.2.4 Приховати користувачу сторінку заблокованого користувача

4.1.2.10.2.5 Приховати заблокованому користувачу сторінку власника списку

4.1.2.11 Надати користувачу можливість створити новий чат з іншим користувачем

4.1.2.11.1 Додати автоматично згенероване повідомлення до чату

4.1.2.12 Надати можливість користувачу керувати повідомленнями в чаті

4.1.2.12.1 Надати можливість користувачу надсилати нові повідомлення

4.1.2.12.1.1 Поле не повинно бути пустим

4.1.2.12.1.2 Якщо отримувач знаходиться в мережі, то йому прийде це повідомлення

4.1.2.12.2 Надати можливість користувачу редагувати власні повідомлення

4.1.2.12.2.1 Поле повинно містити текст повідомлення

4.1.2.12.2.2 Поле не повинно бути пустим

4.1.2.12.2.3 Якщо отримувач знаходиться в мережі, то йому прийде відредаговане повідомлення

4.1.2.12.2.4 Повідомлення буде помічене відредагованим та буде встановлена час редагування

4.1.2.12.3 Надати можливість користувачу видаляти власні повідомлення

4.1.2.12.3.1 Якщо отримувач знаходиться в мережі, то повідомлення буде прибране і в нього

4.1.2.13 Надати користувачу отримати список чатів

4.1.2.14 Надати користувачу можливість керувати дзвінком

4.1.2.14.1 Надати користувачу можливість керувати камерою

4.1.2.14.1.1 Надати можливість ввімкнути камеру

4.1.2.14.1.2 Надати можливість вимкнути камеру

4.1.2.14.2 Надати користувачу керувати мікрофоном

4.1.2.14.2.1 Надати можливість вимкнути мікрофон

4.1.2.14.2.2 Надати можливість ввімкнути мікрофон

4.1.2.14.3 Надати користувачу можливість завершити дзвінок

4.1.2.14.4 Надати користувачу здійснити дзвінок

4.1.2.14.4.1 Співрозмовник повинен бути в мережі

4.1.2.14.4.2 Співрозмовник не може бути заблокованим

4.1.2.14.4.3 Співрозмовник та користувач не можуть бути в чорних списках один одного

4.1.2.14.5 Надати можливість користувачу відповісти на дзвінок

4.1.2.14.6 Надати користувачу відхилити дзвінок

4.1.2.14.7 Завершити дзвінок, якщо за 1 хвилину інший користувач не відповів

4.1.3 Вимоги до адміністратора системи:

4.1.3.1 Надати адміністратору можливість перейти на профіль користувача

4.1.3.1.1 Надати можливість перейти, якщо користувач заблокований

4.1.3.1.2 Надати можливість перейти, якщо користувач у режимі прихованості

4.1.3.1.3 Надати можливість перейти, якщо користувач не верифікований

4.1.3.2 Надати адміністратору можливість заблокувати користувача

4.1.3.2.1 Надати можливість адміністратору залишити причину блокування користувача

4.1.3.2.2 Видалити всі чати, що відносяться до цього користувача

4.1.3.2.3 Видалити всі повідомлення, що відносяться до цього користувача

4.1.3.2.4 Обмежити доступ до системи

4.1.3.2.5 Прибрати з результатів пошуку користувачів

4.1.3.2.6 Заборонити створення чатів з заблокованим користувачем

4.1.3.3 Надати адміністратору можливість веривікувати користувача

4.1.3.4 Надати адміністратору можливість отримати всіх користувачів системи

4.1.3.4.1 Надати можливість пошуку користувачів за фільтрами

4.1.3.4.1.1 Фільтрація за статусом верифікації

4.1.3.4.1.2 Фільтрація за блокуванням

4.1.3.4.1.3 Надати можливість пошуку за логіном та електронною адресою

4.1.3.4.2 Надати можливість відсортувати за датою створення облікових записів

4.1.3.5 Надати адміністратору можливість розблокувати користувача

4.1.3.5.1 Видалити причину блокування користувача

4.1.3.5.2 Додати користувача до результатів пошуку

4.1.3.5.3 Додати можливість створення з ним чатів

4.1.4 Додаткові вимоги

– веб-застосунок повинен бути простим у використані та мати приємний дизайн.

– веб-застосунок повинен бути адаптованим до мобільних пристроїв.

– веб-застосунок повинен забезпечувати стабільну роботу на наступних браузерах:

1) Google Chrome версія 88.0.4324.190 і вище; о

2) Microsoft Edge версія 95.0.1020.44 і вище;

4.2 Вимоги до надійності

Дані, які вводить користувач, мають спочатку перевірятись на відповідність на клієнтській стороні. Якщо до прикладу введена адреса не задовольняє умову, перевірки на коректність – вивести відповідне повідомлення. Забезпечити додаткову перевірку на серверній стороні

додатку, для уникнення запису не коректних даних. Перевірки сервера також виводити в повідомленні для користувача.

Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Потрібен адміністратор, що буде перевіряти профілі, керувати блокуванням користувачів.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 5 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 30 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням усіх операційних систем, які підтримують підключення до мережі інтернет, та можуть надати доступ до веб-застосунку через браузер.

4.5.1 Вимоги до вхідних даних

Зображення користувача повинно мати розширення jpeg або png.

4.5.2 Вимоги до вихідних даних

Особливі вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування JavaScript, Node.js.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища VS Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді відкритих текстових файлів TypeScript (.ts), JavaScript(.js) та Vue (.vue).

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Програмне забезпечення повинно доступне для публічного доступу.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема бази даних;
– архітектура програмного забезпечення;

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.03	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.03	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.04	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.04	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.04	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.04	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.04	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проекту

на тему: Соціальна мережа для пошуку друзів та спілкування

КПІ.ІП-9226.045440.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

BPMN	–	Business process model and notation.
SQL	–	Structured query language.
IT	–	Інформаційні технології.
СУБД	–	Система управління базами даних.
SEO	–	Search engine optimization.
БД	–	База даних.
SPA	–	Single page application.

ВСТУП

Протягом усієї історії, людина була завжди соціальним створінням, яка потребувала спілкування. У сучасну цифрову епоху, соціальні мережі зробили революцію та дозволили користувачам обмінюватись інформацією незважаючи на географічні кордони. Однак у міру зростання популярності основних платформ, таких як Facebook, стає все важче знайти однодумця в інтернеті. Користувачі таких мереж просто не розуміють, чого хоче зареєстрована там людина.

Тож ми прагнемо розробити застосунок, що зосередиться на об’єднанні користувачів, що поділяють спільні інтереси та цінності. Наддасть можливість створити дружні стосунки на основі спілкування. Крім того, спілкування у реальному часі лежить в основі людської комунікації. Від жвавих розмов до спонтанного сміху, це те, чого часто не вистачає під час текстової взаємодії. Тож завдяки зручному пошуку та розширеним комунікаційним засобам, таким як функції чату у реальному часі та відеодзвінки, планується створити середовище, яке дозволить знаходити справжню дружбу і покращувати її.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Веб-застосунок (веб-додаток) - це програмне забезпечення, яке не встановлюється додатково на пристрій. Веб-додаток можна відкрити за допомогою у будь-якому браузері [1]. Для написання клієнтської частини використовують HTML, CSS, JavaScript. Node.js, Python та інші мови програмування широко використовуються для створення бекенд частини. Також існує багато фреймворків для, які спрощують розробку зовнішнього інтерфейсу та серверної частини.

Front-end (клієнтська частина) – це частина веб-застосунку, що являє собою користувацький інтерфейс. Він дозволяє користувачу взаємодіяти з усім, що він бачить [2].

Back-end (серверна частина) – це програмно-апаратна частина [2], яка в цілому працює з даним.

Браузер – це програмне забезпечення, що дозволяє якось взаємодіяти з інформацією в World Wide Web, включаючи веб-сторінки, відео, аудіо та фото [3].

1.2 Змістовний опис і аналіз предметної області

Соціальна мережа для пошуку друзів та спілкування полягає у надані користувачеві великої бази однодумців, які відкриті до нових знайомств, зі зручним способом комунікації у реальному часі.

В системі було вирішено зробити декілька ролей, для відокремлення звичайних користувачів від адміністраторів, що мають перевіряти нові облікові записи на валідність та блокувати користувачів.

Зручність використання веб-застосунку для пошуку друзів полягає в:

- широкій базі однодумців відкритих до нових знайомств;
- зрозумілому та приємному для користувачів інтерфейсі;
- зручному способі комунікації у реальному часі;

– налаштуванні свого облікового запису;

– можливості здійснювати відео та аудіо дзвінки без використання сторонніх застосунків;

– можливості обмежити доступ до свого профілю;

– пошуку людей зі спільними інтересами за різними параметрами.

В свою чергу, зручність використання з боку адміністратора полягає в наявності панелі адміністрування з наступними можливостями:

– пошук користувачів за певними фільтрами;

– керування профілям користувачів (блокування, перевірка коректності облікового запису).

1.3 Аналіз існуючих технологій та успішних ІТ-проектів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації соціальної мережі для пошуку друзів та спілкування. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

В даній роботі немає спеціальних алгоритмічних рішень, а є тільки загальні. Тому необхідність їх аналізу в нашій розробці відсутня.

Для написання веб-застосунку було вирішено розглянути монолітне, мікросервісне та клієнт-серверне архітектурні рішення.

Монолітна архітектура представляє собою одну велику обчислювальну мережу, яка містить в собі інтерфейс клієнта, бізнес логіку сервера та базу даних. Все обслуговується в одному окремому додатку. Зазвичай такі програми не мають модульності, тому коли розробники хочуть щось змінити чи оновити, то вони працюють зі всім застосунком. Перевагами такої архітектури є те, що таке програмне забезпечення легше розгорнути та розробляти. Але великі проекти роблять розробку повільною через великий

обсяг коду. Також, ще однією проблемою є масштабування, не так просто додавати щось нове, не змінивши вже існуючі сервіси.

Тепер розглянемо мікросервісну архітектуру. Вона представляє собою безліч окремих серверів, які мають власну бізнес логіку, базу даних та не мають доступу до інших сервісів. Таке рішення дозволяє краще масштабувати проект та легше підтримувати його, бо достатньо внести зміни в один окремий модуль, не чіпаючи інші. Але є і мінуси: складність розробки та ціна всієї інфраструктури, бо кожний сервіс – це окремий сервер з власною базою даних.

Перейдемо до клієнт-серверної архітектури. Вона дозволяє один розділити монолітний застосунок на два різних – клієнт та сервер. Ідея полягає в тому, що вони спілкуються між через мережу. Клієнт відповідає за представлення отриманої інформації, а сервер опрацьовує і надсилає її на відповідний запит. Переваги такого рішення полягають в швидкості, масштабуванні.

Існує три архітектурних рішення для написання клієнтської частини веб-додатків:

– SPA (single page application);
– SSR (server side rendering);
– SSG (static site generation).

SPA має структуру однієї сторінки HTML без попередньо завантаженого вмісту. Він завантажується через окремі JavaScript файли та розміщуються на одній сторінці. У цих файлах містяться всі дані, пов’язані з логікою програми, інтерфейсом користувача та зв’язком з сервером [4].

Така архітектура має свої плюси:

– зручний інтерфейс;
– швидкість завантаження;
– може використовуватись разом з іншими технологіями.

Але наявні і мінуси:

– з ростом проекту збільшується розмір та складність проекту, зменшується швидкість початкового завантаження, хоча цього можна запобігти динамічним імпортом потрібних сторінок;

– дуже складно адаптувати для SEO.

SSG має підхід відмінний від SPA застосунку. Генератори статичних сайтів продукують вміст під час створення нових сторінок або при внесенні змін у нього. Такий підхід іноді надає приріст у швидкості, але є більш складним у розробці та подальшій підтримці.

SSR відправляє користувачу вже готову сторінку, не потрібно чекати на завантаження додаткового контенту. Візуалізація відбувається на сервері перед надсиланням у браузер. Коли клієнт робить запит сторінки, дані витягуються з бази даних, що сповільнює процес. Хоча деякі елементи можна кешувати [4]. Такий підхід приносить такі плюси:

– краща SEO оптимізація;

– не потрібно пере збирати веб-застосунок після внесення змін.

Хоча є ряд мінусів:

– потребують більше API викликів до сервера,

– є повільнішими ніж інші два варіанти.

Проаналізувавши деякі рішення, було обрано клієнт-серверну архітектуру та single page application технологію, що – забезпечить зручний користувацький інтерфейс та кращу швидкість роботи.

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Розглянемо мови програмування для написання серверної частини. Здебільшого використовують Node.js, Python, Go, Java, C#, Ruby та інші.

Node.js використовує так званий V8 двигун, розроблений Google, який компілює JavaScript в нативний машинний код. Ця мова програмування широко використовується у back-end розробці. Вона має детальну документацію, велике ком’юніті, активно підтримується, постійно додаються нові бібліотеки, яких і так вже досить багато. Node.js є асинхронною, що

дозволяє обробляти набагато більше запитів і потребує менше пам’яті, ніж аналогічна програма написана на іншій мові [5]. Також перевагою є те, що розробка back-end та front-end частин ведеться однією мовою. Це дуже спрощує процес.

Go або GoLang – був розроблений в один час з Node.js, але довгий час залишався не поміченим. Зараз досить багато професіоналів вагаються між Node.js та Go при виборі мови програмування. GoLang використовує синтаксис мови програмування C. Його розроблено, щоб зосередитися на безпеці, простоті та швидкості [6].

Python дуже часто використовується у новачків та є найповільнішим в трійці, яку ми розглядаємо. Його цінують за відносну простоту написання та невеликий об’єм коду. Має велике ком'юніті та багато різних бібліотек та фреймворків, що значно спрощують розробку. При виборі мови програмування для розробки back-end, надають перевагу швидким та зручним Node.js та Go.

Тепер, ще потрібно провести невеличкий аналіз бази даних. Існує два види SQL та NoSQL. SQL широко використовується для керуванням даними в реляційних системах баз даних, де інформація зберігається в рядках і таблицях, що якось зв’язні. Такі бази даних можуть обробляти дуже великі обсяги даних за короткий час. SQL бази даних добре масштабуються вертикально та горизонтально [7]. NoSQL – не реляційна база даних. Вона має іншу структуру ніж SQL DB. NoSQL дозволяє працювати з різними структурами даних тому, що використовує динамічну схему для неструктурованої інформації. Це надає можливість простіше вносити зміни, додавати нові атрибути та поля.[7] Не потрібно планувати точну схему даних наперед. Також NoSQL бази даних краще масштабуються вертикально, мають більшу швидкість запису та читання.

WebSocket – протокол зв’язку, що надає можливість двухсторонньої комунікації між клієнтом та сервером через TCP з’єднання [8]. Ця технологія

дозволяє реалізувати зручний спосіб спілкування у реальному часі між користувачами.

WebRTC – дозволяє браузерам напряму обмінюватись медіа файлами у режимі реального часу з іншими браузерами через безпечний доступ до периферійних пристроїв введення, таких як веб-камери та мікрофони [9]. Пристрої спілкуються без сервера між ними, що зменшує затримку. WebRTC вже є вбудованим в HTML 5, тож не потрібно встановлювати додаткові плагіни чи стороннє програмне забезпечення. Ця технологія використовує UDP протокол, який є більш швидким ніж TCP, але менш надійним.[9] Частіше за все, WebRTC та WebSockets використовують у парі. WebRTC використовують для роботи з медіа-даними, а окремий WebSocket сервер забезпечує обмін метаданими, щоб керувати з'єднанням.

Для написання клієнтської частини існує багато фреймворків та бібліотек, які спрощують розробку зовнішньої частини веб-застосунку. Найбільш популярними є React, Vue.js та AngularJS. Кожен з них має свої переваги та недоліки.

React – це JavaScript бібліотека, призначена для розробки графічного інтерфейсу. Одним з найбільших плюсів є те, що використовуючи майже такий код, можна написати мобільний застосунок завдяки фреймворку React Native [10].

Vue.js – це бібліотека, яка зосереджена на шарі View. Хоча його називають прогресивним фреймворком тому, що його функціональність можна розширити за допомогою офіційних і сторонніх пакетів, таких як Vue Router та Vuex. Синтаксис шаблонів Vue дозволяє створювати компоненти, поєднуючи HTML з спеціальними директивами та функціями [11]. Vue.js є більш швидким ніж аналоги та краще підходить для проектів, що масштабуються.

AngularJS – це фреймворк побудований на MVC партерні. Але новіша версія – Angular 2, вже основана на компонентній системі. Проекти мають чітку структуру – модулі, компоненти та сервіси. Кожен веб-застосунок має

хоча б один основний компонент та модуль. Angular потребує більше пам’яті в порівняні з іншими фреймворками [11].

Під нашу задачу написання серверної частини краще підходить мова програмування Node.js, тож потрібно дослідити та обрати фреймворк, що покращить розробку застосунку. NestJS, Express.js та Fastify є домінуючим на зараз.

Express – це гнучкий фреймворк, який надає зручний набір функцій для написання веб-додатків. В іншу чергу, Fastify є більш сфокусованим на покращенні швидкості. Він займає не великий обсяг пам’яті, використовує асинхронність щоб обробляти великі обсяги інформації, не блокуючи роботу застосунку [12].

NestJS – фреймворк, що побудований на принципах ООП. Він написаний на TypeScript, що дозволяє швидше вести розробку, адже є автоматичне виявлення помилок під час компіляції [13]. В основі своєї роботи, Nest використовує HTTP фреймворки, такі як Express та Fastify, що також впливає на зручність написання та швидкодію готового веб-застосунку.

PostgreSQL – реляційна база даних, яка має свої переваги над іншими СУБД. PostgresSQL реалізує багатоверсійний контроль паралельності (MVCC) без блокування читання, підтримує паралельні запити, які можуть використовувати декілька процесорів/ядер. Також він відомий тим, що захищає цілісність даних на рівні транзакції, це робить його менш вразливим до пошкоджень даних [14].

Для покращення роботи з базою даних слід використати ORM (Object Relational Mapping). Prisma дозволяє швидше формувати запити на Node.js без необхідності використання SQL. Це підвищує швидкість розробки та продуктивність у роботі з базами даних.

Для серверної частини краще підійде мова програмування Node.js, бо вона є швидкою, і потребує менше пам’яті. Реляційна база даних підходить для нашого проекту краще через зручність зберігання даних та відносно

швидку роботу системи. WebSocket та WebRTC підходять для написання функціоналу застосунку, що потребує спілкування у режимі реального часу між сервером та клієнтом.

Для спрощення написання серверної частини було обрано фреймворк Nest на основі Fastify, що є досить швидким та дає можливість зручніше вести розробку. Вирішено використати реляційну базу даних PostgreSQL та Prisma для зручності роботи з нею.

У якості основи клієнтської частини буде використано фреймворк Vue.js, який є швидким та чудово підходить під наші задачі.

Також для роботи з засобами комунікації буде використано WebSockets, що дозволить реалізувати потрібний функціонал.

1.3.3 Аналіз відомих програмних продуктів

На момент написання дипломної роботи було знайдено декілька аналогів. Для порівняння було обрано веб-застосунки:

– GlobalPenfriends.com;

– Bumble.

GlobalPenfriends.com – це веб-додаток, який володіє великою базою людей, що зацікавлені в пошуку друзів. Він був заснований у 2001 році, щоб зближувати людей по всьому світу, незважаючи ні на що. В безкоштовній версії ви можете написати тільки одній людині, чат не є зручним та відсутня можливість відео та аудіо дзвінків взагалі. Є підписка, що знімає обмеження на листування, редагування профілю, тощо.

Bumble – це веб-сайт, що дозволяє шукати людей по всьому світу, які мають бажання знайти собі однодумця чи другу половинку. Bumble був дуже популярним у 2021 році серед аналогічних застосунків у App Store. Там реалізовано базовий функціонал, але пошук доступний тільки за віком і за місцем де знаходитесь ви.

Для порівняння дипломної роботи з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Дипломний проект (Friendly Globe)	GlobalPenfriends.com	Bumble
Реєстрація	+	+	+
Вхід через інші соціальні мережі	-	-	+
Перевірка профілю адміністратором	+	+	+
Обмеження роботи з веб-застосунком	-	+	-
Демонстрація детальної інформації про користувача	+	+	+
Пошук за різними параметрами	+	+	-
Зручний спосіб комунікації у реальному часі	+	-	+
Можливість редагувати відправлені повідомлення	+	-	+
Можливість видалити відправлене повідомлення	+	+	+
Можливість робити відео та аудіо дзвінки	+	-	-
Можливість додати чи змінити фото профілю	+	+	+
Сповіщення через електрону адресу	-	+	-
Можливість приховати свій обліковий запис	+	+	-

Продовження таблиці 1.1

Повне видалення облікового запису	+	+	+
Платні послуги або підписки	-	+	+
Можливість заблокувати користувача	+	+	+

1.4 Аналіз вимог до програмного забезпечення

Головною функцією програмного забезпечення є пошук друзів за різними параметрами та комунікація між користувачами у режимі реального часу, більше функцій можна побачити на рисунку 1.1.

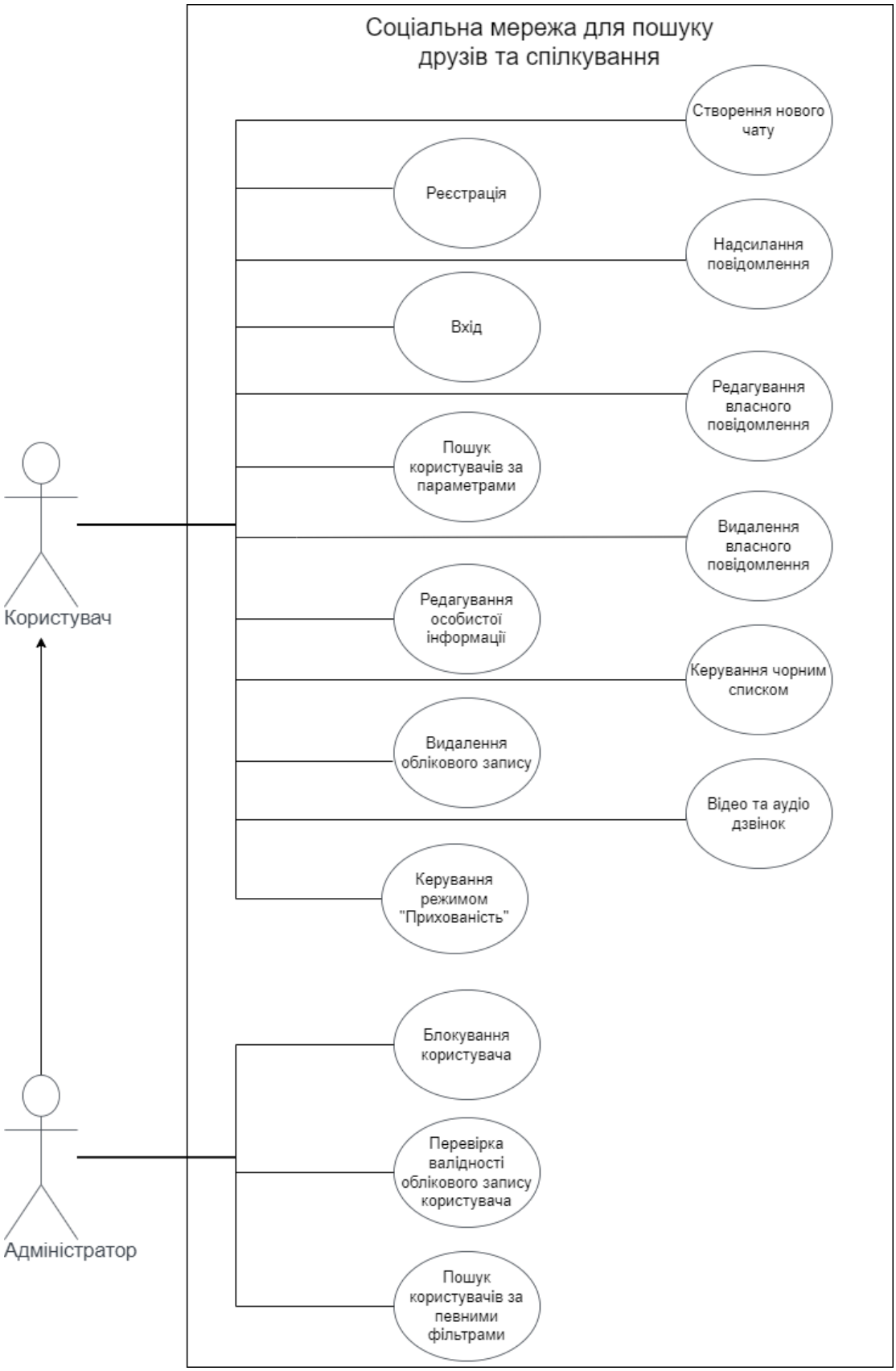


Рисунок 1.1 – Діаграма варіантів використання

В таблицях 1.2 - 1.17 наведені варіанти використання програмного забезпечення.

Таблиця 1.2 - Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	Не зареєстрований
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: – ім’я; – прізвище; – username; – пошта користувача; – інформацію про себе; – мету реєстрації (на що він очікує) – день народження; – пароль в системі; – повтор паролю для підтвердження. Також обирає країну проживання, обирає теми, що цікавлять користувача, мови, якими йому зручно розмовляти. Вводить мету своєї реєстрації. Після заповнення даних користувача натискає кнопку реєстрації. Після успішної реєстрації, користувач перенаправляється на сторінку входу.
Extension	В випадку введення не коректних даних, користувач не зможе зареєструватись, доки не будуть виправлені дані. Якщо поле введено некоректно, то воно буде виділеним.
Post-Condition	Користувач зареєстрований, перехід на сторінку входу

Таблиця 1.3 - Варіант використання UC-2

Use case name	Вхід користувача
Use case ID	UC-02
Goals	Вхід користувача в систему
Actors	Користувач
Trigger	Користувач хоче увійти в систему
Pre-conditions	Користувач зареєструвався
Flow of Events	Користувач переходить на сторінку входу. В поля для входу вводяться відповідні дані: пошта користувача та пароль. Після заповнення даних користувача натискає кнопку входу. Після цього користувач перенаправляється на сторінку особистого облікового запису.
Extension	В випадку введення не коректних даних, кнопка входу стає неактивною. Буде виведене повідомлення про помилку.
Post-Condition	Користувач увійшов до системи.

Таблиця 1.4 - Варіант використання UC-3

Use case name	Пошук користувачів за параметрами
Use case ID	UC-03
Goals	Знайти користувачів в системі з певною характеристикою
Actors	Користувач
Trigger	Користувач хоче знайти собі однодумця з певними параметрами
Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований аккаунт.

Продовження таблиці 1.4

Flow of Events	Користувач переходить на сторінку пошуку. Йому надається можливість пошуку інших користувачів написавши username або обравши проміжок віку, місце проживання, інтереси, мови, які цікавлять. Після введення потрібних параметрів, користувач натискає на кнопку пошуку. Йому демонструється відфільтрований список користувачів.
Extension	Якщо користувач залишив поля пустими та натиснув кнопку пошуку, то йому буде продемонстрований список людей без фільтрації.
Post-Condition	Користувач знайшов людей за вказаними параметрами.

Таблиця 1.5 - Варіант використання UC-4

Use case name	Редагування особистої інформації
Use case ID	UC-04
Goals	Змінити дані користувача
Actors	Користувач
Trigger	Користувач хоче змінити особисті дані
Pre-conditions	Користувач увійшов в систему, не заблокований аккаунт.

Продовження таблиці 1.5

Flow of Events	Користувач переходить на сторінку налаштувань. Йому надається можливість змінити наступні дані: – ім’я; – прізвище; – день народження; – країну; – фото профілю; – теми, що його цікавлять; – мови; – мету реєстрації у веб-застосунку; – стать. Після внесення потрібних змін, користувач натискає на кнопку збереження. Обліковий запис стає не перевіреним і не відображається в пошуку, поки його не буде перевірено адміністратором.
Extension	В випадку введення не коректних даних, кнопка збереження стає неактивною. Якщо якесь конкретне поле введено некоректно, то воно буде виділеним.
Post-Condition	Користувач змінив дані про себе.

Таблиця 1.6 - Варіант використання UC-5

Use case name	Видалення облікового запису
Use case ID	UC-05
Goals	Видалити обліковий запис с системи
Actors	Користувач

Продовження таблиці 1.6

Trigger	Користувач хоче видалити свій обліковий запис
Pre-conditions	Користувач увійшов в систему.
Flow of Events	Користувач переходить на сторінку налаштувань. Натискає на кнопку видалення. З’являється вікно підтвердження з кнопками підтвердження та відмови. Обравши відповідний варіант, обліковий запис буде видалено. Також будуть видалені чати та повідомлення, які відносяться до користувача. Користувача перенаправляє на сторінку входу.
Extension	В випадку відмови від видалення даних, вікно закриється автоматично, інформація залишиться у веб-застосунку.
Post-Condition	Користувач видалив обліковий запис.

Таблиця 1.7 - Варіант використання UC-6

Use case name	Керування режимом «Прихованість»
Use case ID	UC-06
Goals	Додати себе до результатів пошуку чи виключити.
Actors	Користувач
Trigger	Користувач хоче прибрати себе з результатів пошуку чи повернути.
Pre-conditions	Користувач увійшов в систему.
Flow of Events	Користувач переходить на сторінку налаштувань. Натискає на кнопку перемикання режиму прихованості. Якщо режим був ввімкнений, то він вимкнеться і навпаки.
Extension	-
Post-Condition	Користувач потрапляє або не потрапляє до пошукових запитів.

Таблиця 1.8 - Варіант використання UC-7

Use case name	Створення нового чату
Use case ID	UC-07
Goals	Створити новий чат для спілкування
Actors	Користувач
Trigger	Користувач хоче почати діалог з іншою людиною
Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований аккаунт.
Flow of Events	Користувач переходить на сторінку облікового запису, що його цікавить. Натискає на кнопку «Надіслати повідомлення» та його перенаправляється на сторінку чату. Надсилається автоматично згенероване повідомлення обраному користувачу, чат буде збережено у системі.
Extension	Якщо обраний обліковий запис заблокований, в чорному списку чи знаходиться в режимі прихованість, то чат не буде створено. Якщо користувач вже спілкується з обраною людиною, при натисканні на кнопку «Написати», його буде перенаправлено на існуючий чат.
Post-Condition	Користувач має новий чат і може почати спілкування.

Таблиця 1.9 - Варіант використання UC-8

Use case name	Надсилання повідомлення
Use case ID	UC-8
Goals	Надіслати повідомлення
Actors	Користувач
Trigger	Користувач хоче надіслати повідомлення другові

Продовження таблиці 1.9

Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований аккаунт.
Flow of Events	Користувач переходить на сторінку чатів, обирає людину, кому хоче написати. Його перенаправляє на сторінку з обраним користувачем. Вводить текстове повідомлення та натискає на кнопку «Надіслати». Повідомлення з’явиться в історії спілкування. Отримувач зможе його прочитати.
Extension	У разі помилки, повідомлення не буде додане до історії спілкування.
Post-Condition	Користувач надіслав повідомлення.

Таблиця 1.10 - Варіант використання UC-9

Use case name	Редагування власного повідомлення
Use case ID	UC-9
Goals	Виправити надіслане повідомлення
Actors	Користувач
Trigger	Користувач хоче виправити надіслане повідомлення другові
Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований аккаунт. Є успішно надіслане повідомлення.
Flow of Events	Користувач переходить на сторінку чатів, обирає людину, кому хоче написати. Натискає на повідомлення, яке хоче змінити. З’являється нове вікно, обирає варіант змінити. Надається можливість змінити зміст повідомлення, та повторно надіслати. Повідомлення помічається відредагованим.

Продовження таблиці 1.10

Extension	У разі помилки, повідомлення не буде змінено та додатково поміченим.
Post-Condition	Користувач виправив повідомлення.

Таблиця 1.11 - Варіант використання UC-10

Use case name	Видалення власного повідомлення
Use case ID	UC-10
Goals	Видалити надіслане повідомлення
Actors	Користувач
Trigger	Користувач хоче видалити надіслане повідомлення другові
Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований акаунт. Є успішно надіслане повідомлення.
Flow of Events	Користувач переходить на сторінку чатів, обирає людину, кому хоче написати. Натискає на повідомлення, яке хоче видалити. З'являється нове вікно, обирає варіант видалити. Повідомлення зникає з історії чату для обох користувачів.
Extension	У разі помилки, повідомлення не буде видалено.
Post-Condition	Жоден з учасників чату не бачить видалене повідомлення і воно не збережене в системі.

Таблиця 1.12 - Варіант використання UC-11

Use case name	Керування чорним списком
Use case ID	UC-11
Goals	Обмежити або надати доступ до свого облікового запису певному користувачу
Actors	Користувач

Продовження таблиці 1.12

Trigger	Користувач хоче обмежити або надати доступ до свого облікового запису певному користувачу
Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований акаунт.
Flow of Events	Якщо користувач хоче обмежити доступ, то він переходить на сторінку чатів, обирає чат, що хоче заблокувати. Або він може перейти на сторінку профілю користувача, та обмежити доступ там. Натискає на кнопку блокування. Якщо чат існував з заблокованим користувачем, то він автоматично видаляється з системи з усіма повідомленнями, користувач потрапляє до чорного списку. Якщо користувач хоче надати доступ, то він переходить на сторінку чорного списку та натискає кнопку прибрати.
Extension	-
Post-Condition	Якщо користувач обмежив доступ, чат не доступний. Заблокований співрозмовник не може написати користувачу або знайти його. З’явився відповідний запис на сторінці чорного списку. В іншому випадку, розблокований користувач може почати спілкування, запис з чорного списку прибрано, може знайти його в системі.

Таблиця 1.13 - Варіант використання UC-12

Use case name	Відео та аудіо дзвінок
Use case ID	UC-12
Goals	Керування відео та аудіо дзвінком
Actors	Користувач

Продовження таблиці 1.13

Trigger	Користувач хоче зателефонувати другу через відео або аудіо дзвінок
Pre-conditions	Користувач увійшов в систему, має перевірений та не заблокований аккаунт. Мають спільний чат. Учасники чату онлайн.
Flow of Events	Користувач переходить на сторінку чатів, обирає друга, якому хоче зателефонувати. Натискає на кнопку дзвінка та з'являється нове вікно. Співрозмовнику приходить відповідне повідомлення. За замовчанням йде аудіо дзвінок з вимкненим мікрофоном. Користувач може ввімкнути камеру та мікрофон, натиснувши на відповідні кнопки. Співрозмовник приєднався до дзвінка, може також ввімкнути камеру, мікрофон.
Extension	Якщо співрозмовник не підключиться протягом 1 хвилини, після початку виклик або відхилить його, то дзвінок буде скасовано. Вікно закриється.
Post-Condition	Користувач може спілкуватись із співрозмовником через аудіо та відео.

Таблиця 1.14 - Варіант використання UC-13

Use case name	Блокування користувача
Use case ID	UC-13
Goals	Заблокувати користувача
Actors	Адміністратор
Trigger	Адміністратор хоче обмежити доступ користувачу до системи
Pre-conditions	Адміністратор увійшов в систему.

Продовження таблиці 1.14

Flow of Events	Адміністратор переходить на сторінку адміністратора. Натискає кнопку заблокувати навпроти користувача. З'являється нове вікно в якому адміністратор може залишити причину. Натиснувши кнопку заблокувати у відкритому вікні, воно зникне, користувач буде заблокований.
Extension	Закривши нове вікно, не натиснувши кнопку заблокувати, користувач не буде заблокований.
Post-Condition	Користувач заблокований, всі його чати видалені. Він не може зайти у систему та інші не можуть його знайти крім адміністратора.

Таблиця 1.15 - Варіант використання UC-14

Use case name	Перевірка валідності облікового запису користувача
Use case ID	UC-14
Goals	Надати користувачу доступ до системи
Actors	Адміністратор
Trigger	Адміністратор хоче надати доступ користувачу до системи
Pre-conditions	Адміністратор увійшов в систему. Є не веріфікований обліковий запис
Flow of Events	Адміністратор переходить на сторінку адміністратора. Натискає на username користувача, його перенаправляє на сторінку його профілю. Адміністратор перевіряє інформацію. Якщо, обліковий запис не порушує нічого, то адміністратор повертається на сторінку адміністратора та натискає кнопку верифікувати.
Extension	-
Post-Condition	Користувач верифікований

Таблиця 1.16 - Варіант використання UC-15

Use case name	Пошук користувачів за певними фільтрами
Use case ID	UC-15
Goals	Отримати список користувачів за вказаними фільтрами
Actors	Адміністратор
Trigger	Адміністратор хоче отримати список користувачів за вказаними фільтрами
Pre-conditions	Адміністратор увійшов в систему.
Flow of Events	Адміністратор переходить на сторінку адміністратора. Вводить потрібні фільтри. Натискає кнопку фільтрації
Extension	-
Post-Condition	Адміністратор отримав список користувачів за вказаними фільтрами.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. На рисунку 1.2 наведено загальну модель вимог, а в таблицях 1.18 – 1.27 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 1.3.

	Description	Code
1	Забезпечити можливість реєстрації користувача	FR-1
2	Надати можливість користувачу увійти у систему	FR-2
3	Надати користувачу можливість редагувати інформацію про себе	FR-3
4	Надати користувачу можливість змінити пароль	FR-4
5	Надати користувачу можливість видалити обліковий запис	FR-5
6	Надати користувачу можливість керувати режимом прихованості	FR-6
7	Надати користувачу можливість керувати фото профілю	FR-7
8	Надати користувачу можливість пошуку інших користувачів цієї мережі	FR-8
9	Надати користувачу можливість передивитись інформацію про іншого користувача	FR-9
10	Надати користувачу можливість керувати чорним списком	FR-10
11	Надати користувачу можливість створити новий чат з іншим користувачем	FR-11
12	Надати можливість користувачу керувати повідомленнями в чаті	FR-12
13	Надати користувачу можливість отримати список чатів	FR-13
14	Надати користувачу можливість керувати дзівнком	FR-14
15	Надати адміністратору можливість перейти на профіль користувача	FR-15
16	Надати адміністратору можливість заблокувати користувача	FR-16
17	Надати адміністратору можливість провести верифікацію користувача	FR-17
18	Надати адміністратору можливість отримати всіх користувачів системи	FR-18
19	Надати адміністратору можливість розблокувати користувача	FR-19

Рисунок 1.2 – Модель вимог у загальному вигляді

Таблиця 1.17 – Функціональна вимога FR-1

Назва	Забезпечити можливість реєстрації користувача
Опис	Система повинна надавати можливість реєстрації користувачеві шляхом введення імені, прізвища, логіну, пошти, інформації про себе, очікувань користувача, дня народження, паролю та підтвердження паролю. Також користувач повинен вибрати зі списків стать, країну, мови та хобі, що цікавлять.

Таблиця 1.18 – Функціональна вимога FR-2

Назва	Надати можливість користувачу увійти у систему
Опис	Система повинна надавати можливість входу користувачу шляхом введення пошти, паролю.

Таблиця 1.19 – Функціональна вимога FR-3

Назва	Надати користувачу можливість редагувати інформацію про себе
Опис	Система повинна надавати можливість користувачу редагувати наступної інформації про себе: ім’я, прізвище, інформацію про себе, очікування користувача, день народження, країну, стать, список мов та хобі.

Таблиця 1.20 – Функціональна вимога FR-4

Назва	Надати користувачу можливість змінити пароль
Опис	Система повинна надавати можливість користувачу змінити пароль, якщо він увійшов до неї.

Таблиця 1.21 – Функціональна вимога FR-5

Назва	Надати користувачу можливість видалити обліковий запис
Опис	Система повинна надавати можливість користувачу повністю видалити обліковий запис.

Таблиця 1.22 – Функціональна вимога FR-6

Назва	Надати користувачу можливість керувати режимом прихованості
Опис	Система повинна надавати можливість користувачу приховати або додати свій обліковий запис.

Таблиця 1.23 – Функціональна вимога FR-7

Назва	Надати користувачу можливість керувати фото профілю
Опис	Система повинна надавати можливість користувачу додати, змінити чи видалити фото облікового запису.

Таблиця 1.24 – Функціональна вимога FR-8

Назва	Надати користувачу можливість пошуку інших користувачів цієї мережі
Опис	Система повинна надавати можливість користувачу пошуку доступних клієнтів застосунку.

Таблиця 1.25 – Функціональна вимога FR-9

Назва	Надати користувачу можливість передивитись інформацію про іншого користувача
Опис	Система повинна надавати можливість користувачу передивитись інформацію про іншого клієнта цього застосунку.

Таблиця 1.26 – Функціональна вимога FR-10

Назва	Надати користувачу можливість керувати чорним списком
Опис	Система повинна надавати можливість користувачу блокувати чи розблоковувати інших клієнтів застосунку особисто для себе.

Таблиця 1.27 – Функціональна вимога FR-11

Назва	Надати користувачу можливість створити новий чат з іншим користувачем
Опис	Система повинна надавати можливість користувачу створити новий чат з іншим клієнтом.

Таблиця 1.28 – Функціональна вимога FR-12

Назва	Надати можливість користувачу керувати повідомленнями в чаті
Опис	Система повинна надавати можливість користувачу відправляти, редагувати, видаляти власні текстові повідомлення у чаті.

Таблиця 1.29 – Функціональна вимога FR-13

Назва	Надати користувачу можливість отримати список чатів
Опис	Система повинна надавати можливість користувачу отримати всі актуальні його чати.

Таблиця 1.30 – Функціональна вимога FR-14

Назва	Надати користувачу можливість керувати дзвінком
Опис	Система повинна надавати можливість користувачу зробити відео, аудіо дзвінок, відповісти на нього, відхилити або завершити його. Керувати камерою та мікрофоном.

Таблиця 1.31 – Функціональна вимога FR-15

Назва	Надати адміністратору можливість перейти на профіль користувача
Опис	Система повинна надавати можливість адміністратору перейти на профіль, що може бути заблокованим, прихованим чи не верифікованим.

Таблиця 1.32 – Функціональна вимога FR-16

Назва	Надати адміністратору можливість заблокувати користувача
Опис	Система повинна надавати можливість адміністратору обмежити доступ користувачу, з можливістю вказати причину.

Таблиця 1.33 – Функціональна вимога FR-17

Назва	Надати адміністратору можливість провести верифікацію користувача
Опис	Система повинна надавати можливість адміністратору підтвердити коректність даних облікового запису користувача.

Таблиця 1.34 – Функціональна вимога FR-18

Назва	Надати адміністратору можливість отримати всіх користувачів системи
Опис	Система повинна надавати можливість адміністратору отримати список усіх користувачів системи, навіть заблокованих, прихованих та неверифікованих.

Таблиця 1.35 – Функціональна вимога FR-19

Назва	Надати адміністратору можливість розблокувати користувача
Опис	Система повинна надавати можливість адміністратору надати доступ заблокованому користувачу до застосунку.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19
UC-01	+																		
UC-02		+																	
UC-03								+	+										
UC-04			+	+			+												
UC-05					+														
UC-06						+													
UC-07											+								
UC-08												+	+						
UC-09												+	+						
UC-10												+	+						
UC-11										+									
UC-12														+					
UC-13															+	+		+	+
UC-14															+		+	+	
UC-15															+			+	

Рисунок 1.3 – Матриця трасування вимог

1.4.2 Розроблення нефункціональних вимог

Веб-застосунок:

– повинен бути простим у використані та мати приємний дизайн;

– повинен бути адаптованим до мобільних пристроїв;

– повинен забезпечувати стабільну роботу на наступних браузерах:

1) Google Chrome версія 88.0.4324.190 і вище;
2) Microsoft Edge версія 95.0.1020.44 і вище.

1.5 Постановка задачі

Планується розробити соціальну мережу, що буде містити користувачів, які бажають знайти нових друзів. Основною частиною застосунку буде можливість обмінюватись з користувачами текстовими повідомленнями у режимі реально часу, здійснювати відео та аудіо дзвінки без застосування сторонніх засобів спілкування. Також буде реалізовано пошук, що дозволить шукати користувачів за різними параметрами. Всі зареєстровані облікові записи будуть проходити перевірку адміністратором.

Висновки до розділу

У цьому розділі було наведено загальні положення, які відносяться до теми роботи. Також було проаналізовано та описано предметну область розробки. Було визначено, що дана розробка не потребує спеціальних алгоритмічних рішень. У розділі ми проаналізували технічні та архітектурні рішення, обрали клієнт-серверну архітектуру, для розробки соціальної мережі для пошуку друзів та спілкування. Також було вирішено використати single page application, як технологію для розробки користувацького інтерфейсу. Проведено аналіз мов програмування, баз даних, технологій, що дозволяють спілкування між клієнтом та сервером у реальному часі. Розглянули та визначили фреймворки, за допомогою яких буде розроблено соціальну мережу. Для написання клієнтської частини ми обрали Vue.js, серверної - Nest. Вирішено використати реляційну базу даних PostgreSQL та ORM Prisma, що дозволить швидше працювати з бд. Websocket і WebRTC дозволять реалізувати функціонал чату у реальному часі та дзвінків. Було проведено аналіз та порівняння вже готових програмних рішень. У цьому розділі ми також проаналізували вимоги до програмного забезпечення, розробили діаграму варіантів використання та модель вимог з їх описом,

матрицю трасування. Вказали нефункціональні вимоги до нашого веб-застосунку. Наведено постановку задачі.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для опису бізнес процесів програмного забезпечення використовується BPMN модель. На рисунку 2.1 представлена схема бізнес-процесу пошуку користувачів у нашій соціальній мережі.

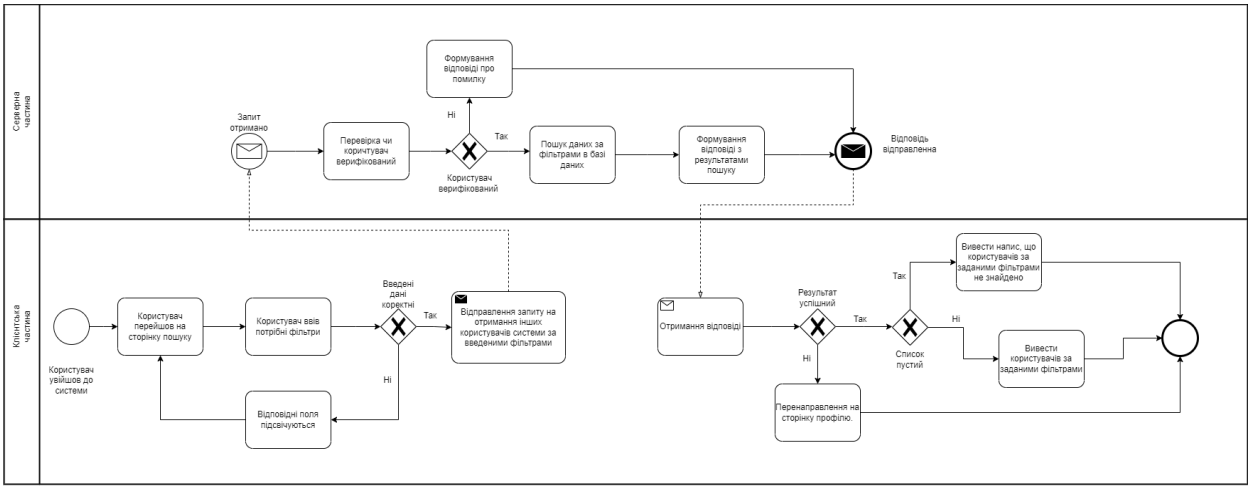


Рисунок 2.1 – Схема бізнес-процесу пошуку користувачів

Опис послідовності пошуку користувачів за певними фільтрами у системі:

– користувач переходить на сторінку пошуку;

– користувач заповнює потрібні фільтри;

– якщо обрані фільтри, не відповідають шаблону заповнення на клієнтській стороні, відповідні поля підсвічуються помилкою;

– якщо фільтри вказані коректно, то відправляється запит на сервер;

– на сервері проходить перевірка чи користувач має верифікований обліковий запис;

– якщо ні, то формується відповідь про помилку та надсилається користувачу;

– якщо верифікований, то проходить пошук користувачів в системі за вказаними фільтрами;

– формується відповідь з результатами пошуку та надсилається користувачу;

– коли користувач отримає відповідь, то відбудеться перевірка успіху запиту;

– якщо результат не успішний, то клієнта буде перенаправлено на сторінку профілю.

– в іншому випадку, буде визначено чи список шуканих клієнтів пустий;

– якщо так, то буде виведено напис про відсутність клієнтів системі за обраними фільтрами.

– якщо ні, то будуть виведений результат пошуку.

На рисунку 2.2 наведено схему бізнес-процесу надсилання повідомлення користувачу.

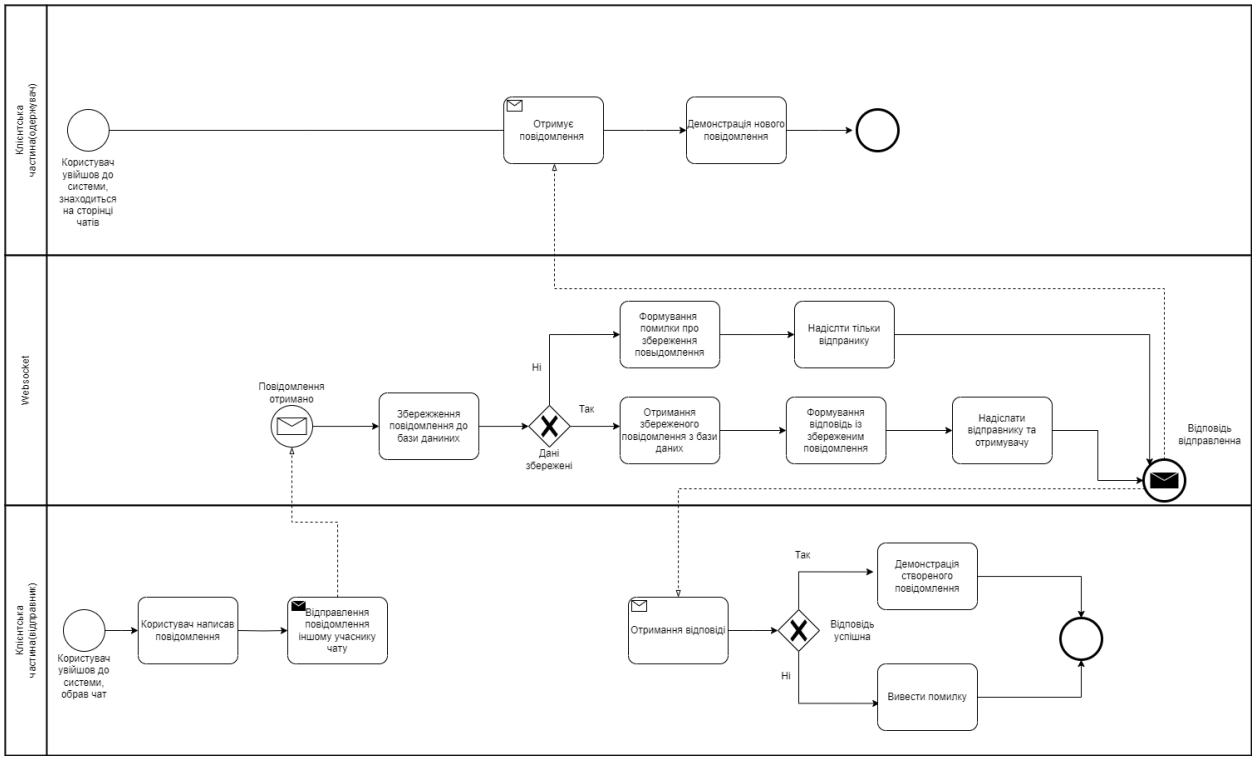


Рисунок 2.2 – Схема бізнес-процесу надсилання повідомлення користувачу

Опис послідовності надсилання повідомлення співрозмовнику:

– користувач знаходиться на сторінці чату, та обрав співрозмовника, кому хоче написати.

– користувач пише повідомлення та відправляє його;

– на сервері повідомлення зберігається в базі даних;

– якщо дані не збереглися або виникла якась помилка, то формується відповідь з помилкою;

– дані надсилаються лише відправнику;

– якщо повідомлення зберіглося в базі даних, то формується відповідь;

– дані надсилаються обом користувачам чату.

– якщо отримувач знаходиться в мережі та на сторінці чату, то йому демонструється нове повідомлення.

– якщо відправник отримав не успішну відповідь, то йому буде виведено відповідну помилку;

– в іншому випадку, йому буде продемонстроване відправлене повідомлення.

2.2 Архітектура програмного забезпечення

За основу архітектури програмного забезпечення було взято клієнт-серверний шаблон. Його деталізація представлена на рисунку 2.3.

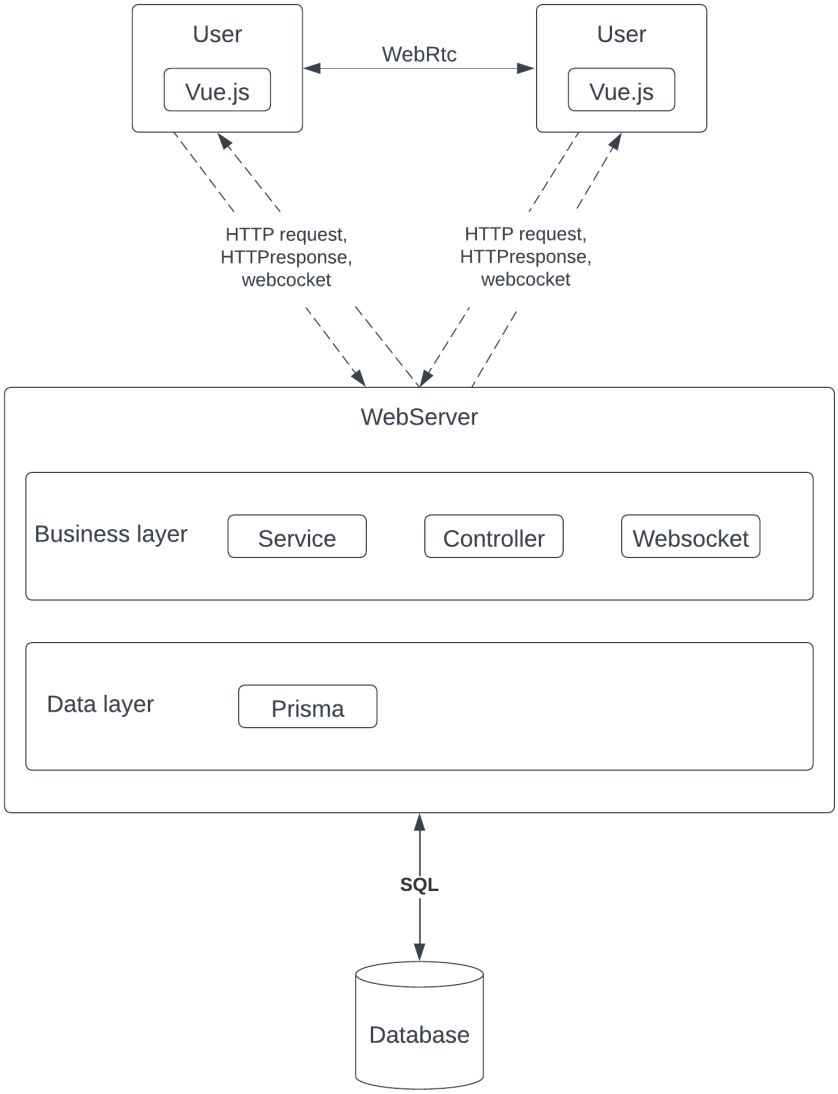


Рисунок 2.3 – Деталізація клієнт-серверної архітектури

Клієнт-серверна архітектура складається з двох модулів. Клієнт відповідає за представлення даних користувачу. Сервер відповідає за всю бізнес логіку застосунку. Компоненти в межах певного рівня мають справу лише з логікою, що стосується його. Компонент Prisma повертає єдиний зв'язок з базою даних, дозволяє генерувати Sql. Service – обробляє дані, що надійшли з бд. Controller повертає оброблені дані користувачу, на клієнтську частину та опрацьовує отриману інформацію. Websocket забезпечує двонаправлений канал зв’язку між клієнтом та сервером, що дозволяє обмінюватись даними у реальному часі. Vue.js – відповідає за представлення даних. Клієнт та сервер спілкуються між собою за допомогою http та веб-

сокет з’єднання. Користувачі діляться між собою потоковим відео та аудіо за допомогою WebRTC технології.

2.3 Конструювання програмного забезпечення

В якості системи управління базами даних використовується Postgres. База даних серверу призначена для зберігання користувачів, а також даних про їх чати, повідомлення, причини блокування. Також в базі даних зберігаються списки країн, мов, хобі. Опис таблиць бази даних наведено у таблицях 2.1 - 2.12. Модель бази даних наведена на рисунку 2.4.

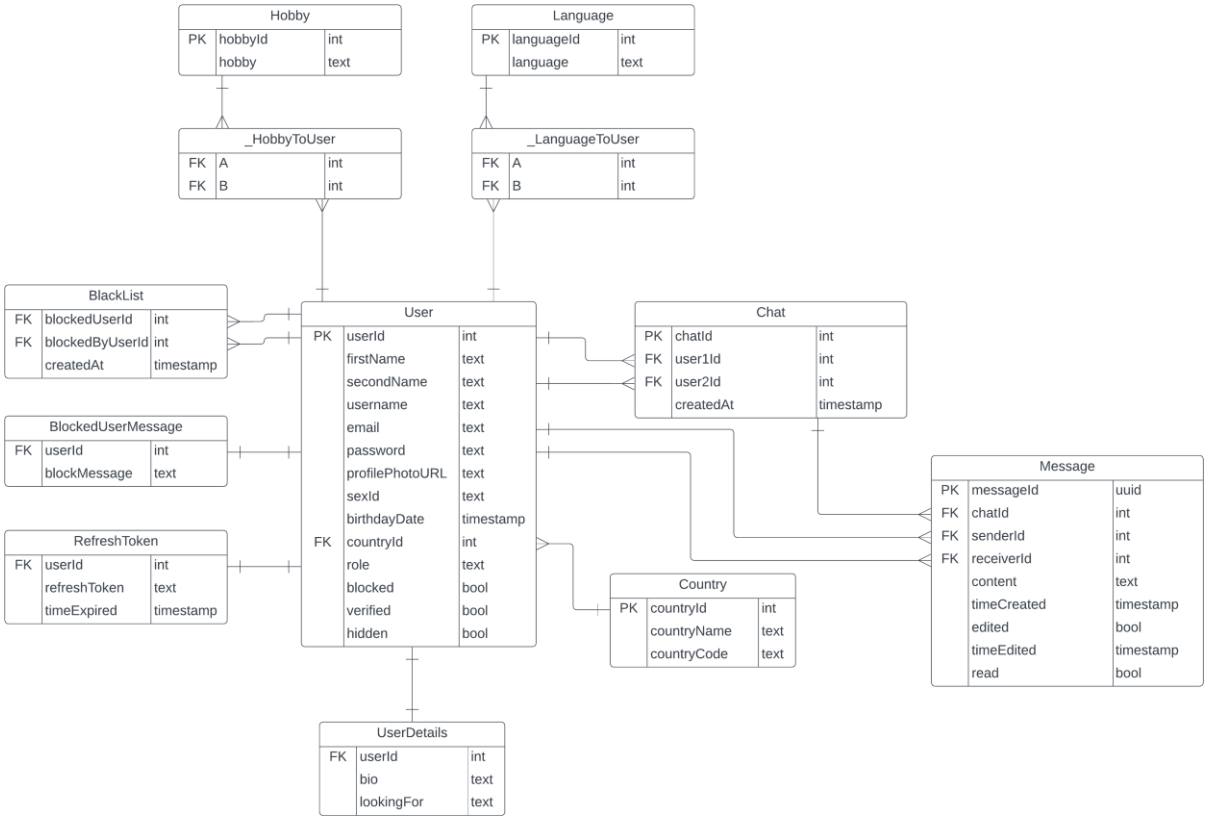


Рисунок 2.4 – Фізична модель бази даних

Таблиця 2.1 – Опис таблиці User

Таблиця	Назва поля	Тип даних	Опис
User	userId	int	ідентифікаційний номер користувача

Продовження таблиці 2.1

Таблиця	Назва поля	Тип даних	Опис
User	firstName	text	ім'я користувача
	secondName	text	прізвище користувача
	username	text	логін користувача
	email	text	електронна пошта користувача
	profilePhotoURL	text	назва файлу фото користувача
	sexId	text	стать користувача
	birthdayDate	timestamp	день народження користувача
	countryId	int	посилання на запис у таблиці Country, де вказано країну походження користувача
	role	text	роль користувача в системі
	blocked	bool	статус блокування користувача в системі
	verified	bool	статус перевірки облікового запису користувача

Продовження таблиці 2.1

Таблиця	Назва поля	Тип даних	Опис
User	Hidden	bool	статус режиму прихованості
	password	text	зашифрований пароль користувача

Таблиця 2.2 – Опис таблиці UserDetails

Таблиця	Назва поля	Тип даних	Опис
UserDetails	userId	int	посилання на запис користувача в таблиці User
	bio	text	інформація користувача
	lookingFor	text	очікування користувача

Таблиця 2.3 – Опис таблиці RefreshToken

Таблиця	Назва поля	Тип даних	Опис
RefreshToken	userId	int	посилання на запис користувача в таблиці User
	refreshToken	text	токен оновлення авторизації користувача
	dateExpired	timestamp	дата до якої дійсний токен

Таблиця 2.4 – Опис таблиці BlockUserMessage

Таблиця	Назва поля	Тип даних	Опис
BlockUserMessage	userId	int	посилання на запис користувача в таблиці User
	blockMessage	text	причина блокування користувача

Таблиця 2.5 – Опис таблиці BlackList

Таблиця	Назва поля	Тип даних	Опис
BlackList	blockedUserId	int	посилання на запис користувача в таблиці User, що заблокований
	blockedByUserId	int	посилання на запис користувача в таблиці User, що заблоковував
	createdAt	timestamp	дата створення запису

Таблиця 2.6 – Опис таблиці Hobby

Таблиця	Назва поля	Тип даних	Опис
Hobby	hobbyId	int	індентифікаційний номер хобі
	hobby	text	назва хобі

Таблиця 2.7 – Опис таблиці _HobbyToUser

Таблиця	Назва поля	Тип даних	Опис
_HobbyToUser	A	int	посилання на запис у таблиці Hobby
	B	text	посилання на запис у таблиці User

Таблиця 2.8 – Опис таблиці Language

Таблиця	Назва поля	Тип даних	Опис
Language	languageId	int	індентифікаційний номер мови
	language	text	назва мови

Таблиця 2.9 – Опис таблиці _HobbyToUser

Таблиця	Назва поля	Тип даних	Опис
_LanguageToUser	A	int	посилання на запис у таблиці Language
	B	text	посилання на запис у таблиці User

Таблиця 2.10 – Опис таблиці Country

Таблиця	Назва поля	Тип даних	Опис
Country	countryId	int	індентифікаційний номер країни

Продовження таблиці 2.10

Таблиця	Назва поля	Тип даних	Опис
Country	countryName	text	назва мови
	countryCode	text	код країни

Таблиця 2.11 – Опис таблиці Chat

Таблиця	Назва поля	Тип даних	Опис
Chat	chatId	int	індентифікаційний номер чату
	user1Id	int	посилання на учасника чату номер один до таблиці User
	user2Id	int	посилання на учасника чату номер 2 до таблиці User
	createdAt	timestamp	дата створення чату

Таблиця 2.12 – Опис таблиці Message

Таблиця	Назва поля	Тип даних	Опис
Message	messageId	int	індентифікаційний номер повідомлення
	senderId	int	посилання на відпраника повідомлення до таблиці User

Продовження таблиці 2.12

Таблиця	Назва поля	Тип даних	Опис
Message	receiverId	int	посилання на отримувача повідомлення до таблиці User
	content	text	текст повідомлення
	timeCreated	timestamp	дата створення повідомлення
	edited	bool	чи відредаговане повідомлення
	timeEdited	timestamp	дата редагування повідомлення
	read	bool	чи прочитане повідомлення отримувачем

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.13.

Таблиця 2.13 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Visual Studio Code	Головне середовище розробки програмного забезпечення.

Продовження таблиці 2.13

№ п/п	Назва утиліти	Опис застосування
2	Postman	Програмне забезпечення необхідне для тестування rest запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
3	TablePlus	Програмне забезпечення яке надає легкий графічний інтерфейс для доступу до бази даних.

Аналіз системних вимог можна побачити в таблиці 2.14:

Таблиця 2.14 – Аналіз рекомендованих системних вимог

№ п/п	Назва утиліти	Опис застосування
1	тип процесору	Intel Core i5 та новіші
2	Оперативна пам’ять	8 Гб та більше
3	Швидкість інтернету	Від 10 мегабіт

2.4 Аналіз безпеки даних

Авторизація проходить за рахунок використання Jwt. Користувач, який не пройшов авторизацію, не може отримати доступ до інформації про інших клієнтів. Користувач, що увійшов до системи, може видалити повністю свій аккаунт разом з чатами, повідомленнями, обмежити доступ до інформації про себе чи змінити її. Пароль користувача зберігається в системі в зашифрованому вигляді.

Висновки до розділу

В другому розділі було проведено моделювання та конструювання програмного забезпечення. Ми розробили схеми бізнес-процесів пошуку користувачів у нашій соціальній мережі за певними параметрами та надсилання повідомлень співрозмовнику. Схеми були розроблені на основі BPMN та описані у текстовому варіанті. Обрано клієнт-серверний шаблон, як основу архітектури програмного забезпечення. Деталізацію представили на рисунку 2.3 та описали її. Було вказано Postgres, як систему управління базами даних та наведено фізичну модель бази даних на рисунку 2.4. Також ми детально описали кожну таблицю бд, з визначенням кожного поля. Вказали утиліти, що були використанні при розробці програмного забезпечення та мінімальні системні вимоги. Провели аналіз безпеки даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

У рамках цьому підрозділу було проведено аналіз статичного коду веб-застосунку у спеціальному веб-сервісі Embold та наведено результати. На рисунку 3.1 представлено оцінку клієнтської частини.

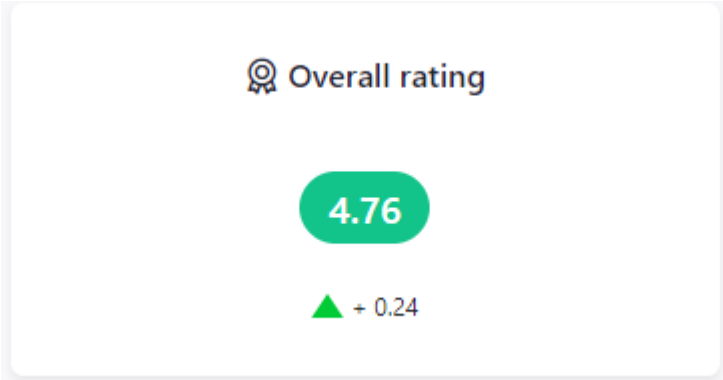


Рисунок 3.1 – Оцінка проведеного аналізу клієнтської частини

На рисунку 3.2 наведені оцінки з певних метрик. Можна помітити, що клієнтська частина має не погані оцінки, її можна буде легко підтримувати, доповнювати різний функціонал.

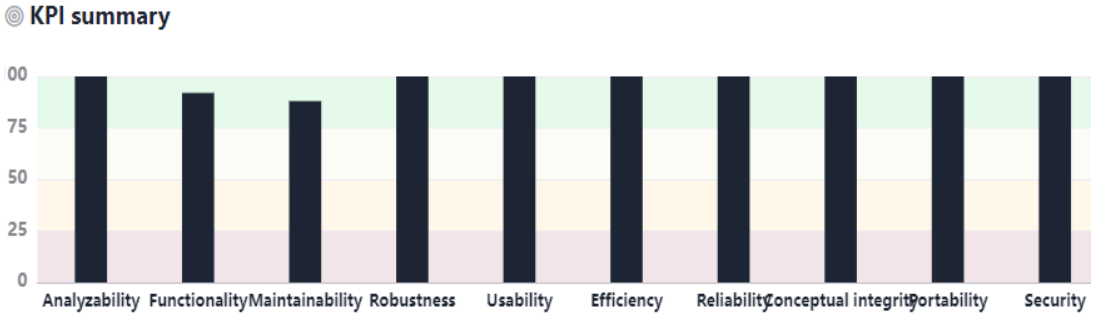


Рисунок 3.2 – Оцінки певних метрик аналізу клієнтської частини

На рисунку 3.3 представлено загальну оцінку аналізу коду серверної частини.

Рисунок 3.3 - Оцінка проведеного аналізу серверної частини

На рисунку 3.4 наведені оцінки певних метрик. Можна помітити, що серверна частина отримала непогані результати, код можна легко проаналізувати, підтримувати. Також застосунок можна з легкістю перенести з однієї платформи на іншу. Також метрика Security має високу оцінку, що демонструє високий степінь надійності.

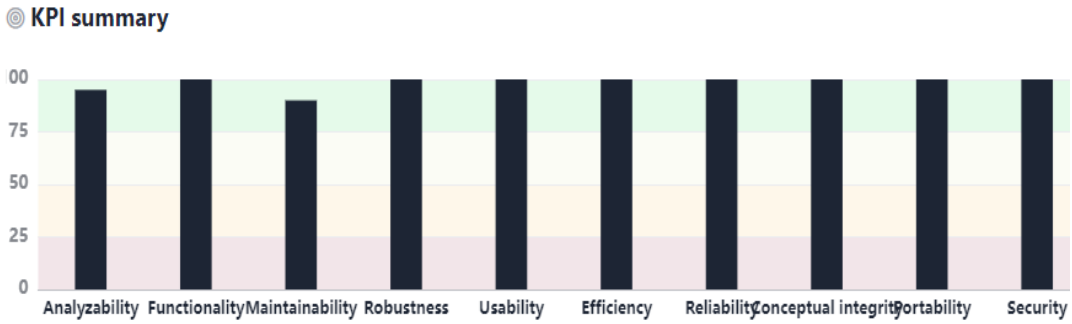


Рисунок 3.4 – Оцінки певних метрик аналізу серверної частини

Також було виконано усі вказані нефункціональні вимоги. Веб-застосунок має приємний дизайн та є досить простим у використані. Також було адаптовано клієнтську частину під використання на мобільних пристроях. Веб-застосунок стабільно працює на наступних браузерах:

– Google Chrome версія 88.0.4324.190 і вище;
– Microsoft Edge версія 95.0.1020.44 і вище.

3.2 Опис процесів тестування

Було проведене мануальне тестування веб-застосунку, опис відповідних тестів наведено у таблицях 3.1 – 3.30.

Таблиця 3.1 – Тест 1.1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Ім’я, прізвище, дата народження, стать, країна, електронна пошта, логін, інформація про себе та очікування, список мов та хобі, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: ім’я, прізвище, дата народження, що має обмеження мінімального та максимального віку 12 та 120 років, коректна електронна пошта та логін, які до цього не були зареєстровані в системі, інформація про себе та очікування від веб-застосунку, пароль від 6 до 18 символів, підтвердження паролю, яке співпадає з раніше введеним паролем. Також обирається стать, країна проживання, хобі та інтереси. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.

Таблиця 3.2 – Тест 1.2

Тест	Реєстрація користувача з електронною поштою, що існує в системі
Модуль	Реєстрація користувача

Продовження таблиці 3.2

Номер тесту	1.2
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Ім’я, прізвище, дата народження, стать, країна, електронна пошта, логін, інформація про себе та очікування, список мов та хобі, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: ім’я, прізвище, дата народження, що має обмеження мінімального та максимального віку 12 та 120 років, коректна електронна пошта, яка вже використовується, логін, який до цього не був зареєстрований в системі, інформація про себе та очікування від веб-застосунку, пароль від 6 до 18 символів, підтвердження паролю, яке співпадає з раніше введеним паролем. Також обирається стать, країна проживання, хобі та інтереси. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Демонструється повідомлення про те, що пошта вже зайнята.
Фактичний результат	Демонструється повідомлення про те, що пошта вже зайнята.

Таблиця 3.3 – Тест 1.3

Тест	Реєстрація користувача з логіном, що існує в системі
Модуль	Реєстрація користувача
Номер тесту	1.3
Початковий стан системи	Користувач знаходиться на сторінці реєстрації

Продовження таблиці 3.3

Вхідні данні	Ім’я, прізвище, дата народження, стать, країна, електронна пошта, логін, інформація про себе та очікування, список мов та хобі, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: ім’я, прізвище, дата народження, що має обмеження мінімального та максимального віку 12 та 120 років, коректна електронна пошта, яка до цього не була зареєстрована в системі, логін, що вже використовується, інформація про себе та очікування від веб-застосунку, пароль від 6 до 18 символів, підтвердження паролю, яке співпадає з раніше введеним паролем. Також обирається стать, країна проживання, хобі та інтереси. Після цього натискається кнопка підтвердження реєстрації.
Очікуваний результат	Демонструється повідомлення про те, що логін вже зайнятий.
Фактичний результат	Демонструється повідомлення про те, що логін вже зайнятий.

Таблиця 3.4 – Тест 1.4

Тест	Авторизація користувача
Модуль	Авторизація користувача
Номер тесту	1.4
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Електронна пошта, пароль

Продовження таблиці 3.4

Опис проведення тесту	У відповідні поля вводяться: електронна пошта, яка вже зареєстрована в системі, пароль від 6 до 18 символів. Після цього натискається кнопка входу.
Очікуваний результат	Після успішної авторизації, користувач перенаправляється на сторінку свого профілю.
Фактичний результат	Після успішної авторизації, користувач перенаправляється на сторінку свого профілю.

Таблиця 3.5 – Тест 1.5

Тест	Авторизація користувача з електронною поштою, що не зареєстрована в системі
Модуль	Авторизація користувача
Номер тесту	1.5
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: електронна пошта, яка не була зареєстрована в системі, пароль від 6 до 18 символів. Після цього натискається кнопка входу.
Очікуваний результат	Виводиться повідомлення про невірно введені дані.
Фактичний результат	Виводиться повідомлення про невірно введені дані.

Таблиця 3.6 – Тест 1.6

Тест	Авторизація користувача з невірним паролем
Модуль	Авторизація користувача
Номер тесту	1.5

Продовження таблиці 3.6

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: електронна пошта, яка зареєстрована в системі, пароль від 6 до 18 символів, який не є вірним. Після цього натискається кнопка входу.
Очікуваний результат	Виводиться повідомлення про невірно введені дані.
Фактичний результат	Виводиться повідомлення про невірно введені дані.

Таблиця 3.7 – Тест 1.7

Тест	Зміна інформації про користувача
Модуль	Зміна інформації про користувача
Номер тесту	1.7
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	Ім’я
Опис проведення тесту	У відповідному полі змінюється інформація. Після цього натискається кнопка збереження.
Очікуваний результат	Демонструється повідомлення, що дані оновлено успішно. Користувач повинен увійти знов в систему. Його обліковий запис стає не верифікованим.
Фактичний результат	Демонструється повідомлення, що дані оновлено успішно. Користувач повинен увійти знов в систему. Його обліковий запис стає не верифікованим.

Таблиця 3.8 – Тест 1.8

Тест	Зміна фото облікового запису користувача
Модуль	Зміна інформації про користувача
Номер тесту	1.8
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	Файл фото з розширенням jpeg або png.
Опис проведення тесту	Користувач натискає на надпис завантажити фото, якщо воно відсутнє, або на вже існуючий аватар. Обирає файл у вікні, що з’явилось. Натискає змінити фото.
Очікуваний результат	У разі успіху, фото буде представлено на сторінці. З’явиться кнопка видалення фото.
Фактичний результат	Фото продемонстровано користувачу. З’явилась кнопка видалення зображення.

Таблиця 3.9 – Тест 1.9

Тест	Зміна паролю
Модуль	Зміна інформації про користувача
Номер тесту	1.9
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	Пароль.
Опис проведення тесту	Користувач вводить новий пароль від 6 до 18 символів та його повторення. Натискає кнопку змінити пароль.
Очікуваний результат	У разі успіху, пароль буде змінено. З’явиться повідомлення, що пароль змінено. Поле з паролем стане пустим.

Продовження таблиці 3.9

Фактичний результат	Пароль змінено, з’явилось повідомлення, про успіх. Поле стало пустим.

Таблиця 3.10 – Тест 1.10

Тест	Ввімкнення режиму прихованості
Модуль	Зміна інформації про користувача
Номер тесту	1.10
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	-
Опис проведення тесту	Користувач натискає на перемикач режиму прихованості.
Очікуваний результат	Користувач не потрапляє до результатів пошуку інших клієнтів системи. Користувачу демонструється повідомлення, що він увійшов в режим прихованості.
Фактичний результат	Користувач не потрапляє до результатів пошуку інших клієнтів системи. Користувачу демонструється повідомлення, що він увійшов в режим прихованості.

Таблиця 3.11 – Тест 1.11

Тест	Вимкнення режиму прихованості
Модуль	Зміна інформації про користувача
Номер тесту	1.11
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	-

Продовження таблиці 3.11

Опис проведення тесту	Користувач натискає на перемикач режиму прихованості.
Очікуваний результат	Користувач знову потрапляє до результатів пошуку інших клієнтів системи. Зникає повідомлення, про увімкнений режим прихованості.
Фактичний результат	Користувач знову потрапляє до результатів пошуку інших клієнтів системи. Зникає повідомлення, про увімкнений режим прихованості.

Таблиця 3.12 – Тест 1.12

Тест	Видалення користувача
Модуль	Зміна інформації про користувача
Номер тесту	1.12
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку видалення облікового запису. Підтверджує свої дії, у вікні, що з’явилось.
Очікуваний результат	Користувач та вся його інформація повністю видаляється з системи.
Фактичний результат	Користувач та вся його інформація повністю видаляється з системи.

Таблиця 3.13 – Тест 1.13

Тест	Видалення користувача (відміна)
Модуль	Зміна інформації про користувача

Продовження таблиці 3.13

Номер тесту	1.13
Початковий стан системи	Користувач авторизований та знаходиться на сторінці зміни даних профілю
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку видалення облікового запису. Натискає на кнопку скасувати.
Очікуваний результат	Користувач та вся його інформація залишається в системі.
Фактичний результат	Користувач та вся його інформація залишається в системі.

Таблиця 3.14 – Тест 1.14

Тест	Пошук користувачів без параметрів
Модуль	Пошук користувачів
Номер тесту	1.14
Початковий стан системи	Користувач авторизований
Вхідні данні	-
Опис проведення тесту	Користувач переходить на сторінку пошуку.
Очікуваний результат	Демонструється список користувачів, що не є заблокованими, не знаходяться в особистому чорному списку, та пройшли верифікацію.
Фактичний результат	Демонструється список користувачів, що не є заблокованими, не знаходяться в особистому чорному списку, та пройшли верифікацію.

Таблиця 3.15 – Тест 1.15

Тест	Пошук користувачів за параметрами
Модуль	Пошук користувачів
Номер тесту	1.15
Початковий стан системи	Користувач авторизований та знаходиться на сторінці пошуку.
Вхідні данні	Логін, мінімально та максимально допустимий вік, мови, хобі, країни, стать.
Опис проведення тесту	Користувач вводить наступні поля: логін, мінімальний вік, що не повинен бути меншим за 12 років та максимальний вік, який не має бути більшим ніж 120 років. Обирає декілька мов, хобі, країн, що його цікавлять. Вказує стать. Натискає на кнопку фільтрації.
Очікуваний результат	Демонструється список користувачів, що не є заблокованими, не знаходяться в особистому чорному списку, та пройшли верифікацію та відповідають заданим параметрам.
Фактичний результат	Демонструється список користувачів, що не є заблокованими, не знаходяться в особистому чорному списку, та пройшли верифікацію та відповідають заданим параметрам.

Таблиця 3.16 – Тест 1.16

Тест	Пошук користувачів за параметрами (не знайдено жодного співпадіння)
Модуль	Пошук користувачів
Номер тесту	1.16
Початковий стан системи	Користувач авторизований та знаходиться на сторінці пошуку.

Продовження таблиці 3.16

Вхідні данні	Логін, мінімально та максимально допустимий вік, мови, хобі, країни, стать.
Опис проведення тесту	Користувач вводить наступні поля: логін, мінімальний вік, що не повинен бути меншим за 12 років та максимальний вік, який не має бути більшим ніж 120 років. Обирає декілька мов, хобі, країн, що його цікавлять. Вказує стать. Натискає на кнопку фільтрації.
Очікуваний результат	Демонструється напис, що жодного користувача не знайдено за вказаними параметрами.
Фактичний результат	Демонструється напис, що жодного користувача не знайдено за вказаними параметрами.

Таблиця 3.17 – Тест 1.17

Тест	Пошук користувачів за параметрами з сортуванням
Модуль	Пошук користувачів
Номер тесту	1.17
Початковий стан системи	Користувач авторизований та знаходиться на сторінці пошуку.
Вхідні данні	Логін, мінімально та максимально допустимий вік, мови, хобі, країни, стать.
Опис проведення тесту	Користувач вводить наступні поля: логін, мінімальний вік, що не повинен бути меншим за 12 років та максимальний вік, який не має бути більшим ніж 120 років. Обирає декілька мов, хобі, країн, що його цікавлять. Вказує стать. Обирає, варіант сортування списку за датою створення облікового запису за зростанням. Натискає на кнопку фільтрації.

Продовження таблиці 3.17

Очікуваний результат	Демонструється відсортований список користувачів за датою створення облікового запису за зростанням, що не є заблокованими, не знаходяться в особистому чорному списку, та пройшли верифікацію та відповідають заданим параметрам.
Фактичний результат	Демонструється відсортований список користувачів за датою створення облікового запису за зростанням, що не є заблокованими, не знаходяться в особистому чорному списку, та пройшли верифікацію та відповідають заданим параметрам.

Таблиця 3.18 – Тест 1.18

Тест	Додавання користувача в чорний список
Модуль	Керування чорним списком
Номер тесту	1.18
Початковий стан системи	Користувач авторизований та знаходиться на сторінці іншого користувача.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку з трьома вертикально розташованими крапочками. У контекстному вікні, що з’явилось, натискає на надпис «Додати до чорного списку».
Очікуваний результат	Контекстне меню зникає. Видаляється спільний чат, якщо він існує. Користувач перенаправляється на сторінку пошуку. Заблокований користувач зникає з результатів пошуку.

Продовження таблиці 3.18

Фактичний результат	Контекстне меню зникло. Видалився спільний чат. Користувач переправлений на сторінку пошуку. Заблокований користувач зник з результатів пошуку.

Таблиця 3.19 – Тест 1.19

Тест	Видалення користувача з чорного списку
Модуль	Керування чорним списком
Номер тесту	1.19
Початковий стан системи	Користувач авторизований та знаходиться на сторінці чорного списку.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку видалення навпроти логіна іншого клієнта системи.
Очікуваний результат	Заблокований користувач зникає зі списку. Потрапляє до результатів пошуку.
Фактичний результат	Заблокований користувач зник зі списку. Потрапляє до результатів пошуку.

Таблиця 3.20 – Тест 1.20

Тест	Створення нового чату, якщо його ще немає
Модуль	Керування чатами та повідомленнями
Номер тесту	1.20
Початковий стан системи	Користувач авторизований та знаходиться на сторінці іншого клієнта системи.
Вхідні данні	-

Продовження таблиці 3.20

Опис проведення тесту	Користувач натискає на кнопку надіслати повідомлення.
Очікуваний результат	Користувач буде переправлений на сторінку з усіма чатами. З’явиться новий чат, та буде надіслане перше автоматично згенероване повідомлення. Отримувач побачить новий чат та повідомлення, якщо буде на сторінці.
Фактичний результат	Користувач буде переправлений на сторінку з усіма чатами. З’явиться новий чат, та буде надіслане перше автоматично згенероване повідомлення. Отримувач побачив новий чат та повідомлення.

Таблиця 3.21 – Тест 1.21

Тест	Видалення чату
Модуль	Керування чатами та повідомленнями
Номер тесту	1.21
Початковий стан системи	Користувач авторизований та знаходиться на сторінці чатів.
Вхідні данні	-
Опис проведення тесту	Користувач обирає потрібний чат. Натискає на кнопку, що відкриє контекстне меню. Потім обирає варіант – видалити чат.
Очікуваний результат	Чат та усі повідомлення видалено у двох користувачів. Пропонується обрати новий чат для переписки.
Фактичний результат	Чат та усі повідомлення видалено у двох користувачів. Запропоновано обрати новий чат для переписки.

Таблиця 3.22 – Тест 1.22

Тест	Створення чату, якщо він вже існує
Модуль	Керування чатами та повідомленнями
Номер тесту	1.22
Початковий стан системи	Користувач авторизований та знаходиться на сторінці іншого клієнта соціальної мережі.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку надіслати повідомлення.
Очікуваний результат	Користувач переправлений на сторінку чатів. Одразу обрано чат з бажаним клієнтом.
Фактичний результат	Користувач переправлений на сторінку чатів. Одразу обрано чат з бажаним клієнтом.

Таблиця 3.23 – Тест 1.23

Тест	Надсилання нового повідомлення
Модуль	Керування чатами та повідомленнями
Номер тесту	1.23
Початковий стан системи	Користувач авторизований, знаходиться на сторінці чатів, обрав чат.
Вхідні данні	-
Опис проведення тесту	Користувач вводить текстове повідомлення. Натискає на кнопку з зображенням паперового літачка.
Очікуваний результат	Створено нове повідомлення в чаті. Отримувач може побачити його.
Фактичний результат	Створено нове повідомлення в чаті. Отримувач може побачити його.

Таблиця 3.24 – Тест 1.24

Тест	Помилка надсилання нового повідомлення
Модуль	Керування чатами та повідомленнями
Номер тесту	1.24
Початковий стан системи	Користувач авторизований, знаходиться на сторінці чатів, обрав чат.
Вхідні данні	-
Опис проведення тесту	Користувач вводить текстове повідомлення. Натискає на кнопку з зображенням паперового літачка.
Очікуваний результат	Демонструється помилка. Користувачі не можуть побачити повідомлення. Воно не збережене в системі.
Фактичний результат	Демонструється помилка. Користувачі не можуть побачити повідомлення. Воно не збережене в системі.

Таблиця 3.25 – Тест 1.25

Тест	Редагування власного повідомлення
Модуль	Керування чатами та повідомленнями
Номер тесту	1.25
Початковий стан системи	Користувач авторизований, знаходиться на сторінці чатів, обрав чат.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на повідомлення, що хоче змінити. Обирає у контекстному меню варіант «змінити». У новому вікні редагує повідомлення та натискає кнопку зберегти.
Очікуваний результат	Повідомлення відредаговано. Користувачі можуть побачити спеціальну помітку.

Продовження таблиці 3.25

Фактичний результат	Повідомлення відредаговано. Користувачі бачать спеціальну помітку.

Таблиця 3.26 – Тест 1.26

Тест	Видалення власного повідомлення
Модуль	Керування чатами та повідомленнями
Номер тесту	1.26
Початковий стан системи	Користувач авторизований, знаходиться на сторінці чатів, обрав чат.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на повідомлення, що хоче видалити. Обирає у контекстному меню варіант «видалити». Підтверджує своє рішення.
Очікуваний результат	Повідомлення видалено. Користувачі більше його не бачать.
Фактичний результат	Повідомлення видалено. Користувачі більше його не бачать.

Таблиця 3.27 – Тест 1.27

Тест	Дзвінок користувачу
Модуль	Керування чатами та повідомленнями
Номер тесту	1.27
Початковий стан системи	Користувач авторизований, знаходиться на сторінці з списком чатів, обрав співбесідника, інший учасник чату знаходиться в мережі.
Вхідні данні	-

Продовження таблиці 3.27

Опис проведення тесту	Користувач натискає на кнопку дзвінка.
Очікуваний результат	Користувач переправлений на сторінку дзвінка. Співбесідник отримує повідомлення, про виклик. Демонструється напис, що йде виклик.
Фактичний результат	Користувач переправлений на сторінку дзвінка. Співбесідник отримує повідомлення, про виклик. Демонструється напис, що йде виклик.

Таблиця 3.28 – Тест 1.28

Тест	Співбесідник відповів на дзвінок
Модуль	Керування чатами та повідомленнями
Номер тесту	1.28
Початковий стан системи	Користувач авторизований, знаходиться на сторінці дзвінка.
Вхідні данні	-
Опис проведення тесту	-
Очікуваний результат	Користувачу демонструється відео та звук співбесідника, якщо вони ввімкненні.
Фактичний результат	Користувачу демонструється камера та звук співбесідника, якщо вони ввімкненні.

Таблиця 3.29 – Тест 1.29

Тест	Співбесідник відхилив дзвінок
Модуль	Керування чатами та повідомленнями
Номер тесту	1.29
Початковий стан системи	Користувач авторизований, знаходиться на сторінці дзвінка.
Вхідні данні	-
Опис проведення тесту	-
Очікуваний результат	Користувачу демонструється повідомлення, що дзвінок відхиллено. Користувач буде переправлений на сторінку з чатами через 5 секунд.
Фактичний результат	Користувачу демонструється повідомлення, що дзвінок відхиллено. Користувач був переправлений на сторінку з чатами через 5 секунд.

Таблиця 3.30 – Тест 1.30

Тест	Користувач завершив розмову
Модуль	Керування чатами та повідомленнями
Номер тесту	1.30
Початковий стан системи	Користувач авторизований, знаходиться на сторінці дзвінка, співбесідник вже під ’єднався.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку завершення розмов.

Продовження таблиці 3.30

Очікуваний результат	Співбесіднику демонструється напис про завершення розмови. Ініціатора дзвінка буде переправлено на сторінку чатів одразу, іншого користувача – через 5 секунд.
Фактичний результат	Співбесіднику демонструється напис про завершення розмови. Ініціатора дзвінка буде переправлено на сторінку чатів одразу, іншого користувача – через 5 секунд.

3.3 Опис контрольного прикладу

Розглянемо приклад дзвінка, з усіма його розгалуженнями. Два користувачі увійшли до системи, мають спільний чат і знаходяться в мережі на момент дзвінка.

Перший користувач починає дзвінок. На рисунку 3.5 продемонстровано екран ініціатора, а на рисунку 3.6 – екран отримувача, після того, як було розпочато дзвінок. Ініціатору було продемонстровано повідомлення, що йде очікування на іншого учасника. В той час одержувач бачить нове вікно, що демонструє інформацію про дзвінок та надає можливість скасувати чи прийняти виклик.

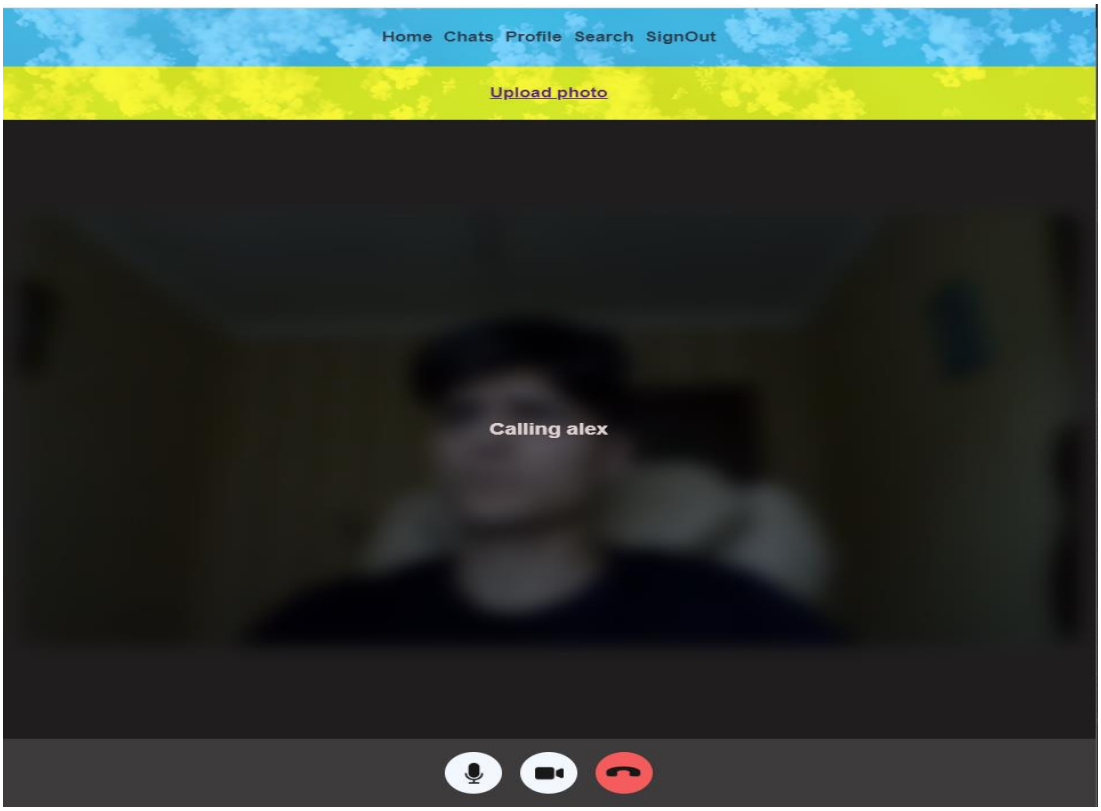


Рисунок 3.5 – Екран ініціатора під час початку дзвінка

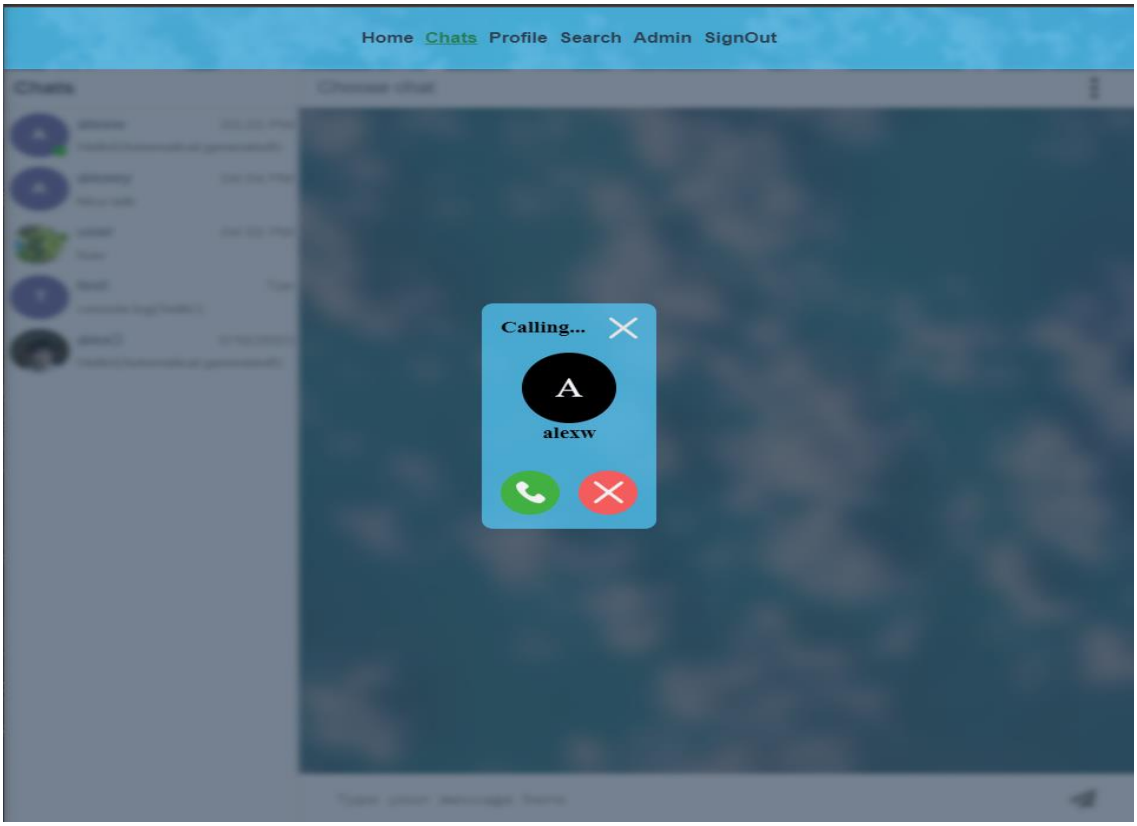


Рисунок 3.6 – Екран отримувача під час початку дзвінка

Якщо, одержувач скасував виклик, то в нього закриється вікно, а ініціатору буде виведено відповідне повідомлення, і якщо користувач не покине сторінку за 5 секунд, то його буде автоматично переправлено на сторінку з чатами. На рисунку 3.7 наведено екран користувача, що почав дзвінок, після того як виклик було відхилено.

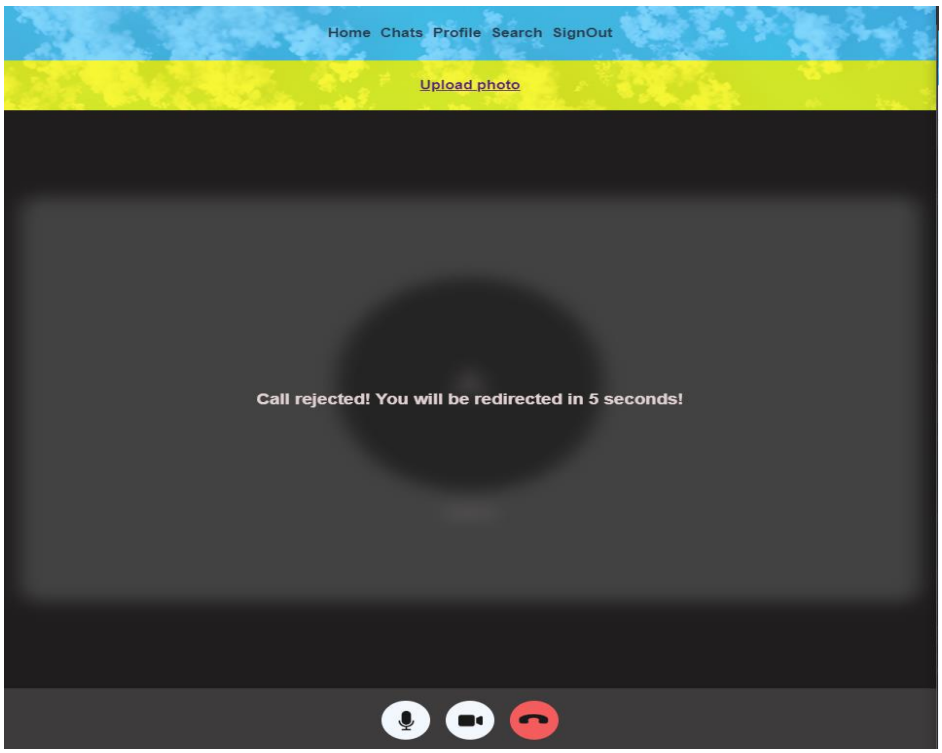


Рисунок 3.7 – Екран користувача, що почав виклик, після відхилення дзвінка

Користувач, що почав дзвінок може керувати відео та аудіо, навіть поки співбесідник не приєднався. На рисунку 3.8 наведено екран ініціатора дзвінка з вимкненою камерою та звуком, які за замовчанням у нього є ввімкнені.

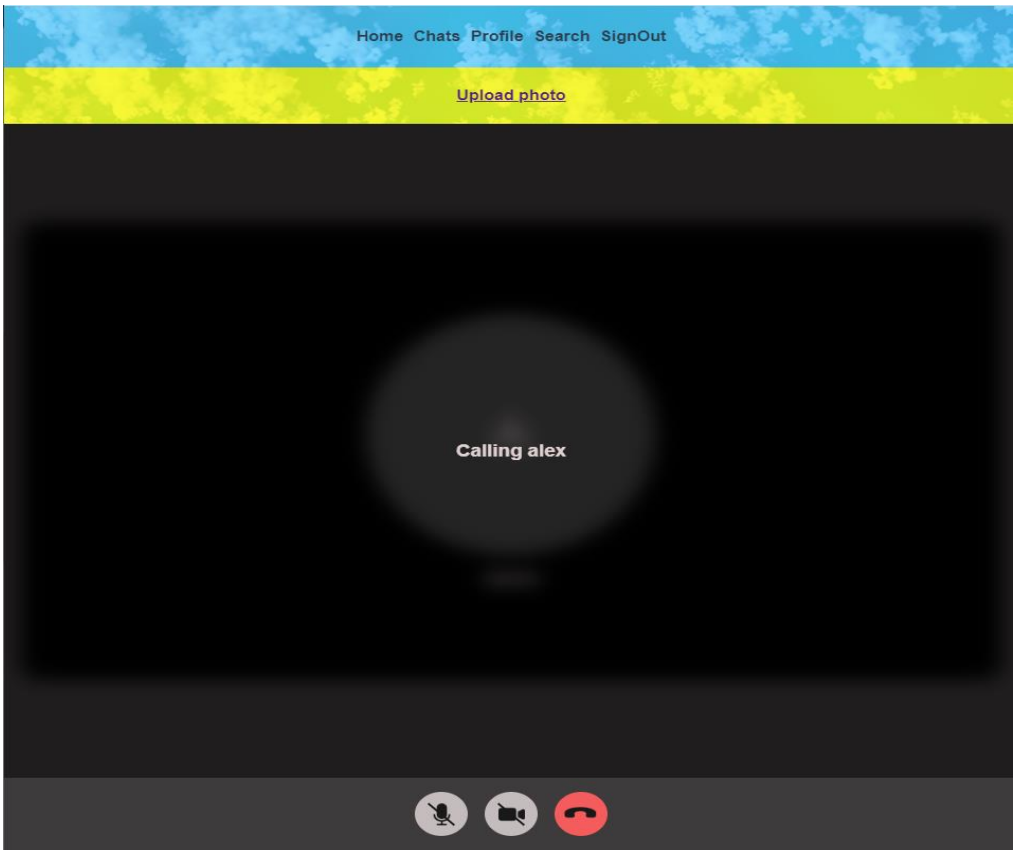


Рисунок 3.8 – Екран ініціатора, з вимкненою камерою та звуком, після початку дзвінка.

Якщо, відповіді не буде протягом однієї хвилини, то виклик буде скасовано та ініціатора буде виведено повідомлення, що час збіг і переправлено на сторінку з чатами через 5 секунд, а у одержувача автоматично закриється вікно. На рисунку 3.9 можна побачити екран ініціатора, після вичерпання однієї хвилини.

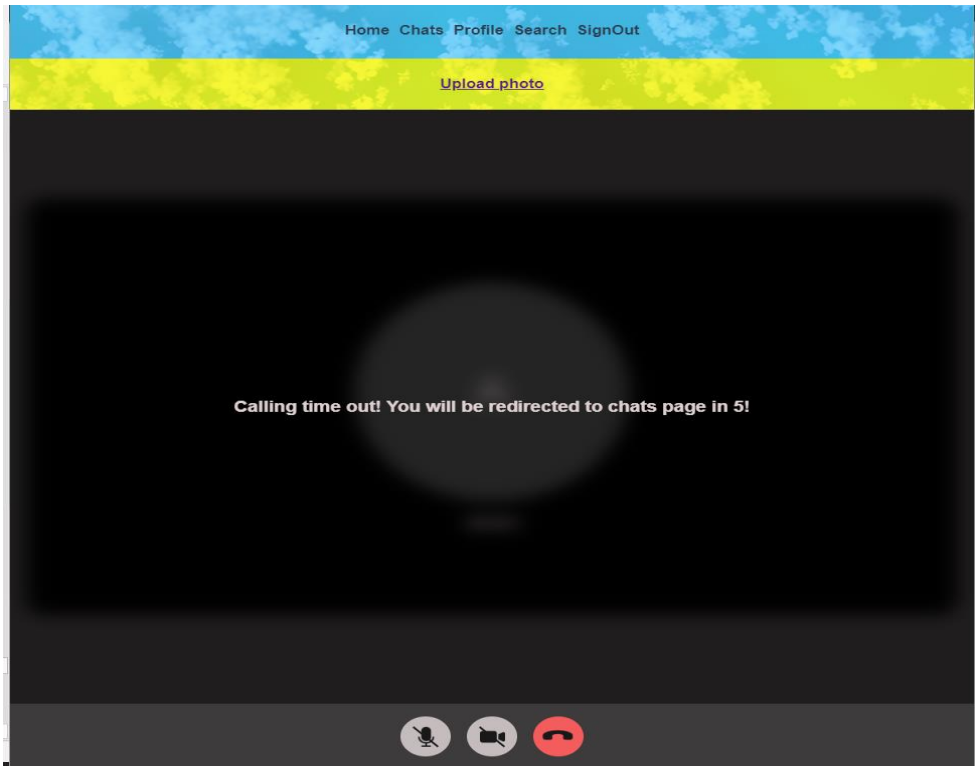


Рисунок 3.9 – Екран ініціатора, після однієї хвилини без відповіді

Якщо отримувач відповість на виклик, то його буде переправлено на сторінку дзвінка. Де він зможе розпочати спілкування, керувати власним аудіо та відео. На рисунку 3.10 можна побачити екран ініціатора, після того, як співбесідник приєднався. Локальне відео знаходиться в лівому верхньому кутку.

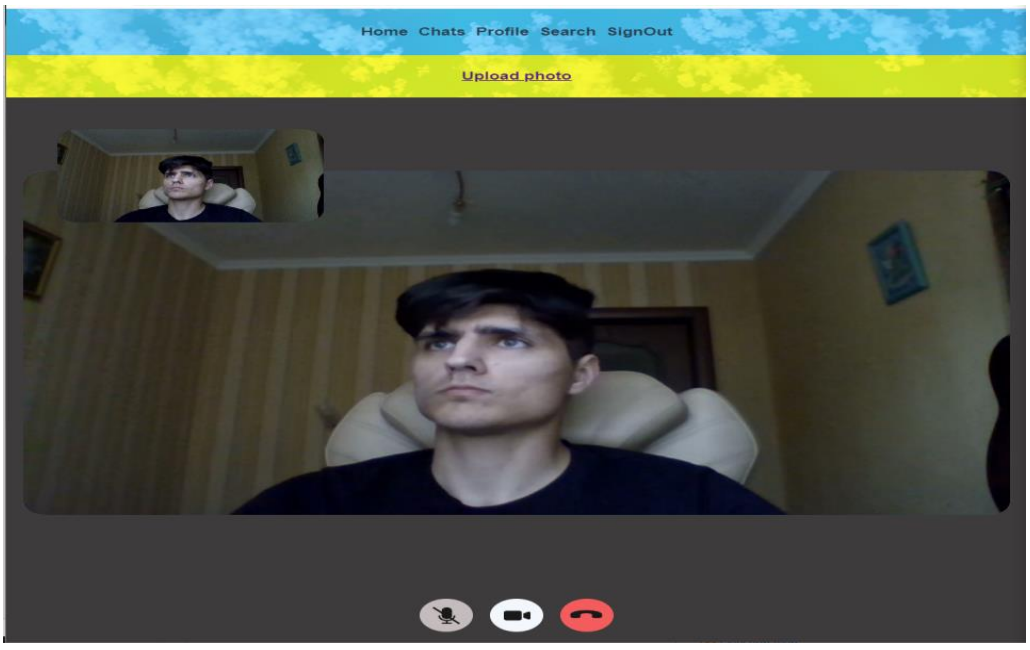


Рисунок 3.10 – Екран ініціатора, після того як співбесідник приєднався і ввімкнув камеру

Якщо хтось з учасників вимкне камеру, то на місці відео буде продемонстровано логін користувача, його фото або першу літеру логіну у кружку. На рисунку 3.11 можна побачити екран, коли отримувач вимкнув камеру.

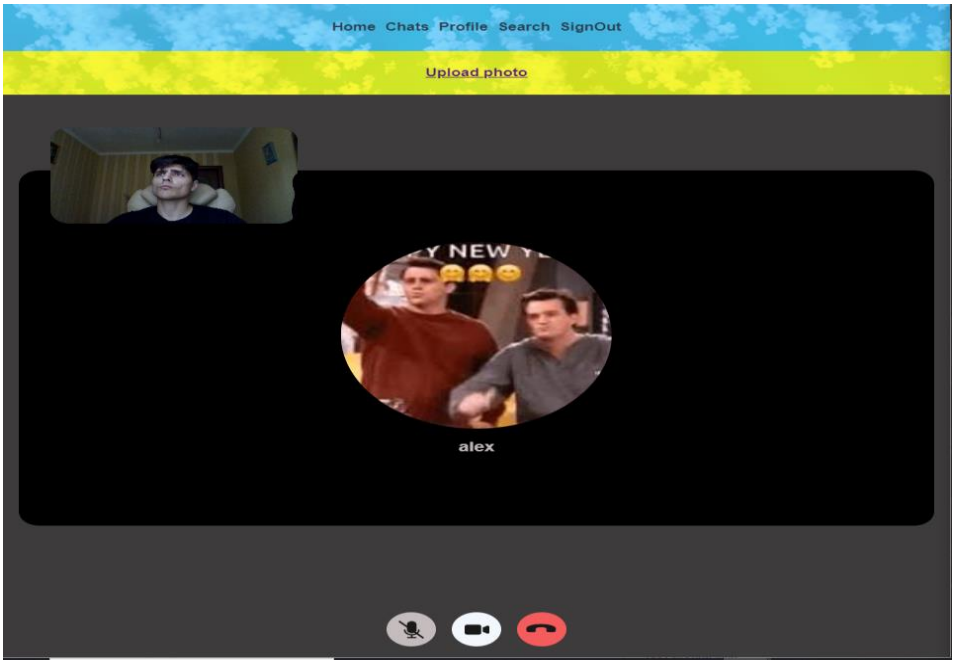


Рисунок 3.11 – Екран ініціатора, якщо отримувач не ввімкнув камеру

Якщо учасник, завершить розмову, то іншому користувачу буде виведено повідомлення про завершення дзвінка. Той хто завершив розмову буде переправлений на сторінку чатів одразу, а інший клієнт – через 5 секунд.

Висновки до розділу

У цьому розділі було проведено аналіз статичного коду серверної та клієнтської частини за допомогою веб-сервісу Embold та наведено результати. Веб-застосунок в цілому отримав не погані оцінки. Також було продемонстровано певні метрики та перевірено не функціональні вимоги.

Також було проведено мануальне тестування веб-застосунка і описано деякі тести у таблицях.

В підрозділі 3.3 було наведено контрольний приклад здійснення дзвінка. Детально описано кроки користувачів з демонстрацією результатів за допомогою рисунків.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розглянемо платформи для розгортання веб-застосунків. Render та Heroku – одні з найпопулярніших сервісів, що не потребують введення даних своєї кредитної карти під час реєстрації та здійснення тестового платежу.

Heroku – ця платформа стала досить популярною, через свою простоту та безкоштовне розгортання свої тестових проектів. Також вони мають гарну цінову політику, хоча безкоштовного плану вже немає.

На заміну йому прийшов Render. Він має досить багато переваг. По-перше, безкоштовний план, незалежно від складності проекту. Також він дозволяє серверу працювати до 100 хвилин, в той час, як Heroku вимикав його після 30 секунд без запитів [15]. Render має кращу надійність та швидкодію, бо використовує HTTP/2 та HTTP/3 [15].

Тож, клієнтську, серверну частину веб-застосунку та базу даних було розміщено на платформі Render. Вона також дозволяє зручно налаштувати автоматичне розгортання з репозиотрію на GitHub.

Сам процес розгортання починається після того, як новий код програмного забезпечення потрапляє до основної гілки. Тоді Render автоматично запускає всі потрібні команди, розпочинає збірку. Цей веб-застосунок дозволяє переглянути всі помилки, якщо вони були під час розгортання. Також потрібно було вказати деякі змінні середовища, щоб частини програмного забезпечення могли працювати один з одним.

На рисунку 4.1 - 4.3 можна побачити інформацію про розгортання клієнтської, серверної частини та бази даних на платформі Render.

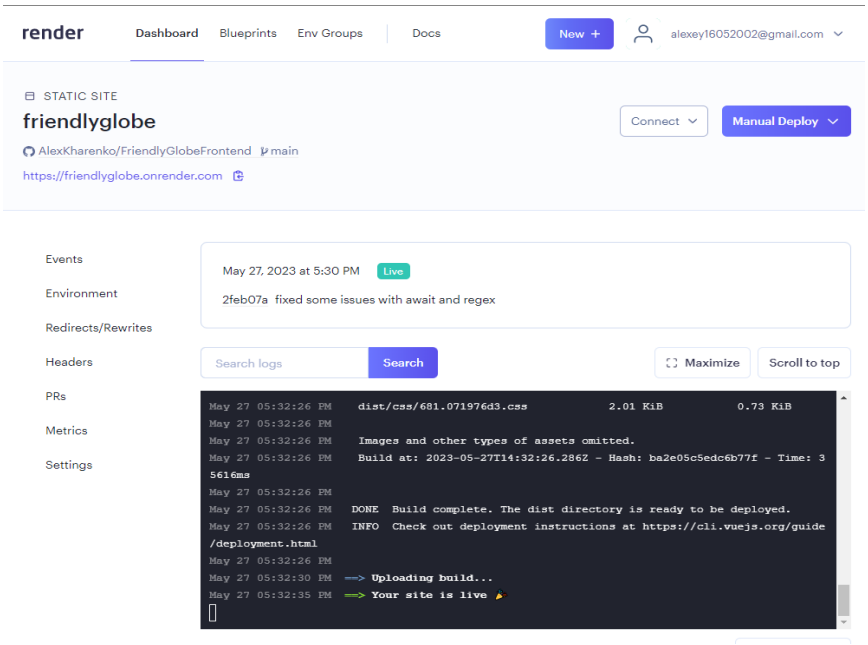


Рисунок 4.1 – Інформація про розгортання клієнтської частини

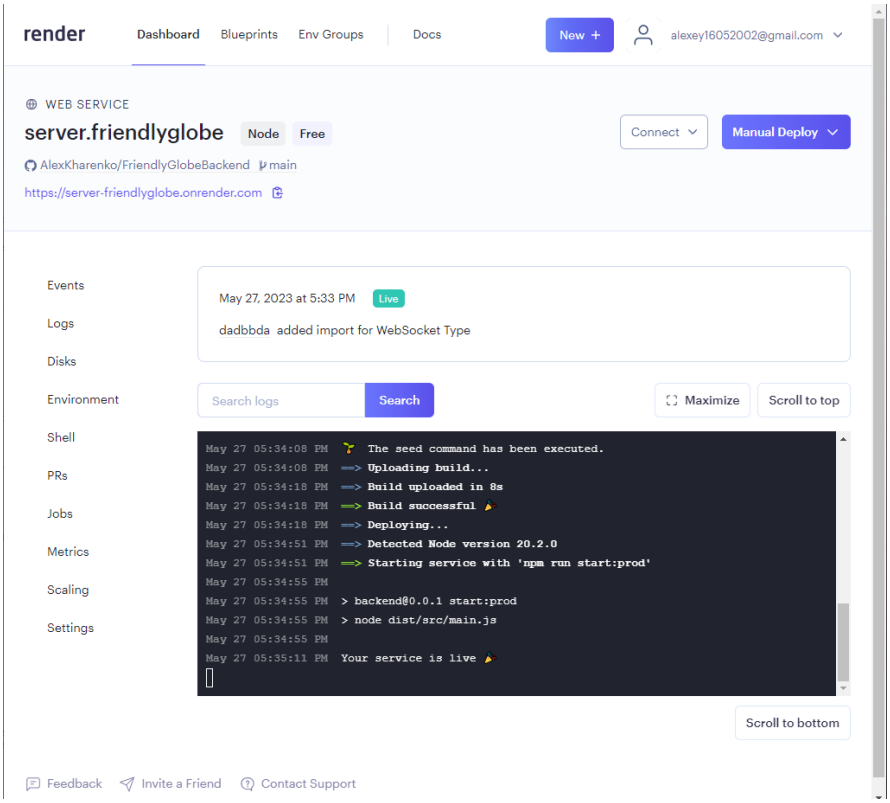


Рисунок 4.2 – Інформація про розгортання серверної частини

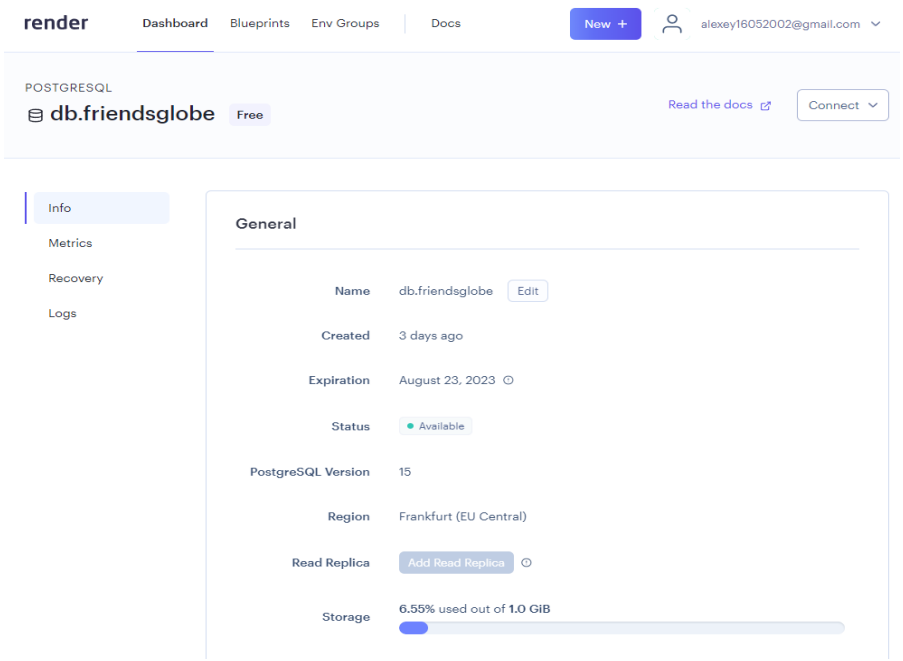


Рисунок 4.3 – Інформація про розгортання бази даних

4.2 Підтримка програмного забезпечення

Для того щоб оновити застосунок, достатньо надіслати новий код на основну гілку репозиторію клієнтської чи серверної частини. Проект збереться автоматично і буде доступний для використання у найкоротший час. Це є основною перевагою платформи Render.

Висновки до розділу

У цьому розділі було проведено аналіз платформ для розгортання програмного забезпечення та обрано Render. Вона дозволяє зручно оновлювати та підтримувати веб-застосунок. Також було описано спосіб публікації нової версії.

ВИСНОВКИ

В рамках виконання даної дипломної роботи було проведено аналіз предметної області та розроблено соціальну мережу, що націлена на пошук друзів та спілкування між ними. Також ми порівняли наш веб-застосунок зі схожими аналогами, щоб зрозуміти який функціонал слід додати.

Далі ми провели аналіз архітектури. В результаті було обрано клієнт-сервісну архітектуру для розробки веб-застосунку. Вона надає кращу підтримку коду та дозволяє краще масштабуватись. Також ще однією перевагою такого рішення є ціна, достатньо двох серверів, на відміну від мікросервісної архітектури.

Наступним кроком було обрано технології, мову програмування та фреймворки. Клієнтську частину було вирішено написати за допомогою SPA. Це дозволило збільшити інтерактивність та швидкість роботи веб застосунка та зменшити навантаження на back-end частину. Vue.js – фреймворк, що дозволив реалізувати користувацький інтерфейс. Для написання серверної частини було обрано мову Node.js з використанням фреймворку Nest.js, що працює на принципах ООП та забезпечить гарну швидкодію застосунку. Для реалізації чату у реальному часі було використано WebSocket. Nest.js має вбудовану підтримку цієї технології, що дозволило не створювати окремий сервер. Також ми використали WebRTC для забезпечення передачі відео та аудіо між користувачами у режимі реального часу.

Зберігання інформації було вирішено використати реляційну базу даних PostgreSQL, що надала кращу швидкість роботи з даними за рахунок використання декількох ядер/процесів. Для покращення швидкості розробки, було вирішено використати Prisma ORM.

Потім ми розробили діаграму використання, щоб краще зрозуміти, як користувачі будуть взаємодіяти із системою, та описали основні функціональні задачі. Також визначили нефункціональні вимоги до нашого застосунку.

Потім провели тестування програмного забезпечення. Провели аналіз статичного коду застосунку, навели оцінки деяких метрик. Також було мануально протестувати деякий функціонал веб-застосунку та представлено відтворення кожної перевірки у таблицях. Ще провели контрольний приклад на основі дзвінка з усіма розгалуженнями, описали кожен наш крок текстом з використанням зображень.

Наступним кроком, ми обрали платформу де розгорнути нашу соціальну мережу. Render – це сервіс, що дозволив нам з легкістю розмістити наш веб-застосунок.

Таким чином, всі поставлення задачі були виконані. Розроблена соціальна мережа дозволяє з легкістю шукати собі нових друзів. А реалізований функціонал для спілкування у реальному часі, спрощує взаємодію між користувачами. Зручний користувацький інтерфейс забезпечує простоту у використанні. У майбутньому планується додати можливість входу за допомогою інших сервісів та інший новий функціонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) ЩО ТАКЕ ВЕБ ДОДАТОК? РІЗНИЦЯ МІЖ САЙТОМ, ВЕБ-ДОДАТКОМ, SPA І PWA [Електронний ресурс] – 2022 – Режим доступу до ресурсу: https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/

2) Що таке Frontend і Backend? Джерело: https://kr-labs.com.ua/blog/shho-take-frontend-i-backend [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://kr-labs.com.ua/blog/shho-take-frontend-i-backend/

3) Browser [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.computerhope.com/jargon/b/browser.htm

4) What is the Difference Between SPAs, SSGs, and SSR? [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://hygraph.com/blog/difference-spa-ssg-ssr

5) 10 reasons why you should use NodeJs [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.projectpro.io/article/10-reasons-why-you-should-use-nodejs/129

6) THE 8 TOP BACK-END PROGRAMMING LANGUAGES FOR 2023 [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://blog.boot.dev/backend/best-backend-programming-languages/

7) SQL vs. NoSQL Databases: What's the Difference? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.ibm.com/cloud/blog/sql-vs-nosql

8) Difference Between WebSocket and Socket.io [Електронний ресурс] – Режим доступу до ресурсу: https://www.educba.com/websocket-vs-socket-io/.

9) What is WebRTC? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://ably.com/blog/what-is-webrtc

10) Angular vs React vs Vue: Core Differences [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.browserstack.com/guide/angular-vs-react-vs-vue

11) Angular vs React vs Vue 2023 [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://athemes.com/guides/angular-vs-react-vs-vue/

12) Express.js vs Fastify - In-Depth Comparison of the Frameworks [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.inkoop.io/blog/express-vs-fastify-in-depth-comparison-of-node-js-frameworks/

13) Why Use The NestJS Framework? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://apiumhub.com/tech-blog-barcelona/why-use-the-nestjs-framework/

14) MySQL vs PostgreSQL -- Choose the Right Database for Your Project [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres

15) Render vs Heroku [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://render.com/render-vs-heroku-comparison

Додаток А Звіт подібності

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015404328

Дата перевірки:
03.06.2023 10:12:38 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2023 10:14:18 EEST

ID користувача:
76913

Назва документа: **ІП-92_Харенко_ПЗ**

Кількість сторінок: **70** Кількість слів: **10792** Кількість символів: **84759** Розмір файлу: **3.31 MB** ID файлу: **1015068011**

115 слів позначені як "вилучені" та не враховуються у підрахунку слів

14.4%
Схожість

Найбільша схожість: **5.1%** з джерелом з Бібліотеки (ID файлу: **1015042084**)

3.17% Джерела з Інтернету	89	Сторінка 72
14.3% Джерела з Бібліотеки	200	Сторінка 73

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0.01%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за **8 слів** та **0%**)

0% Вилучення з Інтернету	5	Сторінка 74
0.01% Вилученого тексту з Бібліотеки	5	Сторінка 74

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

СОЦІАЛЬНА МЕРЕЖА ДЛЯ ПОШУКУ ДРУЗІВ ТА СПІЛКУВАННЯ

Текст програми

КПІ.ІП-9226.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_Ірина ЗЕНІВ___

Нормоконтроль:
Катерина ЛІЩУК

Виконавець:
Олексій ХАРЕНКО

Київ – 2023

Файл main.ts

```
import { NestFactory } from '@nestjs/core';
import {
  FastifyAdapter,
  NestFastifyApplication,
} from '@nestjs/platform-fastify';
import { AppModule } from './modules/app.module';
import { ValidationPipe } from '@nestjs/common';
import fastifyMultipart from '@fastify/multipart';
import cookie from '@fastify/cookie';
import * as fastifyStatic from '@fastify/static';
import { join, dirname } from 'path';
import { WsAdapter } from '@nestjs/platform-ws';

async function bootstrap() {
  const app = await NestFactory.create<NestFastifyApplication>(
    AppModule,
    new FastifyAdapter(),
  );
  app.enableCors({
    origin: [
      'http://localhost:8080',
      'http://192.168.1.6:8080',
      'http://192.168.43.126:8080',
      'http://192.168.1.56:8080',
      process.env.FRONT_END_URL,
    ],
    credentials: true,
  });
  app.register(fastifyStatic, {
    root: join(dirname(dirname(__dirname)), 'public', 'uploads'),
    prefix: '/static/',
  });
  app.useWebSocketAdapter(new WsAdapter(app));
  await app.register(cookie);
  await app.register(fastifyMultipart);
  app.useGlobalPipes(new ValidationPipe());
  await app.listen(process.env.PORT, process.env.IS_LOCALHOST ? '' : '0.0.0.0');
}
bootstrap();
```

Файл ageFilter.ts

```
export const ageFilter = (
  minAge: number | undefined,
  maxAge: number | undefined,
) => {
  const currentDate = new Date();
  const minBirthdate = minAge
    ? new Date(
        currentDate.getFullYear() - minAge,
        currentDate.getMonth(),
        currentDate.getDate(),
      )
    : null;
  const maxBirthdate = maxAge
    ? new Date(
        currentDate.getFullYear() - maxAge - 1,
        currentDate.getMonth(),
        currentDate.getDate(),
      )
    : null;

  const filter: any = {};

  if (minBirthdate) {
    filter.birthdayDate = {
      lte: minBirthdate.toISOString(),
    };
  }

  if (maxBirthdate) {
    filter.birthdayDate = {
      ...filter.birthdayDate,
      gte: maxBirthdate.toISOString(),
    };
```

```
  }

  return filter;
};
```

Файл cookieHelper.ts

```
const ACCESS_TOKEN_COOKIE_ALIVE_MS = 30 * 60;
const REFRESH_TOKEN_COOKIE_ALIVE_MS = 30 * 24 * 60 * 60;

export const setAuthCookie = (res, refreshToken, accessToken) => {
  res
    .setCookie('refreshToken', refreshToken, {
      maxAge: REFRESH_TOKEN_COOKIE_ALIVE_MS,
      httpOnly: true,
      sameSite: 'none',
      secure: true,
      path: '/',
    })
    .setCookie('accessToken', accessToken, {
      maxAge: ACCESS_TOKEN_COOKIE_ALIVE_MS,
      httpOnly: true,
      sameSite: 'none',
      secure: true,
      path: '/',
    });
};

export const removeAuthCookie = (res) => {
  res.setCookie('refreshToken', '', { httpOnly: true, path: '/', maxAge: 0 });
  res.setCookie('accessToken', '', { httpOnly: true, path: '/', maxAge: 0 });
};
```

Файл cookieParser.ts

```
export const cookieParse = (cookies): any => {
  if (!cookies) return {};
  const parsedCookies = {};
  cookies.split('; ').forEach((cookie) => {
    const [key, value] = cookie.split('=');
    parsedCookies[key] = value;
  });
  return parsedCookies;
};
```

Файл createFoldersIfNotExist.ts

```
import { InternalServerErrorException } from '@nestjs/common';
import { promises as fsPromises } from 'fs';


export const createFoldersIfNotExist = async (folderPath) => {
  try {
    await fsPromises.stat(folderPath);
  } catch (err) {
    if (err.code === 'ENOENT') {
      try {
        await fsPromises.mkdir(folderPath, { recursive: true });
      } catch (error) {
        throw new InternalServerErrorException();
      }
```

```
    } else {
      return;
    }
  }
};
```

Файл dateValidation.ts

```
export const maxDate = () => {
  const today = new Date();
  today.setFullYear(today.getFullYear() - 12);
  return today;
};
export const minDate = () => {
  const today = new Date();
  today.setFullYear(today.getFullYear() - 120);
  return today;
};
```

Файл parseArray.ts

```
export const parseArray = (array: string) => {
  if (array === undefined) return undefined;
  return JSON.parse(array);
};
```

Файл parseBool.ts

```
export const parseBool = (bool: string) => {
  if (bool === undefined) return undefined;
  return bool ? bool === 'true' : false;
};
```

Файл websocket.util.ts

```
import { Injectable } from '@nestjs/common';
import { MessageService } from 'src/services/message.service';
import { ChatService } from 'src/services/chat.service';
import { Server } from 'ws';

@Injectable()
export class WebsocketUtils {
  constructor(
    private messageService: MessageService,
    private chatService: ChatService,
  ) {}

  sendError(client, err) {
    client.send(
      JSON.stringify({ event: 'error', data: { message: err.message } }),
    );
  }

  removeUserFromServerByUserId(server, userId) {
    for (const client of server.clients) {
```

```
    if (client['user'] && client['user']?.userId == userId)
      return client.close(1008, 'NewSignIn');
  }
}

sendResponseToUser(server: Server, wsClient, receiverId, event, data) {
  const receiver = [...server.clients].find(
    (client) => client['user']?.userId == receiverId,
  );
  if (wsClient['user']?.userId == receiverId) return;
  if (receiver) receiver.send(JSON.stringify({ event, data }));
}

sendResponseToClient(wsClient, event, data) {
  if (wsClient) wsClient.send(JSON.stringify({ event, data }));
}

sendResponseToUsers(server: Server, wsClient, userIds, event, data) {
  for (const client of server.clients) {
    if (
      client['user']?.userId != wsClient['user']?.userId &&
      userIds.includes(client['user']?.userId)
    ) {
      client.send(JSON.stringify({ event, data }));
    }
  }
}

async getUnreadMessagesCountOfUser(client) {
  const count = await this.messageService.getCountOfUnreadMessages(
    +client['user'].userId,
  );
  this.sendResponseToClient(client, 'unreadMessagesCount', { count });
}

async getUserChatIds(userId) {
  const chats = await this.chatService.getChats(+userId);
  return chats.map((chat) => chat.chatId);
}

async getChatsUsersIds(userId) {
  const chats = await this.chatService.getChats(+userId);
  const chatUserIds = chats.map((chat) => {
    if (chat?.user1?.userId == userId) return chat.user2.userId;
    return chat?.user1.userId;
  });
  return chatUserIds;
}

async getOnlineUsersFromList(server, userIds) {
  const onlineClients = [...server.clients].filter(
    (client) =>
      client['user']?.userId && userIds.includes(client['user'].userId),
```

```
    );
    const onlineUserIds = onlineClients.map((client) => client['user'].userId);

    return onlineUserIds;
  }

  async readMessagesInChat(userId, payload) {
    const { chatId } = payload;
    if (!chatId) return;
    const chat = await this.chatService.getChatById(chatId);
    if (chat) {
      await this.messageService.readMessagesInChat(chatId, userId);
      const secondUserId = this.getSecondUserIdOfChat(userId, chat);
      return { chatId, secondUserId };
    }
    throw new Error("Can't find chat!");
  }

  getSecondUserIdOfChat(clientUserId, chat) {
    if (
      chat &&
      (chat.user1.userId == clientUserId || chat.user2.userId == clientUserId)
    ) {
      if (chat.user1.userId == clientUserId) return chat.user2.userId;
      return chat.user1.userId;
    }
    return null;
  }

  async handleGetMessages(userId, payload) {
    const { chatId, lastMessageTimeCreated } = payload;
    const chat = await this.chatService.getChatById(chatId);
    if (chat && (chat.user1.userId == userId || chat.user2.userId == userId)) {
      const messages = await this.messageService.getMessages(
        chatId,
        lastMessageTimeCreated,
      );
      return messages;
    }
    return [];
  }

  async handleCreateMessage(userId, payload) {
    const { receiverId, content } = payload;
    if (!content) return;
    const newMessage = await this.messageService.createMessage(
      userId,
      receiverId,
      content,
    );
    return newMessage;
  }
```

```
async handleEditMessage(userId, payload) {
  const { messageId, content } = payload;
  if (!content) return;
  const updatedMessage = await this.messageService.updateMessage(
    userId,
    messageId,
    content,
  );
  return updatedMessage;
}

async handleDeleteMessage(userId, payload) {
  const { messageId } = payload;
  const deletedMessage = await this.messageService.deleteMessage(
    userId,
    messageId,
  );
  return deletedMessage;
}

async getChatByChatId(userId, payload) {
  const { chatId } = payload;
  if (!chatId) return null;
  const chat = await this.chatService.getChatById(+chatId);
  if (!chat) return null;
  if (chat.user1.userId != userId && chat.user2.userId != userId) return null;
  return chat;
}

findClientByUserId(server, secondUserId) {
  const foundClient = [...server.clients].find(
    (item) => item['user'] && item['user'].userId == secondUserId,
  );
  return foundClient;
}

getUserFromChat(userId, chat) {
  return chat.user1.userId == userId ? chat.user1 : chat.user2;
}

// calls
getSecondUserOfCall(userId, call) {
  return call.recipient.userId == userId
    ? call.initiator.userId
    : call.recipient.userId;
}

callNotExistHandle(server, client, callsMap, chat) {
  const secondUserId = this.getSecondUserIdOfChat(
    +client['user'].userId,
    chat,
  );
  const recipientClient = this.findClientByUserId(server, secondUserId);
```

```
  const newCall = {
    initiator: this.getUserFromChat(client['user'].userId, chat),
    recipient: this.getUserFromChat(secondUserId, chat),
    timeCallInitiated: new Date(),
    status: 'callInitiated',
  };
  callsMap.set(chat.chatId, newCall);
  this.sendResponseToClient(client, 'dialing', { call: newCall });
  this.sendResponseToClient(recipientClient, 'callInitiated', {
    chatId: chat.chatId,
    ...newCall,
  });
 }
 callExistHandle(server, client, call) {
  const initiatorClient = this.findClientByUserId(
    server,
    call.initiator.userId,
  );
  call.status = 'inProgress';
  this.sendResponseToClient(client, 'connectedToCall', { call });
  this.sendResponseToClient(initiatorClient, 'recipientAnswered', {
    status: call.status,
  });
 }

 callRejectHandle(server, callMap, call, chatId) {
  const initiatorClient = this.findClientByUserId(
    server,
    call.initiator.userId,
  );
  callMap.delete(chatId);
  this.sendResponseToClient(initiatorClient, 'callRejected', {
    status: 'rejected',
  });
 }

 callEndHandle(server, client, callMap, chat) {
  const secondUserId = this.getSecondUserIdOfChat(
    +client['user'].userId,
    chat,
  );
  const secondCallClient = this.findClientByUserId(server, secondUserId);
  callMap.delete(chat.chatId);
  this.sendResponseToClient(secondCallClient, 'callEnded', {
    chatId: chat.chatId,
    status: 'ended',
  });
 }

 toggleVideoHandle(server, client, payload, call) {
  const secondUserOfCall = this.getSecondUserOfCall(
    client['user'].userId,
    call,
```

```
    );
    const secondCallClient = this.findClientByUserId(server, secondUserOfCall);
    this.sendResponseToClient(secondCallClient, 'remoteVideoToggled', {
      enabled: payload.enabled || false,
    });
  }

  async endAllCalls(server, client, callsMap) {
    const chatIds = await this.getUserChatIds(+client['user'].userId);
    const activeCallIds = this.findAllActiveUserCalls(callsMap, chatIds);
    if (chatIds.length == 0) return;
    for (const callId of activeCallIds) {
      if (callId) {
        const call = callsMap.get(callId);
        if (!call) continue;
        const callSecondUserId = this.getSecondUserOfCall(
          client['user'].userId,
          call,
        );
        const callSecondClient = this.findClientByUserId(
          server,
          callSecondUserId,
        );
        this.sendResponseToClient(callSecondClient, 'callEnded', {
          chatId: callId,
          status: 'ended',
        });
        callsMap.delete(callId);
      }
    }
  }

  findAllActiveUserCalls(callsMap, chatIds) {
    const activeCallIds = [];
    for (const chatId of chatIds) {
      const call = callsMap.get(chatId);
      if (call) activeCallIds.push(chatId);
    }
    return activeCallIds;
  }

  callAnswerHandle(server, client, callsMap, chatId, chatIds) {
    const activeCallIds = this.findAllActiveUserCalls(callsMap, chatIds);
    for (const callId of activeCallIds) {
      if (callId != chatId) {
        const call = callsMap.get(callId);
        if (!call) continue;
        const callSecondUserId = this.getSecondUserOfCall(
          client['user'].userId,
          call,
        );
        const callSecondClient = this.findClientByUserId(
          server,
```

```
        callSecondUserId,
      );
      this.sendResponseToClient(callSecondClient, 'callEnded', {
        chatId,
        status: 'ended',
      });
      callsMap.delete(callId);
    }
  }
 }

 callRTCOfferHandle(server, client, callMap, chat, offer) {
  const secondUserId = this.getSecondUserIdOfChat(
   +client['user'].userId,
   chat,
  );
  const secondCallClient = this.findClientByUserId(server, secondUserId);
  this.sendResponseToClient(secondCallClient, 'offerCreated', {
   chatId: chat.chatId,
   offer,
  });
 }

 callRTCAnswerHandle(server, client, callMap, chat, answer) {
  const secondUserId = this.getSecondUserIdOfChat(
   +client['user'].userId,
   chat,
  );
  const secondCallClient = this.findClientByUserId(server, secondUserId);
  this.sendResponseToClient(secondCallClient, 'answerCreated', {
   chatId: chat.chatId,
   answer,
  });
 }

 iceCandidateHandle(server, client, callMap, chat, candidate) {
  const secondUserId = this.getSecondUserIdOfChat(
   +client['user'].userId,
   chat,
  );
  const secondCallClient = this.findClientByUserId(server, secondUserId);
  this.sendResponseToClient(secondCallClient, 'newIceCandidate', {
   chatId: chat.chatId,
   candidate,
  });
 }
}
```

Файл auth.service.ts

```
import {
 BadRequestException,
 ForbiddenException,
 Injectable,
 UnauthorizedException,
```

```
} from '@nestjs/common';
import { UserService } from './user.service';
import { CreateUserDto } from 'src/dtos/create-user.dto';

import * as bcrypt from 'bcrypt';
import { TokenService } from './token.service';

@Injectable()
export class AuthService {
  constructor(
    private userService: UserService,
    private tokenService: TokenService,
  ) {}
  async signUp(createUserDTO: CreateUserDto) {
    const foundUserByEmail = await this.userService.findUserByEmail(
      createUserDTO.email,
    );
    if (foundUserByEmail)
      throw new BadRequestException('User with this email already exists!');
    const foundUserByUsername = await this.userService.findUserByUsername(
      createUserDTO.username,
    );
    if (foundUserByUsername)
      throw new BadRequestException('User with this username already exists!');
    const saltSize = process.env.BCRYPT_SALT;
    const hash = await bcrypt.hash(createUserDTO.password, +saltSize);
    createUserDTO.password = hash;
    await this.userService.createUser(createUserDTO);
  }
  async signIn(email: string, password: string) {
    const foundUserByEmail = await this.userService.findUserByEmail(email);
    if (!foundUserByEmail) throw new BadRequestException('Bad credentials!');
    const { password: userPassword, blocked, ...user } = foundUserByEmail;
    const isPasswordCorrect = await bcrypt.compare(password, userPassword);
    if (!isPasswordCorrect) throw new BadRequestException('Bad credentials!');
    if (blocked)
      throw new ForbiddenException(
        `Your account has been blocked! Reason: ${
          foundUserByEmail?.blockedUserMessage?.blockMessage || ''
        }`,
      );
    const { tokens } = await this.tokenGenLogic(user);
    return { ...tokens, user };
  }
  async tokenGenLogic(user) {
    const tokens = this.tokenService.generateTokens(user);
    await this.tokenService.saveRefreshToken(user.userId, tokens.refreshToken);
    return { user, tokens };
  }
  async signOut(refreshToken) {
    await this.tokenService.deleteRefreshTokenFromDB(refreshToken);
  }
  async refreshToken(refreshToken) {
    if (!refreshToken) throw new UnauthorizedException();
    const userData = this.tokenService.validateRefreshToken(refreshToken);
    if (!userData) throw new UnauthorizedException();
    const tokenFromDB = await this.tokenService.getRefreshToken(refreshToken);
    if (!tokenFromDB) throw new UnauthorizedException();

    const user = await this.userService.findUserByUserId(userData.userId);
    if (user.blocked) throw new BadRequestException('You are blocked!');
    const { tokens } = await this.tokenGenLogic(user);

    return { ...tokens, user };
  }
}
```

Файл chat.service.ts

```
import { Injectable } from '@nestjs/common';
import { PrismaService } from './prisma.service';
import { UserService } from './user.service';

@Injectable()
export class ChatService {
  constructor(
    private prismaService: PrismaService,
    private userService: UserService,
```

```
) {}

async deleteChatByUsersId(user1Id, user2Id) {
  await this.prismaService.chat.deleteMany({
    where: {
      OR: [
        { user1Id, user2Id },
        { user1Id: user2Id, user2Id: user1Id },
      ],
    },
  });
}

async deleteAllChatsWithUserId(userId) {
  await this.prismaService.chat.deleteMany({
    where: { OR: [{ user1Id: userId }, { user2Id: userId }] },
  });
}

async deleteChatByChatId(chatId) {
  await this.prismaService.chat.deleteMany({
    where: { chatId },
  });
}

async getChats(userId: number) {
  const chats = await this.prismaService.chat.findMany({
    where: {
      OR: [
        {
          user1Id: userId,
        },
        {
          user2Id: userId,
        },
      ],
    },
    select: {
      chatId: true,
      user1: {
        select: {
          userId: true,
          username: true,
          profilePhotoURL: true,
        },
      },
      user2: {
        select: {
          userId: true,
          username: true,
          profilePhotoURL: true,
        },
      },
      messages: {
        select: {
          messageId: true,
          chatId: true,
          senderId: true,
          receiverId: true,
          content: true,
          timeCreated: true,
          edited: true,
          timeEdited: true,
          read: true,
        },
        orderBy: {
          timeCreated: 'desc',
        },
        take: 1,
      },
    },
  });
  const chatsWithUnreadCount = await Promise.all(
    chats.map(async (chat) => {
      const unreadCount = await this.prismaService.message.count({
        where: {
          chatId: chat.chatId,
          receiverId: userId,
          read: false,
```

```
        },
      });

      return {
        ...chat,
        unreadCount,
      };
    }),
  );
  const sortedChats = chatsWithUnreadCount.sort(
    (a, b) =>
      b.messages[0]?.timeCreated.getTime() -
      a.messages[0]?.timeCreated.getTime(),
  );
  return sortedChats;
}

async getChatById(chatId) {
  const chat = await this.prismaService.chat.findUnique({
    where: {
      chatId,
    },
    select: {
      chatId: true,
      user1: {
        select: {
          userId: true,
          username: true,
          profilePhotoURL: true,
        },
      },
      user2: {
        select: {
          userId: true,
          username: true,
          profilePhotoURL: true,
        },
      },
    },
  });
  return chat;
}

async getChatByUsersIds(user1Id, user2Id) {
  const chat = await this.prismaService.chat.findFirst({
    where: {
      OR: [
        { user1Id, user2Id },
        { user1Id: user2Id, user2Id: user1Id },
      ],
    },
    select: {
      chatId: true,
      user1: {
        select: {
          userId: true,
          username: true,
          profilePhotoURL: true,
        },
      },
      user2: {
        select: {
          userId: true,
          username: true,
          profilePhotoURL: true,
        },
      },
    },
  });
  return chat;
}

async createChat(userId, newFriendUserId) {
  await this.userService.checkUserAvailability(userId, newFriendUserId);
  const existingChat = await this.getChatByUsersIds(userId, newFriendUserId);
  if (existingChat) return existingChat;
  const chat = await this.prismaService.chat.create({
    data: {
      user1Id: userId,
```

```
        user2Id: newFriendUserId,
      },
      select: {
        chatId: true,
        user1: {
          select: {
            userId: true,
            username: true,
            profilePhotoURL: true,
          },
        },
        user2: {
          select: {
            userId: true,
            username: true,
            profilePhotoURL: true,
          },
        },
        messages: {
          select: {
            messageId: true,
            chatId: true,
            senderId: true,
            receiverId: true,
            content: true,
            timeCreated: true,
            edited: true,
            timeEdited: true,
            read: true,
          },
          orderBy: {
            timeCreated: 'desc',
          },
          take: 1,
        },
      },
    });
    const message = await this.prismaService.message.create({
      data: {
        chatId: chat.chatId,
        senderId: userId,
        receiverId: newFriendUserId,
        content: 'Hello!(Automatical generated!)',
      },
    });
    chat.messages = [message];
    return chat;
  }
}
```

Файл lists.service.ts

```
import { Injectable } from '@nestjs/common';
import { PrismaService } from './prisma.service';

@Injectable()
export class ListsService {
  constructor(private prismaService: PrismaService) {}
  async getCountriesList() {
    return await this.prismaService.country.findMany();
  }
  async getHobbiesList() {
    return await this.prismaService.hobby.findMany();
  }
  async getLanguagesList() {
    return await this.prismaService.language.findMany();
  }
}
```

Файл message.service.ts

```
import { Injectable } from '@nestjs/common';
import { PrismaService } from './prisma.service';
import { ChatService } from './chat.service';

const MESSAGES_LIST_SIZE = 50;

@Injectable()
```

```
export class MessageService {
  constructor(
    private prismaService: PrismaService,
    private chatService: ChatService,
  ) {}

  async getMessages(chatId: number, lastMessageTimeCreated?: string) {
    let timeCreated: any = {};
    if (lastMessageTimeCreated)
      timeCreated = { lt: new Date(lastMessageTimeCreated).toISOString() };
    const messages = await this.prismaService.message.findMany({
      where: {
        chatId,
        timeCreated,
      },
      orderBy: {
        timeCreated: 'desc',
      },
      take: MESSAGES_LIST_SIZE,
    });
    return messages;
  }

  async getCountOfUnreadMessages(userId) {
    const unreadCount = await this.prismaService.message.count({
      where: {
        receiverId: userId,
        read: false,
      },
    });

    return unreadCount;
  }

  async getMessageById(messageId: string) {
    const message = await this.prismaService.message.findFirst({
      where: { messageId },
    });
    return message;
  }

  async deleteMessage(senderId: number, messageId: string) {
    const message = await this.getMessageById(messageId);
    if (!message) throw new Error('No message found!');
    if (message.senderId !== senderId)
      throw new Error('You can not delete not your own messages!');
    const deletedMessage = await this.prismaService.message.delete({
      where: { messageId },
    });
    return deletedMessage;
  }

  async readMessagesInChat(chatId: number, receiverId: number) {
    await this.prismaService.message.updateMany({
      where: { chatId, receiverId, read: false },
      data: { read: true },
    });
  }

  async updateMessage(senderId: number, messageId: string, messageContent) {
    const message = await this.getMessageById(messageId);
    if (!message) throw new Error('No message found!');
    if (message.senderId !== senderId)
      throw new Error('Only sender can change the message!');
    const editedMessage = await this.prismaService.message.update({
      where: { messageId },
      data: {
        edited: true,
        timeEdited: new Date(),
        content: messageContent,
      },
    });
    return editedMessage;
  }

  async createMessage(senderId: number, receiverId: number, message: string) {
    const chatByUsersId = await this.chatService.getChatByUsersIds(
      senderId,
      receiverId,
```

```
    );
    if (!chatByUsersId) throw new Error("Can't find chat!");
    const newMessage = await this.prismaService.message.create({
      data: {
        chatId: chatByUsersId.chatId,
        senderId,
        receiverId,
        content: message,
      },
    });
    return newMessage;
  }
}
```

Файл prisma.service.ts

```
import { INestApplication, Injectable } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService extends PrismaClient {
  async enableShutdownHooks(app: INestApplication) {
    this.$on('beforeExit', async () => {
      await app.close();
    });
  }
}
```

Файл token.service.ts

```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { PrismaService } from './prisma.service';

const ACCESS_TOKEN_ALIVE = '30m';
const REFRESH_TOKEN_ALIVE = '30d';

@Injectable()
export class TokenService {
  constructor(
    private jwtService: JwtService,
    private prismaService: PrismaService,
  ) {}
  generateTokens(payload) {
    const accessToken = this.jwtService.sign(payload, {
      secret: process.env.JWT_ACCESS_TOKEN_SECRET,
      expiresIn: ACCESS_TOKEN_ALIVE,
    });
    const refreshToken = this.jwtService.sign(payload, {
      secret: process.env.JWT_REFRESH_TOKEN_SECRET,
      expiresIn: REFRESH_TOKEN_ALIVE,
    });
    return { accessToken, refreshToken };
  }
  validateRefreshToken(refreshToken) {
    try {
      const userData = this.jwtService.verify(refreshToken, {
        secret: process.env.JWT_REFRESH_TOKEN_SECRET,
      });
      return userData;
    } catch (error) {
      return null;
    }
  }
  validateAccessToken(accessToken) {
    try {
      const userData = this.jwtService.verify(accessToken, {
        secret: process.env.JWT_ACCESS_TOKEN_SECRET,
      });
      return userData;
    } catch (error) {
      return null;
    }
  }

  async createRefreshTokenInDB(userId, refreshToken) {
    await this.prismaService.refreshToken.create({
      data: {
```

```
      refreshToken,
      userId: userId,
    },
  });
 }

 async deleteRefreshTokenFromDB(refreshToken: string) {
  await this.prismaService.refreshToken.delete({ where: { refreshToken } });
 }

 async deleteRefreshTokenFromDBByUserId(userId) {
  await this.prismaService.refreshToken.delete({ where: { userId } });
 }

 async getRefreshTokenByUserId(userId) {
  return await this.prismaService.refreshToken.findUnique({
   where: { userId },
  });
 }

 async getRefreshToken(refreshToken) {
  return await this.prismaService.refreshToken.findUnique({
   where: { refreshToken },
  });
 }

 async saveRefreshToken(userId, refreshToken) {
  await this.prismaService.refreshToken.upsert({
   where: {
    userId,
   },
   update: {
    refreshToken,
   },
   create: {
    userId,
    refreshToken,
   },
  });
 }
}
```

Файл user.service.ts

```
import {
 BadRequestException,
 Injectable,
 NotFoundException,
} from '@nestjs/common';
import { PrismaService } from './prisma.service';
import { CreateUserDto } from 'src/dtos/create-user.dto';
import { Sex } from '@prisma/client';
import {
 AdminAllUsersSelect,
 AllUsersSelect,
} from 'src/interfaces/user.interface';
import { v4 } from 'uuid';
import { createWriteStream } from 'fs';
import { extname, join, dirname } from 'path';
import * as fs from 'fs';
import { ageFilter } from 'src/utils/ageFilter';
import { UpdateUserDto } from 'src/dtos/update-user.dto';
import * as bcrypt from 'bcrypt';
import * as util from 'util';
import { pipeline } from 'stream';
import { createFoldersIfNotExist } from 'src/utils/createFoldersIfNotExist';
const pump = util.promisify(pipeline);

const PAGE_SIZE = 10;

@Injectable()
export class UserService {
 constructor(private prismaService: PrismaService) {}
 excludeUserFields(user, keys) {
  for (const key of keys) {
   delete user[key];
  }
  return user;
```

```
}

async getAllUsers(
  userId,
  filters: {
    verified?: boolean;
    blocked?: boolean;
    hidden?: boolean;
  },
  searchString,
  orderBy,
  page,
) {
  const users = await this.prismaService.user.findMany({
    where: {
      ...filters,
      OR: [
        {
          username: { contains: searchString },
        },
        { email: { contains: searchString } },
      ],
      NOT: {
        userId,
      },
    },
    skip: (page - 1) * PAGE_SIZE,
    take: PAGE_SIZE,
    orderBy: { createdAt: orderBy },
    select: {
      ...AdminAllUsersSelect,
    },
  });
  const count = await this.prismaService.user.count({
    where: {
      ...filters,
      NOT: {
        userId,
      },
    },
  });
  return {
    users,
    count,
  };
}

async getUsers(userId, filters, orderBy, page) {
  const {
    searchString,
    countryList,
    languageList,
    hobbyList,
    minAge,
    maxAge,
    sexId,
  } = filters;
  const users = await this.prismaService.user.findMany({
    where: {
      AND: [
        {
          username: {
            contains: searchString,
          },
        },
        {
          languages: {
            some: {
              languageId: {
                in: languageList,
              },
            },
          },
        },
        {
          hobbies: {
            some: {
              hobbyId: {
                in: hobbyList,
```

```
            },
          },
        },
      },
      {
        country: { countryId: { in: countryList } },
      },
      ageFilter(minAge, maxAge),
      { sexId },
    ],

    verified: true,
    blocked: false,
    hidden: false,
    NOT: {
      OR: [
        { blockedByUsers: { some: { blockedUserId: userId } } },
        { blocksUsers: { some: { blockedByUserId: userId } } },
      ],
    },
  },
  skip: (page - 1) * PAGE_SIZE,
  take: PAGE_SIZE,
  orderBy: { createdAt: orderBy },
  select: { ...AllUsersSelect },
});
const count = await this.prismaService.user.count({
  where: {
    AND: [
      {
        username: {
          contains: searchString,
        },
      },
      {
        languages: {
          some: {
            languageId: {
              in: languageList,
            },
          },
        },
      },
      {
        hobbies: {
          some: {
            hobbyId: {
              in: hobbyList,
            },
          },
        },
      },
      {
        country: { countryId: { in: countryList } },
      },
      { sexId },
      ageFilter(minAge, maxAge),
    ],
    verified: true,
    blocked: false,
    hidden: false,
    NOT: {
      OR: [
        { blockedByUsers: { some: { blockedUserId: userId } } },
        { blocksUsers: { some: { blockedByUserId: userId } } },
      ],
    },
  },
});
return {
  users,
  count,
};
}

async getUsersFromBlackList(userId) {
  const users = await this.prismaService.user.findMany({
    where: {
      blocked: false,
```

```
      blocksUsers: {
        some: {
          blockedByUserId: userId,
        },
      },
    },
    select: { ...AllUsersSelect },
  });

  return users;
}

async blockUser(userId, blockMessage) {
  const data: any = {};
  if (blockMessage) {
    data.blockedUserMessage = {
      upsert: { create: { blockMessage }, update: { blockMessage } },
    };
  }
  await this.prismaService.chat.deleteMany({
    where: { OR: [{ user1Id: userId }, { user2Id: userId }] },
  });
  await this.prismaService.refreshToken.deleteMany({ where: { userId } });
  return await this.prismaService.user.update({
    where: { userId },
    data: { blocked: true, ...data },
  });
}

async unblockUser(userId) {
  await this.prismaService.user.update({
    where: { userId },
    data: { blocked: false },
  });
  return await this.prismaService.blockedUserMessage.deleteMany({
    where: { userId },
  });
}

async verifyUser(userId) {
  return await this.prismaService.user.update({
    where: { userId },
    data: { verified: true },
  });
}

async blacklistUser(blockedUserId, blockedByUserId) {
  await this.prismaService.blackList.create({
    data: {
      blockedUser: { connect: { userId: blockedUserId } },
      blockedByUser: { connect: { userId: blockedByUserId } },
    },
  });
  await this.prismaService.chat.deleteMany({
    where: {
      OR: [
        { user1Id: blockedUserId, user2Id: blockedByUserId },
        { user1Id: blockedByUserId, user2Id: blockedUserId },
      ],
    },
  });
}

async removeUserFromBlackList(blockedUserId, blockedByUserId) {
  try {
    await this.prismaService.blackList.delete({
      where: {
        blockedUserId_blockedByUserId: {
          blockedUserId,
          blockedByUserId,
        },
      },
    });
  } catch (error) {
    if (error.code === 'P2025') {
      throw new BadRequestException(
        `User with id ${blockedUserId} not found`,
      );
    }
```

```
  }
}

async findUserByEmail(email: string) {
  return await this.prismaService.user.findUnique({
    where: { email },
    select: {
      userId: true,
      username: true,
      email: true,
      password: true,
      blocked: true,
      verified: true,
      hidden: true,
      profilePhotoURL: true,
      blockedUserMessage: { select: { blockMessage: true } },
      role: true,
    },
  });
}

async findUserByUserId(userId: number) {
  const user = await this.prismaService.user.findUnique({
    where: { userId },
    select: {
      userId: true,
      username: true,
      email: true,
      blocked: true,
      verified: true,
      hidden: true,
      profilePhotoURL: true,
      role: true,
    },
  });
  return user;
}

async findUserByUsername(username: string) {
  const user = await this.prismaService.user.findUnique({
    where: { username },
    include: {
      country: true,
      userDetails: true,
      languages: true,
      hobbies: true,
    },
  });
  if (!user) return null;
  return this.excludeUserFields(user, ['password', 'role']);
}

async checkUserAvailability(userId, lookingForUserId) {
  const isAllowed = await this.prismaService.blackList.findMany({
    where: {
      OR: [
        {
          blockedUserId: lookingForUserId,
          blockedByUserId: userId,
        },
        {
          blockedUserId: userId,
          blockedByUserId: lookingForUserId,
        },
      ],
    },
  });
  const { verified, blocked, hidden } =
    await this.prismaService.user.findUnique({
      where: { userId: lookingForUserId },
      select: { verified: true, blocked: true, hidden: true },
    });

  if (
    (isAllowed.length !== 0 || !verified || blocked || hidden) &&
    userId !== lookingForUserId
  ) {
    throw new NotFoundException('User not found');
  }
```

```
    return;
  }

  async createUser(createUserDTO: CreateUserDto): Promise<{ userId: number }> {
    return await this.prismaService.user.create({
      data: {
        firstName: createUserDTO.firstName,
        secondName: createUserDTO.secondName,
        username: createUserDTO.username,
        email: createUserDTO.email,
        password: createUserDTO.password,
        sexId: Sex[createUserDTO.sexId],
        birthdayDate: new Date(createUserDTO.birthdayDate),
        country: { connect: { countryId: createUserDTO.country.countryId } },
        languages: {
          connect: [
            ...createUserDTO.languages.map((language) => {
              return { languageId: language.languageId };
            }),
          ],
        },
        hobbies: {
          connect: [
            ...createUserDTO.hobbies.map((hobby) => {
              return { hobbyId: hobby.hobbyId };
            }),
          ],
        },
        userDetails: {
          create: {
            bio: createUserDTO.bio,
            lookingForText: createUserDTO.lookingForText,
          },
        },
      },
      select: {
        userId: true,
      },
    });
  }

  async updateUser(
    userId,
    updateUserDTO: UpdateUserDto,
  ): Promise<{ userId: number }> {
    return await this.prismaService.user.update({
      where: { userId },
      data: {
        firstName: updateUserDTO.firstName,
        secondName: updateUserDTO.secondName,
        sexId: Sex[updateUserDTO.sexId],
        birthdayDate: new Date(updateUserDTO.birthdayDate),
        country: { connect: { countryId: updateUserDTO.country.countryId } },
        verified: false,
        languages: {
          connect: [
            ...updateUserDTO.languages.map((language) => {
              return { languageId: language.languageId };
            }),
          ],
        },
        hobbies: {
          connect: [
            ...updateUserDTO.hobbies.map((hobby) => {
              return { hobbyId: hobby.hobbyId };
            }),
          ],
        },
        userDetails: {
          update: {
            bio: updateUserDTO.bio,
            lookingForText: updateUserDTO.lookingForText,
          },
        },
      },
    });
  }

  async updatePassword(userId: number, password: string) {
```

```
    const saltSize = process.env.BCRYPT_SALT;
    const hashedPassword = await bcrypt.hash(password, +saltSize);

    await this.prismaService.user.update({
      where: { userId },
      data: {
        password: hashedPassword,
      },
    });
  }

  async updateProfileVisibility(userId: number, hidden: boolean) {
    await this.prismaService.user.update({
      where: { userId },
      data: {
        hidden,
      },
    });
  }

  async deleteProfile(userId: number) {
    const { profilePhotoURL } = await this.prismaService.user.findUnique({
      where: { userId },
      select: { profilePhotoURL: true },
    });
    await this.prismaService.user.deleteMany({
      where: { userId },
    });

    if (profilePhotoURL) await this.removeProfilePhotoFile(profilePhotoURL);
  }

  async deleteProfilePhoto(userId: number) {
    const { profilePhotoURL } = await this.prismaService.user.findUnique({
      where: { userId },
      select: { profilePhotoURL: true },
    });
    await this.prismaService.user.update({
      where: { userId },
      data: {
        profilePhotoURL: null,
      },
    });
    if (profilePhotoURL) await this.removeProfilePhotoFile(profilePhotoURL);
  }

  async removeProfilePhotoFile(profielPhotoURL: string) {
    const parentDir = dirname(dirname(dirname(__dirname)));
    await fs.unlink(
      join(parentDir, 'public', 'uploads', 'images', profielPhotoURL),
      (err) => {
        return err;
      },
    );
  }
  async createUserProfilePhoto(userId: number, profilePhoto: any) {
    const user = await this.findUserByUserId(userId);
    if (!user) throw new BadRequestException('User not found');
    if (user.profilePhotoURL) {
      await this.removeProfilePhotoFile(user.profilePhotoURL);
    }
    const filename = `${v4()}${extname(profilePhoto.filename)}`;
    const parentDir = dirname(dirname(dirname(__dirname)));
    const folderPath = join(parentDir, 'public', 'uploads', 'images');
    await createFoldersIfNotExist(folderPath);
    const stream = createWriteStream(join(folderPath, filename));
    await pump(profilePhoto.file, stream);
    await this.prismaService.user.update({
      where: { userId },
      data: { profilePhotoURL: filename },
    });
    return filename;
  }
}
```

Файл user.interface.ts

```
export const AdminUserSelect = {
```

```
  userId: true,
  firstName: true,
  secondName: true,
  username: true,
  email: true,
  sexId: true,
  birthdayDate: true,
  createdAt: true,
  profilePhotoURL: true,
  role: true,
  blocked: true,
  verified: true,
  hidden: true,
  country: true,
  userDetails: true,
  languages: true,
  hobbies: true,
};

export const AdminAllUsersSelect = {
  userId: true,
  username: true,
  email: true,
  role: true,
  createdAt: true,
  blocked: true,
  verified: true,
  hidden: true,
  blockedUserMessage: { select: { blockMessage: true } },
};

export const UserSelect = {
  userId: true,
  firstName: true,
  secondName: true,
  username: true,
  sexId: true,
  birthdayDate: true,
  createdAt: true,
  profilePhotoURL: true,
  country: true,
  userDetails: true,
  languages: true,
  hobbies: true,
  hidden: true,
  blocked: true,
  verified: true,
};

export const AllUsersSelect = {
  userId: true,
  firstName: true,
  secondName: true,
  username: true,
  sexId: true,
  birthdayDate: true,
  profilePhotoURL: true,
  country: true,
  userDetails: true,
};
```

Файл admin.guard.ts

```
import {
  Injectable,
  CanActivate,
  ExecutionContext,
  ForbiddenException,
} from '@nestjs/common';

@Injectable()
export class AdminGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const req = context.switchToHttp().getRequest();
    if (req?.isAdmin) return true;
    throw new ForbiddenException('You have no permisions!');
  }
}
```

Файл auth.guard.ts

```
import {
  Injectable,
  Inject,
  CanActivate,
  ExecutionContext,
  ForbiddenException,
  UnauthorizedException,
} from '@nestjs/common';
import { TokenService } from 'src/services/token.service';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(
    @Inject(TokenService)
    private readonly tokenService: TokenService,
  ) {}
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const req = context.switchToHttp().getRequest();
    const { accessToken } = req?.cookies;
    if (accessToken) {
      const payload = await this.validate(accessToken);
      if (payload) {
        req.payload = payload;
        req.isAdmin = payload.role === 'ADMIN' ? true : false;
        return true;
      }
    }
    throw new UnauthorizedException();
  }
  async validate(token: string) {
    try {
      return await this.tokenService.validateAccessToken(token);
    } catch (e) {
      throw new ForbiddenException();
    }
  }
}
```

Файл blocked.guard.ts

```
import {
  Injectable,
  CanActivate,
  ExecutionContext,
  ForbiddenException,
} from '@nestjs/common';

@Injectable()
export class BlockedGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const req = context.switchToHttp().getRequest();
    if (!req?.payload?.blocked) return true;

    throw new ForbiddenException('You have no permisions!');
  }
}
```

Файл verified.guard.ts

```
import {
  Injectable,
  CanActivate,
  ExecutionContext,
  ForbiddenException,
} from '@nestjs/common';

@Injectable()
export class VerifiedGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const req = context.switchToHttp().getRequest();
    if (req?.payload?.verified) return true;

    throw new ForbiddenException('You have no permisions!');
  }
}
```

Файл addToBlacklist.dto.ts

```
import { IsNotEmpty, IsNumber } from 'class-validator';

export class AddToBlacklistDto {
  @IsNotEmpty()
  @IsNumber()
  blockedUserId: number;
}
```

Файл country.dto.ts

```
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class CountryDto {
  @IsNotEmpty()
  @IsNumber()
  countryId: number;

  @IsNotEmpty()
  @IsString()
  countryName: string;

  @IsNotEmpty()
  @IsString()
  countryCode: string;
}
```

Файл create-user.dto.ts

```
import { Transform, Type } from 'class-transformer';
import {
  IsEmail,
  IsNotEmpty,
  IsString,
  MinLength,
  MaxLength,
  ValidateNested,
  IsArray,
  IsOptional,
  MinDate,
  MaxDate,
  IsDate,
  ValidateIf,
} from 'class-validator';
import { CountryDto } from './country.dto';
import { LanguageDto } from './language.dto';
import { HobbyDto } from './hobby.dto';
import { maxDate, minDate } from 'src/utils/dateValidation';

export class CreateUserDto {
  @IsString()
  @IsNotEmpty()
  firstName: string;

  @IsString()
  @IsNotEmpty()
  secondName: string;

  @IsString()
  @IsNotEmpty()
  username: string;

  @IsEmail()
  @IsNotEmpty()
  email: string;

  @IsString()
  @IsNotEmpty()
  @MinLength(6)
  @MaxLength(18)
  password: string;

  @IsDate()
  @IsNotEmpty()
  @Transform(({ value }) => new Date(value))
```

```
  @MinDate(minDate())
  @MaxDate(maxDate())
  birthdayDate: Date;

  @IsNotEmpty()
  @IsString()
  sexId: string;

  @IsNotEmpty()
  @Type(() => CountryDto)
  @ValidateNested()
  country: CountryDto;

  @IsString()
  @IsNotEmpty()
  @MinLength(100)
  @MaxLength(1000)
  bio: string;

  @IsOptional()
  @ValidateIf((dto) => dto.lookingForText)
  @IsString()
  @MaxLength(1000)
  @MinLength(100)
  lookingForText?: string;

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => LanguageDto)
  languages: LanguageDto[];

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => HobbyDto)
  hobbies: HobbyDto[];
}
```

Файл hobby.dto.ts

```
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class HobbyDto {
  @IsNotEmpty()
  @IsNumber()
  hobbyId: number;

  @IsNotEmpty()
  @IsString()
  hobby: string;
}
```

Файл language.dto.ts

```
import { IsNotEmpty, IsNumber, IsString } from 'class-validator';

export class LanguageDto {
  @IsNotEmpty()
  @IsNumber()
  languageId: number;

  @IsNotEmpty()
  @IsString()
  language: string;
}
```

Файл update-user-hidden.dto.ts

```
import { IsNotEmpty, IsBoolean } from 'class-validator';

export class UpdateUserHiddenDto {
  @IsBoolean()
  @IsNotEmpty()
  hidden: boolean;
}
```

Файл update-user-password.dto.ts

```
import { IsNotEmpty, IsString, MinLength, MaxLength } from 'class-validator';

export class UpdateUserPasswordDto {
  @IsString()
  @IsNotEmpty()
  @MinLength(6)
  @MaxLength(18)
  password: string;
}
```

Файл update-user.dto.ts

```
import { Transform, Type } from 'class-transformer';
import {
  IsNotEmpty,
  IsString,
  MinLength,
  MaxLength,
  ValidateNested,
  IsArray,
  IsOptional,
  MinDate,
  MaxDate,
  IsDate,
  ValidateIf,
} from 'class-validator';
import { CountryDto } from './country.dto';
import { LanguageDto } from './language.dto';
import { HobbyDto } from './hobby.dto';
import { maxDate, minDate } from 'src/utils/dateValidation';

export class UpdateUserDto {
  @IsString()
  @IsNotEmpty()
  firstName: string;

  @IsString()
  @IsNotEmpty()
  secondName: string;

  @IsDate()
  @IsNotEmpty()
  @Transform(({ value }) => new Date(value))
  @MinDate(minDate())
  @MaxDate(maxDate())
  birthdayDate: Date;

  @IsNotEmpty()
  @IsString()
  sexId: string;

  @IsNotEmpty()
  @Type(() => CountryDto)
  @ValidateNested()
  country: CountryDto;

  @IsString()
  @IsNotEmpty()
  @MinLength(100)
  @MaxLength(1000)
  bio: string;

  @IsOptional()
  @ValidateIf((dto) => dto.lookingForText)
  @IsString()
  @MaxLength(1000)
  @MinLength(100)
  lookingForText?: string;

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => LanguageDto)
  languages: LanguageDto[];

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => HobbyDto)
  hobbies: HobbyDto[];
```

```
}
```

Файл auth.controller.ts

```
import { Body, Controller, Post, Res, Req } from '@nestjs/common';
import { CreateUserDto } from 'src/dtos/create-user.dto';
import { AuthService } from 'src/services/auth.service';
import { removeAuthCookie, setAuthCookie } from 'src/utils/cookieHelper';

@Controller()
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('signup')
  async SignUp(@Body() body: CreateUserDto) {
    await this.authService.signUp(body);
    return { success: true };
  }
  @Post('signin')
  async SignIn(@Body() body, @Res({ passthrough: true }) res) {
    const { email, password } = body;
    const { refreshToken, accessToken, user } = await this.authService.signIn(
      email,
      password,
    );
    setAuthCookie(res, refreshToken, accessToken);
    res.status(200);
    return { success: true, user };
  }
  @Post('signout')
  async logout(@Req() req, @Res({ passthrough: true }) res) {
    const { refreshToken } = req?.cookies;
    await this.authService.signOut(refreshToken);
    removeAuthCookie(res);
    res.status(200);
    return { success: true };
  }
  @Post('auth/refresh')
  async refreshToken(@Req() req, @Res({ passthrough: true }) res) {
    const cookies = req.cookies;
    const { refreshToken, accessToken, user } =
      await this.authService.refreshToken(cookies.refreshToken);

    setAuthCookie(res, refreshToken, accessToken);
    res.status(200);
    return { success: true, user };
  }
}
```

Файл chat.controller.ts

```
import {
  Controller,
  Get,
  UseGuards,
  Param,
  Req,
  Post,
  Body,
  Delete,
  BadRequestException,
} from '@nestjs/common';
import { AuthGuard } from 'src/guards/auth.guard';
import { BlockedGuard } from 'src/guards/blocked.guard';
import { VerifiedGuard } from 'src/guards/verified.guard';
import { ChatService } from 'src/services/chat.service';

@Controller('/chats')
export class ChatController {
  constructor(private readonly chatService: ChatService) {}

  @UseGuards(AuthGuard, VerifiedGuard, BlockedGuard)
  @Get('/')
  async getChats(@Req() req) {
    const chats = await this.chatService.getChats(req.payload.userId);

    return { success: true, chats };
  }
```

```
  @UseGuards(AuthGuard, BlockedGuard)
  @Post('/')
  async createChat(@Req() req, @Body() body) {
    const { newFriendUserId } = body;
    if (req.payload.userId == +newFriendUserId)
      throw new BadRequestException('You can`t create chat with yourself');
    const chat = await this.chatService.createChat(
      req.payload.userId,
      +newFriendUserId,
    );

    return { success: true, chat };
  }

  @UseGuards(AuthGuard, BlockedGuard)
  @Delete('/:chatId')
  async deleteChat(@Req() req, @Param('chatId') chatId) {
    await this.chatService.deleteChatByChatId(+chatId);

    return { success: true };
  }
}
```

Файл lists.controller.ts

```
import { Controller, Get } from '@nestjs/common';
import { ListsService } from 'src/services/lists.service';

@Controller('lists')
export class ListsController {
  constructor(private readonly listsService: ListsService) {}

  @Get('chl')
  async getCHLLists() {
    const countries = await this.listsService.getCountriesList();
    const hobbies = await this.listsService.getHobbiesList();
    const languages = await this.listsService.getLanguagesList();
    return { success: true, countries, hobbies, languages };
  }
}
```

Файл user.controller.ts

```
import {
  Controller,
  Get,
  UseGuards,
  Param,
  NotFoundException,
  Query,
  Req,
  Post,
  Body,
  Delete,
  BadRequestException,
  ForbiddenException,
  Patch,
  Res,
} from '@nestjs/common';
import { UserService } from 'src/services/user.service';
import { AuthGuard } from 'src/guards/auth.guard';
import { parseBool } from 'src/utils/parseBool';
import { parseArray } from 'src/utils/parseArray';
import { AdminGuard } from 'src/guards/admin.guard';
import { VerifiedGuard } from 'src/guards/verified.guard';
import { BlockedGuard } from 'src/guards/blocked.guard';
import { UpdateUserHiddenDto } from 'src/dtos/update-user-hidden.dto';
import { UpdateUserPasswordDto } from 'src/dtos/update-user-password.dto copy';
import { UpdateUserDto } from 'src/dtos/update-user.dto';
import { removeAuthCookie } from 'src/utils/cookieHelper';
import { AddToBlacklistDto } from 'src/dtos/addToBlacklist.dto';

@Controller()
export class UserController {
  constructor(private readonly userService: UserService) {}

  @UseGuards(AuthGuard, BlockedGuard)
```

```
@Patch('/users/profile/hidden')
async updateProfileVisibility(@Req() req, @Body() body: UpdateUserHiddenDto) {
  await this.userService.updateProfileVisibility(
    req.payload.userId,
    body.hidden,
  );

  return { success: true };
}

@UseGuards(AuthGuard, BlockedGuard)
@Patch('/users/profile/password')
async updatePassword(@Req() req, @Body() body: UpdateUserPasswordDto) {
  await this.userService.updatePassword(req.payload.userId, body.password);

  return { success: true };
}

@UseGuards(AuthGuard, BlockedGuard)
@Delete('/users/profile')
async deleteProfile(@Req() req, @Res({ passthrough: true }) res) {
  const { userId } = req.payload;
  await this.userService.deleteProfile(userId);
  removeAuthCookie(res);
  res.status(200);
  return { success: true, signOut: true };
}

@UseGuards(AuthGuard, BlockedGuard)
@Patch('/users/profile')
async updateUser(
  @Req() req,
  @Res({ passthrough: true }) res,
  @Body() updateUserDto: UpdateUserDto,
) {
  await this.userService.updateUser(req.payload.userId, updateUserDto);
  removeAuthCookie(res);
  res.status(200);
  return { success: true, signOut: true };
}

@UseGuards(AuthGuard, BlockedGuard)
@Get('/users/profile')
async getProfile(@Req() req) {
  const user = await this.userService.findUserByUsername(
    req.payload.username,
  );
  if (!user) throw new NotFoundException('No such user with this username!');
  delete user['verified'];
  delete user['blocked'];
  return { success: true, user };
}

@UseGuards(AuthGuard, BlockedGuard)
@Get('/users/:username')
async getUserByUsername(@Req() req, @Param('username') username) {
  const user = await this.userService.findUserByUsername(username);
  if (!user) throw new NotFoundException('No such user with this username!');
  if (req.isAdmin) {
    return { success: true, user };
  }
  if (!req.payload.verified && req.payload.userId !== user.userId)
    throw new ForbiddenException('You must be veiried');
  await this.userService.checkUserAvailability(
    req.payload.userId,
    user.userId,
  );
  delete user['verified'];
  delete user['blocked'];
  return { success: true, user };
}

@UseGuards(AuthGuard, AdminGuard, BlockedGuard)
@Get('/admin/allusers')
async getAllUsers(
  @Req() req,
  @Query('search') searchString: string,
  @Query('page') page: string,
  @Query('verified') verified?: string,
```

```
  @Query('blocked') blocked?: string,
  @Query('hidden') hidden?: string,
  @Query('orderBy') orderBy?: string,
) {
  const filters = {
    verified: parseBool(verified),
    blocked: parseBool(blocked),
    hidden: parseBool(hidden),
  };
  const { users, count } = await this.userService.getAllUsers(
    req.payload.userId,
    filters || {},
    searchString,
    orderBy,
    +page || 1,
  );

  return { success: true, users, count };
}

@UseGuards(AuthGuard, AdminGuard, BlockedGuard)
@Patch('/admin/block/:userId')
async blockUser(
  @Res({ passthrough: true }) res,
  @Body() body,
  @Param('userId') userId: number,
) {
  try {
    const { blockMessage } = body;
    await this.userService.blockUser(+userId, blockMessage);
    return { success: true };
  } catch (err) {
    res.status(400);
    return { success: false, message: 'Something went wrong, try later' };
  }
}

@UseGuards(AuthGuard, AdminGuard, BlockedGuard)
@Patch('/admin/unblock/:userId')
async unblockUser(
  @Res({ passthrough: true }) res,
  @Param('userId') userId: number,
) {
  try {
    await this.userService.unblockUser(+userId);
    return { success: true };
  } catch (err) {
    res.status(400);
    console.log(err);
    return { success: false, message: 'Something went wrong, try later' };
  }
}

@UseGuards(AuthGuard, AdminGuard, BlockedGuard)
@Patch('/admin/verify/:userId')
async verifyUser(
  @Res({ passthrough: true }) res,
  @Param('userId') userId: number,
) {
  try {
    await this.userService.verifyUser(+userId);
    return { success: true };
  } catch (err) {
    res.status(400);
    return { success: false, message: 'Something went wrong, try later' };
  }
}

@UseGuards(AuthGuard, VerifiedGuard, BlockedGuard)
@Get('/users')
async getUsers(
  @Req() req,
  @Query('page') page: string,
  @Query('sex') sex: string,
  @Query('orderBy') orderBy?: string,
  @Query('search') searchString?: string,
  @Query('countries') countryList?: string | undefined,
  @Query('languages') languageList?: string | undefined,
  @Query('hobbies') hobbyList?: string | undefined,
```

```
  @Query('minAge') minAge?: string,
  @Query('maxAge') maxAge?: string,
) {
  const filters = {
    searchString,
    countryList: parseArray(countryList),
    languageList: parseArray(languageList),
    hobbyList: parseArray(hobbyList),
    sexId: sex,
    minAge: +minAge,
    maxAge: +maxAge,
  };
  const { users, count } = await this.userService.getUsers(
    req.payload.userId,
    filters || {},
    orderBy,
    +page || 1,
  );

  return { success: true, users, count };
}

@UseGuards(AuthGuard, VerifiedGuard, BlockedGuard)
@Get('/users/blacklist')
async getUsersFromBlackList(@Req() req) {
  const users = await this.userService.getUsersFromBlackList(
    req.payload.userId,
  );

  return { success: true, users };
}

@UseGuards(AuthGuard, VerifiedGuard, BlockedGuard)
@Post('/users/blacklist')
async addUserToBlackList(@Req() req, @Body() body: AddToBlacklistDto) {
  if (body.blockedUserId == req.payload.userId)
    throw new BadRequestException('You can`t add yourself to blacklist');
  await this.userService.blacklistUser(
    body.blockedUserId,
    req.payload.userId,
  );

  return { success: true };
}

@UseGuards(AuthGuard, VerifiedGuard, BlockedGuard)
@Delete('/users/blacklist/:blUserId')
async removeUserFromBlackList(@Req() req, @Param('blUserId') blUserId) {
  await this.userService.removeUserFromBlackList(
    +blUserId,
    req.payload.userId,
  );

  return { success: true };
}

@UseGuards(AuthGuard, BlockedGuard)
@Patch('users/profile/photo')
async updateProfilePhoto(@Req() req) {
  const file = await req.file();
  if (!file) throw new BadRequestException('No file provided!');
  const profilePhotoURL = await this.userService.createUserProfilePhoto(
    req.payload.userId,
    file,
  );
  return {
    success: true,
    profilePhotoURL,
  };
}

@UseGuards(AuthGuard, BlockedGuard)
@Delete('users/profile/photo')
async deleteProfilePhoto(@Req() req) {
  await this.userService.deleteProfilePhoto(req.payload.userId);
  return {
    success: true,
  };
}
```

}

Файл chat.gateway.ts

```
import {
  SubscribeMessage,
  WebSocketGateway,
  WebSocketServer,
} from '@nestjs/websockets';
import { TokenService } from 'src/services/token.service';
import { cookieParse } from 'src/utils/cookieParser';
import { Server, WebSocket } from 'ws';
import { WebsocketUtils } from 'src/utils/websocket.util';

@WebSocketGateway({
  cors: {
    origin: '*',
  },
})
export class ChatGateway {
  @WebSocketServer() server: Server;

  callsMap = new Map();
  constructor(
    private websocketUtils: WebsocketUtils,
    private tokenService: TokenService,
  ) {}

  async validateToken(token: string) {
    try {
      return await this.tokenService.validateAccessToken(token);
    } catch (e) {
      return false;
    }
  }

  async validateClient(client, cookies) {
    if (!cookies?.accessToken) {
      client.close(1008, 'Unauthorized');
      return false;
    }
    const payload = await this.validateToken(cookies.accessToken);
    if (!payload || !payload.verified || payload.blocked) {
      client.close(1008, 'Forbidden');
      return false;
    }
    this.websocketUtils.removeUserFromServerByUserId(
      this.server,
      payload.userId,
    );
    client['user'] = payload;
    return true;
  }

  async handleConnection(client: WebSocket, req) {
    const cookieHeader = req.headers.cookie;
    const cookies = cookieParse(cookieHeader);
    const result = await this.validateClient(client, cookies);
    if (!result) return;

    const chatsUserIds = await this.websocketUtils.getChatsUsersIds(
      client['user'].userId,
    );
    this.websocketUtils.sendResponseToUsers(
      this.server,
      client,
      chatsUserIds,
      'userWentOnline',
      { userId: client['user'].userId },
    );
    await this.websocketUtils.getUnreadMessagesCountOfUser(client);
  }

  async handleDisconnect(client: WebSocket) {
    try {
      if (client['user']) {
        const chatsUserIds = await this.websocketUtils.getChatsUsersIds(
          client['user'].userId,
```

```
      );
      this.websocketUtils.sendResponseToUsers(
        this.server,
        client,
        chatsUserIds,
        'userWentOffline',
        { userId: client['user'].userId },
      );
      await this.websocketUtils.endAllCalls(
        this.server,
        client,
        this.callsMap,
      );
      return;
    }
  } catch (err) {
    console.log(err);
  }
}

@SubscribeMessage('removeChatForUser')
async handleBlackListUser(client: WebSocket, payload) {
  if (payload.userId)
    this.websocketUtils.sendResponseToUser(
      this.server,
      client,
      payload.userId,
      'removeChat',
      { userId: client['user'].userId },
    );
}

@SubscribeMessage('userCreatedChat')
async handleUserCreatedChat(client: WebSocket, payload) {
  if (payload.chat) {
    const secondUserId = this.websocketUtils.getSecondUserIdOfChat(
      client['user'].userId,
      payload.chat,
    );
    this.websocketUtils.sendResponseToUser(
      this.server,
      client,
      secondUserId,
      'newChat',
      { chat: payload.chat },
    );
  }
}

@SubscribeMessage('getOnlineUsers')
async handleGetOnlineUsers(client: WebSocket) {
  const chatsUserIds = await this.websocketUtils.getChatsUsersIds(
    client['user'].userId,
  );
  const onlineUserIds = await this.websocketUtils.getOnlineUsersFromList(
    this.server,
    chatsUserIds,
  );
  const data = { onlineUsers: onlineUserIds };
  this.websocketUtils.sendResponseToClient(client, 'onlineUsers', data);
}

@SubscribeMessage('readMessagesInChat')
async handleReadMessagesInChat(client: WebSocket, payload) {
  try {
    const { chatId, secondUserId } =
      await this.websocketUtils.readMessagesInChat(
        client['user'].userId,
        payload,
      );

    const data = {
      chatId,
      receiverId: client['user'].userId,
    };
    this.websocketUtils.sendResponseToClient(client, 'messagesRead', data);
    this.websocketUtils.sendResponseToUser(
      this.server,
      client,
```

```
      secondUserId,
      'messagesRead',
      data,
    );
  } catch (err) {
    if (err.message == "Can't find chat!")
      return this.websocketUtils.sendResponseToClient(
        client,
        'noChatWithSuchReceiver',
        {
          chatId: payload?.chatId,
        },
      );
    this.websocketUtils.sendError(client, err);
  }
}

@SubscribeMessage('getMessages')
async handleGetMessages(client: WebSocket, payload) {
  const messages = await this.websocketUtils.handleGetMessages(
    client['user'].userId,
    payload,
  );
  const data = { messages };
  this.websocketUtils.sendResponseToClient(
    client,
    payload.first ? 'firstMessages' : 'moreMessages',
    data,
  );
}

@SubscribeMessage('sendMessage')
async handleSendMessage(client: WebSocket, payload) {
  try {
    const newMessage = await this.websocketUtils.handleCreateMessage(
      client['user'].userId,
      payload,
    );
    if (newMessage) {
      const data = { newMessage };
      this.websocketUtils.sendResponseToUser(
        this.server,
        client,
        payload.receiverId,
        'newMessage',
        data,
      );
      this.websocketUtils.sendResponseToClient(
        client,
        'messageCreated',
        data,
      );
    } else {
      this.websocketUtils.sendResponseToClient(client, 'alertMessage', {
        message: 'Failed to send new message!',
      });
    }
  } catch (err) {
    if (err.message == "Can't find chat!")
      return this.websocketUtils.sendResponseToClient(
        client,
        'noChatWithSuchReceiver',
        {
          receiverId: payload.receiverId,
        },
      );
    this.websocketUtils.sendError(client, err);
  }
}

@SubscribeMessage('editMessage')
async handleEditMessage(client: WebSocket, payload) {
  try {
    const updatedMessage = await this.websocketUtils.handleEditMessage(
      client['user'].userId,
      payload,
    );
    if (updatedMessage) {
      const data = { updatedMessage };
```

```
      this.websocketUtils.sendResponseToUser(
        this.server,
        client,
        updatedMessage.receiverId,
        'messageEdited',
        data,
      );
      this.websocketUtils.sendResponseToClient(client, 'messageEdited', data);
    } else {
      this.websocketUtils.sendResponseToClient(client, 'alertMessage', {
        message: 'Failed to edit message!',
      });
    }
  } catch (err) {
    if (err.message == "Can't find chat!")
      return this.websocketUtils.sendResponseToClient(
        client,
        'noChatWithSuchReceiver',
        {
          receiverId: payload.receiverId,
        },
      );
    if (err.message == 'No message found!')
      return this.websocketUtils.sendResponseToClient(
        client,
        'alertMessage',
        {
          message: err.message,
        },
      );
    if (err.message == 'Only sender can change the message!')
      return this.websocketUtils.sendResponseToClient(
        client,
        'alertMessage',
        {
          message: err.message,
        },
      );
    this.websocketUtils.sendError(client, err);
  }
}

@SubscribeMessage('deleteMessage')
async handleDeleteMessage(client: WebSocket, payload) {
  try {
    const deletedMessage = await this.websocketUtils.handleDeleteMessage(
      client['user'].userId,
      payload,
    );
    if (deletedMessage) {
      const data = { deletedMessage };
      this.websocketUtils.sendResponseToUser(
        this.server,
        client,
        deletedMessage.receiverId,
        'messageDeleted',
        data,
      );
      this.websocketUtils.sendResponseToClient(
        client,
        'messageDeleted',
        data,
      );
    } else {
      this.websocketUtils.sendResponseToClient(client, 'alertMessage', {
        message: 'Failed to edit message!',
      });
    }
  } catch (err) {
    if (err.message == "Can't find chat!")
      return this.websocketUtils.sendResponseToClient(
        client,
        'noChatWithSuchReceiver',
        {
          receiverId: payload.receiverId,
        },
      );
    if (err.message == 'No message found!')
      return this.websocketUtils.sendResponseToClient(
```

```
        client,
        'alertMessage',
        {
          message: err.message,
        },
      );
    if (err.message == 'You can not delete not your own messages!')
      return this.websocketUtils.sendResponseToClient(
        client,
        'alertMessage',
        {
          message: err.message,
        },
      );
    this.websocketUtils.sendError(client, err);
  }
}

// Calling logic
@SubscribeMessage('callEnter')
async handleCallEnter(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat)
    return this.websocketUtils.sendResponseToClient(
      client,
      'noChatFound',
      {},
    );
  const call = this.callsMap.get(chat.chatId);

  if (call && client['user'].userId == call.initiator.userId) {
    this.callsMap.delete(chat.chatId);
  }
  if (call) {
    return this.websocketUtils.callExistHandle(this.server, client, call);
  } else {
    return this.websocketUtils.callNotExistHandle(
      this.server,
      client,
      this.callsMap,
      chat,
    );
  }
}

@SubscribeMessage('answerCall')
async handleAnswerCall(client: WebSocket, payload) {
  const chatIds = await this.websocketUtils.getUserChatIds(
    +client['user'].userId,
  );
  if (chatIds.length == 0 || !payload?.chatId) return;
  this.websocketUtils.callAnswerHandle(
    this.server,
    client,
    this.callsMap,
    payload.chatId,
    chatIds,
  );
}

@SubscribeMessage('rejectCall')
async handleRejectCall(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat) return;
  const call = this.callsMap.get(chat.chatId);
  if (!call) return;
  this.websocketUtils.callRejectHandle(
    this.server,
    this.callsMap,
    call,
    chat.chatId,
  );
}
```

```
@SubscribeMessage('endCall')
async handleEndCall(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});
  const call = this.callsMap.get(chat.chatId);
  if (!call)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});
  this.websocketUtils.callEndHandle(this.server, client, this.callsMap, chat);
}

@SubscribeMessage('toggleVideo')
async handleToggleVideo(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat) return;
  const call = this.callsMap.get(chat.chatId);
  if (!call) return;
  this.websocketUtils.toggleVideoHandle(this.server, client, payload, call);
}

@SubscribeMessage('callOffer')
async handleCallOffer(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});
  const call = this.callsMap.get(chat.chatId);
  if (!call)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});

  this.websocketUtils.callRTCOfferHandle(
    this.server,
    client,
    this.callsMap,
    chat,
    payload.offer,
  );
}

@SubscribeMessage('callAnswer')
async handleAnswerOffer(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});
  const call = this.callsMap.get(chat.chatId);
  if (!call)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});

  this.websocketUtils.callRTCAnswerHandle(
    this.server,
    client,
    this.callsMap,
    chat,
    payload.answer,
  );
}

@SubscribeMessage('iceCandidate')
async handleIceCandidate(client: WebSocket, payload) {
  const chat = await this.websocketUtils.getChatByChatId(
    +client['user'].userId,
    payload,
  );
  if (!chat)
    return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});
  const call = this.callsMap.get(chat.chatId);
  if (!call)
```

```
      return this.websocketUtils.sendResponseToClient(client, 'forbidden', {});
    this.websocketUtils.iceCandidateHandle(
      this.server,
      client,
      this.callsMap,
      chat,
      payload.candidate,
    );
  }
}
```

Файл schema.prisma

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema


generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}


enum Role {
  USER
  ADMIN
}

enum Sex {
  NK
  M
  F
  NA
}

model RefreshToken {
  userId       Int      @unique
  refreshToken String   @unique
  dateExpired  DateTime @default(dbgenerated("NOW() + interval '30 day'"))
  user         User     @relation(fields: [userId], references: [userId], onDelete: Cascade)
}

model BlockedUserMessage {
  userId Int @unique
  blockMessage String?
  user User @relation(fields: [userId], references: [userId], onDelete: Cascade)
}

model User {
  userId         Int          @id @default(autoincrement())
  firstName      String
  secondName     String
  username       String       @unique
  email          String       @unique
  password       String
  profilePhotoURL String?
  sexId          Sex          @default(NK)
  birthdayDate   DateTime
  countryId      Int
  createdAt      DateTime     @default(now())
  role           Role         @default(USER)
  blocked        Boolean      @default(false)
  verified       Boolean      @default(false)
  hidden         Boolean      @default(false)
  userDetails    UserDetails?
  refreshToken   RefreshToken?
  blockedUserMessage BlockedUserMessage?
  country        Country      @relation(fields: [countryId], references: [countryId])
  hobbies        Hobby[]
  languages      Language[]
  blockedByUsers  BlackList[]      @relation("blockedByUser")
  blocksUsers    BlackList[]      @relation("blocksUser")
  chatsAsUser1   Chat[]          @relation("User1")
  chatsAsUser2   Chat[]          @relation("User2")
```

```
  sentMessages    Message[]       @relation("Sender")
  receivedMessages Message[]      @relation("Receiver")
}

model Chat {
  chatId        Int      @id @default(autoincrement())
  user1      User     @relation("User1", fields: [user1Id], references: [userId])
  user1Id    Int
  user2      User     @relation("User2", fields: [user2Id], references: [userId])
  user2Id    Int
  createdAt  DateTime @default(now())
  messages   Message[] @relation("chatMessages")

  @@unique([user1Id, user2Id])
}

model Message {
  messageId String @id @default(uuid())
  chatId Int
  senderId Int
  receiverId Int
  content String
  timeCreated DateTime @default(now())
  edited Boolean @default(false)
  timeEdited DateTime?
  read Boolean @default(false)
  chat       Chat      @relation("chatMessages", fields: [chatId], references: [chatId], onDelete: Cascade)
  sender      User     @relation("Sender", fields: [senderId], references: [userId])
  receiver    User     @relation("Receiver", fields: [receiverId], references: [userId])
  @@index([senderId])
  @@index([receiverId])
  @@index([timeCreated])
}

model BlackList {
  blockedUserId Int
  blockedByUserId Int
  createdAt DateTime @default(now())
  blockedUser    User    @relation("blocksUser", fields: [blockedUserId], references: [userId], onDelete: Cascade)
  blockedByUser  User    @relation("blockedByUser", fields: [blockedByUserId], references: [userId], onDelete: Cascade)
  @@id([blockedUserId, blockedByUserId])
}

model UserDetails {
  userId        Int     @unique
  bio           String
  lookingForText String?
  user          User    @relation(fields: [userId], references: [userId], onDelete: Cascade)
}

model Country {
  countryId   Int    @id @default(autoincrement())
  countryName String @unique
  countryCode String @unique
  user        User[]
}

model Hobby {
  hobbyId Int    @id @default(autoincrement())
  hobby   String @unique
  users   User[]
}

model Language {
  languageId Int    @id @default(autoincrement())
  language   String @unique
  users      User[]
}
```

Файл main.js

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import store from "./store";

createApp(App).use(store).use(router).mount("#app");
```

Файл getAge.js

```
export const getAge = (dateString) => {
  let today = new Date();
  let birthDate = new Date(dateString);
  let age = today.getFullYear() - birthDate.getFullYear();
  let monthDiff = today.getMonth() - birthDate.getMonth();
  if (
    monthDiff < 0 ||
    (monthDiff === 0 && today.getDate() < birthDate.getDate())
  ) {
    age--;
  }
  return age;
};
```

Файл isNumber.js

```
export const isNumberWithoutDecimal = (value) => {
  if (typeof value === "number") {
    value = value.toString();
  }
  // eslint-disable-next-line prettier/prettier
  return (/^\d+$/).test(value);
};
```

Файл auth.module.js

```
import statusHandler from "../../handlers/status.handler";
import { postRequest } from "../../handlers/request.handler";
import notAuthorized from "../../handlers/notAuth.handler";
import router from "@/router";

export default {
  state: { isAuthenticated: false, payload: {}, isAdmin: false },
  getters: {
    isAuthenticated: (state) => state.isAuthenticated,
    getPayload: (state) => state.payload,
    isAdmin: (state) => state.isAdmin,
  },
  mutations: {
    setIsAuthenticated: (state, value) => (state.isAuthenticated = value),
    setPayload: (state, value) => (state.payload = value),
    setHidden: (state, value) => (state.payload.hidden = value),
    setProfilePhoto: (state, value) => (state.payload.profilePhotoURL = value),
    setIsAdmin: (state, value) => (state.isAdmin = value),
  },
  actions: {
    async signUp(context, data) {
      const response = await postRequest("/signup", data);
      const result = await statusHandler(response);
      return result;
    },
    async signIn({ commit, dispatch }, data) {
      const response = await postRequest("/signin", data);
      const result = await statusHandler(response);
      if (result.success) {
        commit("setIsAuthenticated", true);
        commit("setPayload", result.user);
        commit("setIsAdmin", result.user.role == "ADMIN" ? true : false);
        dispatch("connectToWS");
      } else {
        commit("setIsAuthenticated", false);
        commit("setPayload", {});
        commit("setIsAdmin", false);
```

```
      dispatch("disconnectWs");
    }
    return result;
  },
  async refreshTokens({ commit, dispatch }) {
    await notAuthorized(
      () => {
        commit("setIsAuthenticated", false);
        commit("setPayload", {});
        commit("setIsAdmin", false);
        dispatch("disconnectWs");
      },
      (response) => {
        commit("setIsAuthenticated", true);
        commit("setPayload", response.user);
        commit("setIsAdmin", response.user.role == "ADMIN" ? true : false);
        dispatch("getWebSocketInctance");
      }
    );
  },
  async signOut({ commit, dispatch }) {
    const response = await postRequest("/signout", {});
    const result = await statusHandler(response);
    commit("setIsAuthenticated", false);
    commit("setPayload", {});
    commit("setIsAdmin", false);
    dispatch("disconnectWs");
    router.push("/signin");
    return result;
  },
 },
};
```

Файл chat.module.js

```
import statusHandler from "../../handlers/status.handler";
import {
 deleteRequest,
 getRequest,
 postRequest,
} from "../../handlers/request.handler";

export default {
 state: {},
 actions: {
  async getChats() {
    const response = await getRequest(`/chats`);
    const result = await statusHandler(response, async () => {
      const func = await getRequest(`/chats`);
      return func;
    });
    if (result?.success) {
      return result.chats;
    }
    return [];
  },
  async createChat(data, userData) {
    const response = await postRequest(`/chats`, userData);
    const result = await statusHandler(response, async () => {
      const func = await postRequest(`/chats`, userData);
      return func;
    });
    if (result?.success) {
      return result.chat;
    }
  },

  async deleteChat(data, chatId) {
    if (!chatId) return;
    const response = await deleteRequest(`/chats/${chatId}`);
    const result = await statusHandler(response, async () => {
      const func = await deleteRequest(`/chats/${chatId}`);
      return func;
    });
    if (result?.success) {
      return result.success;
    }
    return;
```

```
  },
 },
};
```

Файл error.module.js

```
export default {
 state: { errorMessage: "" },
 getters: { getErrorMessage: (state) => state.errorMessage },
 mutations: {
  setErrorMessage: (state, errorMessage) =>
   (state.errorMessage = errorMessage),
 },
 actions: {
  async setErrorMessage({ commit }, errorMessage) {
   commit("setErrorMessage", errorMessage);
  },
  async resetErrorMessage({ commit }) {
   commit("setErrorMessage", "");
  },
 },
};
```

Файл lists.module.js

```
import statusHandler from "../../handlers/status.handler";
import { getRequest } from "../../handlers/request.handler";

export default {
 state: {
  sexIds: [
   { id: "NK", title: "Not known" },
   { id: "M", title: "Male" },
   { id: "F", title: "Female" },
   { id: "NA", title: "Not applicable" },
  ],
  filterSexList: [
   { title: "Any" },
   { title: "Male", value: "M" },
   { title: "Female", value: "F" },
  ],
  orderByList: [
   { title: "Account created ↓", value: "desc" },
   { title: "Account created ↑", value: "asc" },
  ],
  countries: [],
  hobbies: [],
  languages: [],
 },
 getters: {
  getSexList: (state) => state.sexIds,
  getFilterSexList: (state) => state.filterSexList,
  getOrderByList: (state) => state.orderByList,
  getCountriesList: (state) => state.countries,
  getHobbiesList: (state) => state.hobbies,
  getLanguagesList: (state) => state.languages,
 },
 mutations: {
  setCountries: (state, value) => (state.countries = value),
  setHobbies: (state, value) => (state.hobbies = value),
  setLanguagesList: (state, value) => (state.languages = value),
 },
 actions: {
  async getCHLLists({ state, commit }) {
   if (
    state.countries.length != 0 &&
    state.hobbies.length != 0 &&
    state.languages.length != 0
   )
    return;
   const response = await getRequest("/lists/chl");
   const result = await statusHandler(response);
   if (result?.success) {
    commit("setCountries", result.countries);
    commit("setHobbies", result.hobbies);
    commit("setLanguagesList", result.languages);
   }
   return result;
```

```
    },
  },
};
```

Файл loading.module.js

```
export default {
  state: { loadingStatus: false },
  getters: { getLoadingStatus: (state) => state.loadingStatus },
  mutations: {
    setLoadingStatus: (state, loadingStatus) =>
      (state.loadingStatus = loadingStatus),
  },
};
```

Файл modals.module.js

```
export default {
  state: {
    isModalBlockUserOpened: false,
    isModalCallingUserOpened: false,
    modalUserId: null,
    callInfo: null,
  },
  getters: {
    isModalBlockUserOpened: (state) => state.isModalBlockUserOpened,
    getUserIdModalData: (state) => state.modalUserId,
    isModalCallingUserOpened: (state) => state.isModalCallingUserOpened,
    getCallInfo: (state) => state.callInfo,
  },
  mutations: {
    setModalUserIdData: (state, modalUserId) =>
      (state.modalUserId = modalUserId),
    setBlockUserModalOpeningStatus: (state, isModalBlockUserOpened) =>
      (state.isModalBlockUserOpened = isModalBlockUserOpened),
    setCallInfo: (state, callInfo) => (state.callInfo = callInfo),
    setModalCallingUserStatus: (state, isModalCallingUserOpened) =>
      (state.isModalCallingUserOpened = isModalCallingUserOpened),
  },
  actions: {
    openBlockUserModal({ commit }, userId) {
      commit(`setBlockUserModalOpeningStatus`, true);
      commit("setModalUserIdData", userId);
    },
    closeBlockUserModal({ commit }) {
      commit(`setBlockUserModalOpeningStatus`, false);
      commit("setModalUserIdData", null);
    },
    openCallingUserModal({ commit }, callInfo) {
      commit(`setModalCallingUserStatus`, true);
      commit("setCallInfo", callInfo);
    },
    closeCallingUserModal({ commit }) {
      commit(`setModalCallingUserStatus`, false);
      commit("setCallInfo", null);
    },
  },
};
```

Файл user.module.js

```
import statusHandler from "../../handlers/status.handler";
import {
  deleteRequest,
  getRequest,
  patchRequest,
  postRequest,
  patchRequestForm,
} from "../../handlers/request.handler";
import router from "@/router";

export default {
```

```
state: { unreadMessagesCount: 0 },
getters: { getUnreadMessagesCount: (state) => state.unreadMessagesCount },
mutations: {
  setUnreadMessagesCount: (state, value) =>
    (state.unreadMessagesCount = value),
},
actions: {
  async updateUser({ commit }, userData) {
    const response = await patchRequest(`/users/profile`, userData);
    const result = await statusHandler(response, async () => {
      const func = await patchRequest(`/users/profile/`, userData);
      return func;
    });
    if (result?.success && result.signOut) {
      commit("setIsAuthenticated", false);
      commit("setPayload", {});
      commit("setIsAdmin", false);
      router.push("/signin");
    }
    return result;
  },

  async deleteUser({ commit }) {
    const response = await deleteRequest(`/users/profile`);
    const result = await statusHandler(response, async () => {
      const func = await deleteRequest(`/users/profile/`);
      return func;
    });
    if (result?.success && result.signOut) {
      commit("setIsAuthenticated", false);
      commit("setPayload", {});
      commit("setIsAdmin", false);
      return router.push("/signin");
    }
    return result;
  },

  async addToBlacklist(data, userId) {
    const response = await postRequest(`/users/blacklist`, {
      blockedUserId: userId,
    });
    const result = await statusHandler(response, async () => {
      const func = await postRequest(`/users/blacklist`, {
        blockedUserId: userId,
      });
      return func;
    });
    return result;
  },

  async removeUserFromBlacklist(data, userId) {
    const response = await deleteRequest(`/users/blacklist/${userId}`);
    const result = await statusHandler(response, async () => {
```

```
    const func = await deleteRequest(`/users/blacklist/${userId}`);
    return func;
  });

  return result;
},

async updateProfilePhoto({ commit }, form) {
  const response = await patchRequestForm(`/users/profile/photo`, form);
  const result = await statusHandler(response, async () => {
    const func = await patchRequestForm(`/users/profile/photo`, form);
    return func;
  });
  if (result?.success) commit("setProfilePhoto", result.profilePhotoURL);
  return result;
},

async deleteProfilePhoto({ commit }) {
  const response = await deleteRequest(`/users/profile/photo`);
  const result = await statusHandler(response, async () => {
    const func = await deleteRequest(`/users/profile/photo`);
    return func;
  });
  if (result?.success) commit("setProfilePhoto", null);
  return result;
},

async updateUserHidden({ commit }, userData) {
  const response = await patchRequest(`/users/profile/hidden`, userData);
  const result = await statusHandler(response, async () => {
    const func = await patchRequest(`/users/profile/hidden`, userData);
    return func;
  });
  if (result?.success) commit("setHidden", userData.hidden);
  return result;
},

async updateUserPassword(data, userData) {
  const response = await patchRequest(`/users/profile/password`, userData);
  const result = await statusHandler(response, async () => {
    const func = await patchRequest(`/users/profile/password`, userData);
    return func;
  });
  return result;
},

async getProfile() {
  const response = await getRequest(`/users/profile`);
  const result = await statusHandler(response, async () => {
    const func = await getRequest(`/users/profile`);
    return func;
  });
  if (result?.success) {
```

```
    return result;
  }
  return { ...result, user: {} };
},

async getUserProfile(data, username) {
  const response = await getRequest(`/users/${username}`);
  const result = await statusHandler(response, async () => {
    const func = await getRequest(`/users/${username}`);
    return func;
  });
  if (result?.success) {
    return result;
  }
  return { ...result, user: {} };
},

async verifyUser(data, userId) {
  const response = await patchRequest(`/admin/verify/${userId}`);
  const result = await statusHandler(response, async () => {
    const func = await patchRequest(`/admin/verify/${userId}`);
    return func;
  });
  return result;
},

async unblockUser(data, userId) {
  const response = await patchRequest(`/admin/unblock/${userId}`);
  const result = await statusHandler(response, async () => {
    const func = await patchRequest(`/admin/unblock/${userId}`);
    return func;
  });
  return result;
},

async blockUser(data, { userId, blockMessage }) {
  const response = await patchRequest(`/admin/block/${userId}`, {
    blockMessage,
  });
  const result = await statusHandler(response, async () => {
    const func = await patchRequest(`/admin/block/${userId}`, {
      blockMessage,
    });
    return func;
  });
  return result;
},

async getAdminUsers(data, filters = {}) {
  const query = Object.entries(filters)
    .map(([key, value]) => {
      if (value !== undefined) return `${key}=${value}`;
    })
```

```
      .join("&");
    const response = await getRequest(`/admin/allusers?${query}`);
    const result = await statusHandler(response, async () => {
      const func = await getRequest(`/admin/allusers?${query}`);
      return func;
    });
    if (result?.success) {
      return result;
    }
    return [];
  },

  async getUsers(data, filters = {}) {
    const query = Object.entries(filters)
      .map(([key, value]) => {
        if (value !== undefined && value !== "") {
          if (Array.isArray(value)) {
            if (value.length > 0) return `${key}=[${value}]`;
          } else return `${key}=${value}`;
        }
      })
      .join("&");
    const response = await getRequest(`/users?${query}`);
    const result = await statusHandler(response, async () => {
      const func = await getRequest(`/users?${query}`);
      return func;
    });
    if (result?.success) {
      return result;
    }
    return [];
  },

  async getBlacklist() {
    const response = await getRequest(`/users/blacklist`);
    const result = await statusHandler(response, async () => {
      const func = await getRequest(`/users/blacklist`);
      return func;
    });
    if (result?.success) {
      return result.users;
    }
    return [];
  },
 },
};
```

Файл ws.module.js

```
import router from "@/router";

export default {
```

```
state: { WebSocket: null, message: "" },
getters: {
  getWS: (state) => state.WebSocket,
  getWsMessage: (state) => state.message,
},
mutations: {
  setWS: (state, value) => (state.WebSocket = value),
  setWsMessage: (state, value) => (state.message = value),
},
actions: {
  async getWebSocketInctance({ state, dispatch }) {
    if (!state.WebSocket) await dispatch("connectToWS");
    return state.WebSocket;
  },
  connectToWS({ commit, dispatch }) {
    const webSocket = new WebSocket(process.env.VUE_APP_WS_URL);
    webSocket.addEventListener("close", async (event) => {
      if (event.reason == "NewSignIn") {
        await dispatch("refreshTokens");
        return router.push("/signin");
      }
      if (event.reason == "Unauthorized") {
        await dispatch("refreshTokens");
        return;
      }
      if (event.reason == "Forbidden") {
        return commit("setWS", null);
      }
    });
    webSocket.addEventListener("message", (event) => {
      const data = JSON.parse(event.data);
      commit("setWsMessage", data);
      dispatch("changeUnreadMessagesCount", { event: data.event });
      dispatch("callInitiated", { event: data.event, data: data.data });
      dispatch("unreadMessagesCount", { event: data.event, data: data.data });
    });
    commit("setWS", webSocket);
  },
  callInitiated({ dispatch }, { event, data }) {
    if (event == "callInitiated") {
      dispatch("openCallingUserModal", data);
    }
  },
  unreadMessagesCount({ commit }, { event, data }) {
    if (event == "unreadMessagesCount") {
      commit("setUnreadMessagesCount", data.count);
    }
  },
  changeUnreadMessagesCount({ commit, getters }, { event }) {
    if (event == "newMessage") {
      commit("setUnreadMessagesCount", getters.getUnreadMessagesCount + 1);
    }
    if (event == "messageDeleted") {
```

```
        commit("setUnreadMessagesCount", getters.getUnreadMessagesCount - 1);
      }
    },
    disconnectWs({ state, commit }) {
      if (state.WebSocket) state.WebSocket.close();
      commit("setWS", null);
    },
    async sendWsMessage({ dispatch }, payload) {
      const ws = await dispatch("getWebSocketInctance");
      if (!ws) {
        return;
      }
      if (ws.readyState != WebSocket.OPEN) setTimeout(() => {}, 100);
      if (ws.readyState == WebSocket.OPEN) ws.send(JSON.stringify(payload));
    },
  },
};
```

Файл index.js

```
import { createRouter, createWebHistory } from "vue-router";
import store from "../store";
import HomeView from "../views/HomeView.vue";

const routes = [
  {
    path: "/",
    name: "home",
    title: "FriendlyGlobe",
    component: HomeView,
  },
  {
    path: "/admin",
    name: "AdminPanelRoute",
    meta: {
      title: "Admin panel",
      requiresAuth: true,
      onlyAdmin: true,
    },
    component: () => import("../views/Admin/AdminPanelView.vue"),
  },
  {
    path: "/search",
    name: "UsersSearch",
    meta: {
      title: "Find friends",
      requiresAuth: true,
      onlyVerified: true,
    },
    component: () => import("../views/UsersSearch.vue"),
  },
  {
    path: "/users/:username",
```

```
  name: "UserProfile",
  meta: {
   title: "User",
   requiresAuth: true,
  },
  component: () => import("../views/Profile/ProfileView.vue"),
 },
 {
  path: "/profile/update",
  name: "UpdateProfile",
  meta: {
   title: "Update profile",
   requiresAuth: true,
  },
  component: () => import("../views/Profile/UpdateProfile.vue"),
 },
 {
  path: "/chats",
  name: "ChatsPage",
  meta: {
   title: "Chats",
   requiresAuth: true,
   onlyVerified: true,
  },
  component: () => import("../views/ChatsView.vue"),
 },
 {
  path: "/calls",
  name: "CallsPage",
  meta: {
   title: "Call",
   requiresAuth: true,
   onlyVerified: true,
  },
  component: () => import("../views/CallView.vue"),
 },
 {
  path: "/profile/blacklist",
  name: "ProfileBlacklist",
  meta: {
   title: "Your blacklist",
   requiresAuth: true,
  },
  component: () => import("../views/Profile/BlacklistView.vue"),
 },
 {
  path: "/signin",
  name: "SignInRoute",
  meta: { title: "Sign In", onlyUnAuth: true },
  component: () => import("../views/Auth/SignInView.vue"),
 },

 {
```

```
    path: "/signup",
    name: "SignUpRoute",
    meta: { title: "Sign Up", onlyUnAuth: true },
    component: () => import("../views/Auth/SignUpView.vue"),
  },
  {
    path: "/404",
    name: "404",
    meta: {
      title: "Not Found",
    },
    component: function () {
      return import("../views/NotFound.vue");
    },
  },
  {
    path: "/500",
    name: "500",
    meta: {
      title: "Server Internal Error",
    },
    component: function () {
      return import("../views/ServerError.vue");
    },
  },
  {
    path: "/:catchAll(.*)",
    redirect: "/",
  },
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
  scrollBehavior() {
    return { top: 0 };
  },
});

const getCurrUser = async () => {
  if (store.getters.isAuthenticated) return store.getters.getPayload;
  await store.dispatch("refreshTokens");
  if (store.getters.isAuthenticated) return store.getters.getPayload;
  return null;
};

const requiresLoading = (toRoute) => {
  if (toRoute.matched.some((record) => record.meta.requiresLoading)) {
    store.commit("setLoadingStatus", true);
  } else {
    store.commit("setLoadingStatus", false);
  }
};
```

```
router.beforeEach(async (toRoute, fromRoute, next) => {
  if (store.getters.getErrorMessage !== "") store.dispatch("resetErrorMessage");
  requiresLoading(toRoute);
  window.document.title =
    toRoute.meta && toRoute.meta.title ? toRoute.meta.title : "FriendlyGlobe";
  if (toRoute.matched.some((record) => record.meta.requiresAuth)) {
    const user = await getCurrUser();
    if (user) {
      if (toRoute.meta.onlyAdmin && !store.getters.isAdmin)
        return next(`/users/${user.username}`);
      if (toRoute.meta.onlyVerified && !store.getters.getPayload.verified)
        return next(`/users/${user.username}`);
      return next();
    }
    return next("/signin");
  } else if (toRoute.matched.some((record) => record.meta.onlyUnAuth)) {
    if (await getCurrUser()) {
      return next("/");
    }
  }
  next();
});

export default router;
```

Файл notAuth.handler.js

```
export default async function notAuthorized(onError, onSuccess) {
  const response = await fetch(
    `${process.env.VUE_APP_SERVER_URL}/auth/refresh`,
    {
      method: "POST",
      credentials: "include",
    }
  );
  const { status } = response;
  const JSONResponse = await response.json();
  if (status >= 400) {
    return onError(JSONResponse);
  } else if (status === 200) return onSuccess(JSONResponse);
}
```

Файл request.handler.js

```
import store from "../store";

const preRequestSend = (loading) => {
  if (loading) store.commit("setLoadingStatus", true);
  store.dispatch("resetErrorMessage");
};

const afterRequestSend = () => {
  store.commit("setLoadingStatus", false);
```

```
};

export async function postRequest(
  address = "",
  data = "",
  loading = true,
  credentials = true,
  url = ""
) {
  preRequestSend(loading);
  const response = await fetch(
    `${url || process.env.VUE_APP_SERVER_URL}${address}`,
    {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      credentials: !credentials ? "same-origin" : "include",
      body: JSON.stringify(data),
    }
  );
  afterRequestSend();
  return response;
}

export async function patchRequest(
  address = "",
  data = "",
  loading = true,
  credentials = true,
  url = ""
) {
  preRequestSend(loading);
  const response = await fetch(
    `${url || process.env.VUE_APP_SERVER_URL}${address}`,
    {
      method: "PATCH",
      headers: {
        "Content-Type": "application/json",
      },
      credentials: !credentials ? "same-origin" : "include",
      body: JSON.stringify(data),
    }
  );
  afterRequestSend();
  return response;
}

export async function patchRequestForm(
  address = "",
  data = "",
  loading = true,
  credentials = true,
```

```
  url = ""
) {
  preRequestSend(loading);
  const response = await fetch(
    `${url || process.env.VUE_APP_SERVER_URL}${address}`,
    {
      method: "PATCH",
      credentials: !credentials ? "same-origin" : "include",
      body: data,
    }
  );
  afterRequestSend();
  return response;
}

export async function updateRequest(
  address = "",
  data = "",
  loading = true,
  credentials = true,
  url = ""
) {
  preRequestSend(loading);
  const response = await fetch(
    `${url || process.env.VUE_APP_SERVER_URL}${address}`,
    {
      method: "UPDATE",
      headers: {
        "Content-Type": "application/json",
      },
      credentials: !credentials ? "same-origin" : "include",
      body: JSON.stringify(data),
    }
  );
  afterRequestSend();
  return response;
}

export async function getRequest(
  address = "",
  loading = true,
  credentials = true,
  url = ""
) {
  preRequestSend(loading);
  const response = await fetch(
    `${url || process.env.VUE_APP_SERVER_URL}${address}`,
    {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
      credentials: !credentials ? "same-origin" : "include",
```

```
    }
  );
  afterRequestSend();
  return response;
}

export async function deleteRequest(
  address = "",
  loading = true,
  credentials = true,
  url = ""
) {
  preRequestSend(loading);
  const response = await fetch(
    `${url || process.env.VUE_APP_SERVER_URL}${address}`,
    {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
      },
      credentials: !credentials ? "same-origin" : "include",
    }
  );
  afterRequestSend();
  return response;
}
```

Файл status.handler.js

```
import router from "@/router";
import notAuthorized from "./notAuth.handler";
import store from "../store/index";

const onError = ({ message }) => {
  router.push("/signin");
  store.dispatch("setErrorMessage", message);
  return;
};

export default async function statusHandler(
  response,
  onUnAuthError = () => {}
) {
  const { status } = response;
  if (status == 200 || status == 201) {
    const json = await response.json();
    return json;
  } else {
    if (status === 401) {
      return notAuthorized(onError, async () => {
        const res = await onUnAuthError();
        if (res.status !== 401) return statusHandler(res);
        store.dispatch("setIsAuthenticated", false);
        const json = await res.json();
        return json;
      });
    } else if (status === 400 || status === 403) {
      const JSONResponse = await response.json();
      store.dispatch("setErrorMessage", JSONResponse.message);
      return JSONResponse;
    } else if (status === 404) return router.push("/404");
    else if (status >= 500) return router.push("/500");
  }
}
```

Файл App.vue

```
<template>
 <div
  class="wrapper"
  :class="{
   'message-block-shown': showMessageBlock,
  }"
 >
  <NavBar />
  <div class="messages-block" v-if="showMessageBlock">
   <div class="sticky-message" v-if="getPayload.hidden">
    You are in hidden mode
   </div>
   <div class="sticky-message alert" v-if="!getPayload.verified">
    Your account is not verified! Some functions is not availiable!
   </div>
   <div class="sticky-message" v-if="!getPayload.profilePhotoURL">
    <router-link to="/profile/update">Upload photo</router-link>
   </div>
  </div>
  <div class="background-fixed"></div>
  <router-view
   class="main"
   :class="{
    'margin-top': !showMessageBlock,
   }"
  />
  <CallingUserModal />
 </div>
</template>

<script>
import { mapActions, mapGetters } from "vuex";
import CallingUserModal from "./components/Modals/CallingUserModal.vue";
import NavBar from "./components/NavBar.vue";
export default {
 components: { CallingUserModal, NavBar },
 computed: {
  ...mapGetters(["isAuthenticated", "isAdmin", "getPayload", "getWS"]),
  showMessageBlock() {
   return (
    (this.getPayload.hidden ||
     !this.getPayload.verified ||
     !this.getPayload.profilePhotoURL) &&
    this.isAuthenticated
   );
  },
 },
 methods: { ...mapActions(["signOut"]) },
 async created() {
  await this.$store.dispatch("refreshTokens");
```

```
  },
};
</script>
```

Файл UsersSearch.vue

```
<template>
  <div class="users-search">
    <h2>Find your friend</h2>
    <div class="filters-container">
      <h3>Filters:</h3>
      <form @submit.prevent="onSubmit">
        <div class="filter-list">
          <FormInput
            title="Search"
            placeholder="Find user by username"
            v-model.trim="filters.search"
          />
          <FormInput
            title="Min age"
            placeholder="Min age"
            v-model.trim="filters.minAge"
          >
            <MinAgeError :v="v.filters.minAge" />
            <MinMaxValueError :v="v.filters.minAge" :min="12" :max="120" />
          </FormInput>
          <FormInput
            title="Max age"
            placeholder="Max age"
            v-model.trim="filters.maxAge"
          >
            <MaxAgeError :v="v.filters.maxAge" />
            <MinMaxValueError :v="v.filters.maxAge" :min="12" :max="120" />
          </FormInput>
          <FormInputSlot title="Languages">
            <template #input>
              <VueMultiselect
                v-model="languages"
                placeholder="Search languages"
                label="language"
                track-by="languageId"
                :options="getLanguagesList"
                :multiple="true"
              />
            </template>
          </FormInputSlot>
          <FormInputSlot title="Hobbies">
            <template #input>
              <VueMultiselect
                v-model="hobbies"
                placeholder="Search hobbies"
                label="hobby"
                track-by="hobbyId"
```

```
        :options="getHobbiesList"
        :multiple="true"
      />
    </template>
  </FormInputSlot>
  <FormInputSlot title="Countries">
    <template #input>
      <VueMultiselect
        v-model="countries"
        placeholder="Search countries"
        label="countryName"
        track-by="countryId"
        :options="getCountriesList"
        :multiple="true"
      />
    </template>
  </FormInputSlot>
  <FormInputSlot title="Sex:">
    <template #input>
      <VueMultiselect
        v-model="sex"
        :options="getFilterSexList"
        :close-on-select="true"
        :clear-on-select="false"
        placeholder="Select"
        :searchable="false"
        :allow-empty="false"
        label="title"
        track-by="title"
        selectLabel=" "
        deselectLabel=" "
      />
    </template>
  </FormInputSlot>
  <FormInputSlot title="Order by date:">
    <template #input>
      <VueMultiselect
        v-model="orderBy"
        :options="getOrderByList"
        :close-on-select="true"
        :clear-on-select="false"
        placeholder="Select"
        :searchable="false"
        :allow-empty="false"
        label="title"
        track-by="title"
        selectLabel=" "
        deselectLabel=" "
      />
    </template>
  </FormInputSlot>
</div>
<CustomButton btnType="submit">Filter</CustomButton>
```

```
    </form>
   </div>
   <div class="users-list users-grid" v-if="users.length > 0">
    <UserListItem v-for="user in users" :key="user.userId" :user="user" />
   </div>
   <div class="users-list users-grid" v-if="users.length == 0">
    <h3>No users found</h3>
   </div>
   <PaginationList
    v-if="totalPages > 0"
    :current-page="currentPage"
    :total-pages="totalPages"
    @page-changed="changePage"
   />
  </div>
</template>

<script>
import { mapActions, mapGetters } from "vuex";

import useVuelidate from "@vuelidate/core";
import { minValue, maxValue } from "@vuelidate/validators";

import FormInputSlot from "@/components/Inputs/FormInputSlot.vue";
import FormInput from "@/components/Inputs/FormInput.vue";
import VueMultiselect from "vue-multiselect";
import CustomButton from "@/components/Buttons/CustomButton.vue";
import PaginationList from "@/components/PaginationList.vue";
import MinMaxValueError from "@/components/InputErrors/MinMaxValueError.vue";
import MaxAgeError from "@/components/InputErrors/MaxAgeError.vue";
import MinAgeError from "@/components/InputErrors/MinAgeError.vue";
import UserListItem from "../components/ListItems/UserListItem.vue";

export default {
  name: "UsersSearch",
  components: {
    FormInputSlot,
    VueMultiselect,
    FormInput,
    CustomButton,
    PaginationList,
    MinMaxValueError,
    MaxAgeError,
    MinAgeError,
    UserListItem,
  },

  setup() {
    return { v: useVuelidate() };
  },
  data() {
    return {
      users: [],
```

```
    currentPage: 1,
    usersCount: 0,
    itemsPerPage: 10,
    orderBy: "",
    sex: "",
    countries: [],
    languages: [],
    hobbies: [],
    filters: {
      search: "",
      minAge: "",
      maxAge: "",
      countries: [],
      languages: [],
      hobbies: [],
      orderBy: "",
      sex: "",
    },
  };
},
validations() {
  return {
    filters: {
      minAge: {
        minValue: minValue(12),
        maxValue: maxValue(120),
        maxAge: function (value) {
          if (this.filters.maxAge == "" || value == "") {
            return true;
          }
          return value <= this.filters.maxAge;
        },
      },
      maxAge: {
        minValue: minValue(12),
        maxValue: maxValue(120),
        minAge: function (value) {
          if (this.filters.minAge == "" || value == "") {
            return true;
          }
          return value >= this.filters.minAge;
        },
      },
    },
  };
},
computed: {
  ...mapGetters([
    "getHobbiesList",
    "getLanguagesList",
    "getCountriesList",
    "getOrderByList",
    "getFilterSexList",
```

```
  ]),
  totalPages() {
    return Math.ceil(this.usersCount / this.itemsPerPage);
  },
 },
 methods: {
  ...mapActions(["getUsers", "getCHLLists"]),
  async changePage(pageNumber) {
    this.currentPage = pageNumber;
    const { users, count: usersCount } = await this.getUsers({
      ...this.filters,
      page: this.currentPage,
    });
    this.users = users;
    this.usersCount = usersCount;
  },
  async onSubmit() {
    this.v.$touch();
    if (this.v.$error) return;
    this.currentPage = 1;
    const { users, count: usersCount } = await this.getUsers({
      ...this.filters,
      page: this.currentPage,
    });
    this.users = users;
    this.usersCount = usersCount;
  },
 },
 watch: {
  languages() {
    this.filters.languages = this.languages.map((item) => item.languageId);
  },
  hobbies() {
    this.filters.hobbies = this.hobbies.map((item) => item.hobbyId);
  },
  countries() {
    this.filters.countries = this.countries.map((item) => item.countryId);
  },
  orderBy() {
    this.filters.orderBy = this.orderBy.value;
  },
  sex() {
    this.filters.sex = this.sex.value;
  },
 },
 async mounted() {
  this.orderBy = await this.getOrderByList[0];
  this.sex = await this.getFilterSexList[0];
  const { users, count: usersCount } = await this.getUsers({
    ...this.filters,
    page: this.currentPage,
  });
  this.users = users;
```

```
    this.usersCount = usersCount;
    await this.getCHLLists();
  },
};
</script>
```

Файл ChatsView.vue

```
<template>
  <div class="chats-view">
    <div class="side-bar-left" :class="{ hidden: sideBarHidden }">
      <h3>Chats</h3>
      <ChatItem
        v-for="chat in chats"
        :key="chat.chatId"
        :chat="chat"
        :active="currentChat?.chatId == chat.chatId"
        @click="chooseChat(chat)"
      />
    </div>
    <button class="toggle-chats" @click="toggleSideBar">
      <span class="line"></span>
      <span class="line"></span>
      <span class="line"></span>
    </button>
    <div class="chat-container">
      <div class="header">
        <div class="username">
          <router-link v-if="currentChat" :to="`users/${receiver.username}`">{{
            receiver.username
          }}</router-link>
          <span v-else>{{ receiver.username }}</span>
        </div>
        <div class="is-online">
          <span class="online-dot" v-if="currentChat?.online"></span>
        </div>
        <button
          class="call-user-btn"
          @click="callUser"
          v-if="currentChat"
          :disabled="!currentChat.online"
        >
          <img src="../assets/icons/phone-call.png" alt="phone-call" />
        </button>
        <DotsMenu
          :isCurrUser="isCurrUserNull"
          :user="receiver"
          blacklistLink="/profile/blacklist"
          @addToBlacklist="addToBlacklistEmit"
        >
          <li v-if="!isCurrUserNull">
            <button @click="deleteChatClick">Delete</button>
          </li>
```

```
      </DotsMenu>
    </div>
    <div class="body" @scroll="debouncedCheckScrollTop" ref="body">
      <div class="message-container" ref="messageContainer">
        <MessageItem
          v-for="(message, index) in messages"
          :key="index"
          :message="message"
          :currentUserId="getPayload.userId"
          @onEditClick="openEditModal"
          @onDeleteClick="onDeleteMessage"
        />
      </div>
    </div>

    <MessageInput
      :disable="!currentChat"
      :chatId="currentChat?.chatId"
      @onSend="sendNewMessage"
    />
  </div>
  <EditMessageModal
    @editMessage="onEditMessage"
    :message="messageToEdit"
    v-model="editMessageModalOpened"
  />
 </div>
</template>

<script>
import { mapActions, mapGetters } from "vuex";
import DotsMenu from "../components/DotsMenu.vue";
import MessageInput from "../components/Inputs/MessageInput.vue";
import ChatItem from "../components/ListItems/ChatItem.vue";
import MessageItem from "../components/ListItems/MessageItem.vue";
import EditMessageModal from "../components/Modals/EditMessageModal.vue";
export default {
 components: {
  MessageInput,
  MessageItem,
  ChatItem,
  DotsMenu,
  EditMessageModal,
 },
 name: "ChatsView",
 data() {
  return {
   currentChat: null,
   messageToEdit: null,
   editMessageModalOpened: false,
   chats: [],
   lastTopPosition: 0,
   sideBarHidden: true,
```

```
    };
  },
  computed: {
    ...mapGetters([
      "getWS",
      "getPayload",
      "getWsMessage",
      "getUnreadMessagesCount",
    ]),
    debouncedCheckScrollTop() {
      return this.debounce(this.checkScroll, 200);
    },
    receiver() {
      if (this.currentChat == null)
        return { userId: null, username: "Choose chat", online: false };
      return this.currentChat.user1.userId == this.getPayload.userId
        ? this.currentChat.user2
        : this.currentChat.user1;
    },
    messages() {
      if (!this.currentChat) return [];
      return this.currentChat.messages;
    },
    isCurrUserNull() {
      if (!this.currentChat) return true;
      return false;
    },
    lastUnreadMessageIndex() {
      let lastUnreadMessageIndex = -1;
      for (let i = 0; i < this.messages.length; i++) {
        if (
          !this.messages[i].read &&
          this.messages[i].receiverId == this.getPayload.userId
        ) {
          lastUnreadMessageIndex = i;
          break;
        }
      }
      return lastUnreadMessageIndex;
    },
    eventHandlers() {
      return {
        error: () => {
          this.$router.push("/505");
        },
        messageCreated: async () => {
          await this.appendNewMessage(this.getWsMessage.data.newMessage);
        },
        newMessage: async () => {
          await this.appendNewMessage(this.getWsMessage.data.newMessage);
        },
        newChat: () => {
          this.changeUnreadMessagesCount("+");
```

```
        this.appendNewChat(this.getWsMessage.data.chat);
      },
      moreMessages: () => {
        this.appendNewMessages(this.getWsMessage.data.messages);
      },
      firstMessages: async () => {
        await this.setFirstMessages(this.getWsMessage.data.messages);
      },
      messagesRead: () => {
        this.readMessagesInChatByUserId(
          this.getWsMessage.data.chatId,
          this.getWsMessage.data.receiverId
        );
      },
      onlineUsers: () => {
        this.makeChatsOnline(this.getWsMessage.data.onlineUsers);
      },
      userWentOnline: () => {
        this.makeChatOnlineOrOffline(this.getWsMessage.data.userId, true);
      },
      userWentOffline: () => {
        this.makeChatOnlineOrOffline(this.getWsMessage.data.userId, false);
      },
      alertMessage: () => {
        alert(this.getWsMessage.data.message);
      },
      messageEdited: () => {
        this.editLocalMessage(this.getWsMessage.data.updatedMessage);
      },
      messageDeleted: () => {
        this.removeLocalMessage(this.getWsMessage.data.deletedMessage);
      },

      noChatWithSuchReceiver: () => {
        this.currentChat = null;
        this.removeChat(
          this.getWsMessage.data.receiverId,
          this.getWsMessage.data.chatId
        );
        alert("No chat found with such user!");
      },
      removeChat: () => {
        this.currentChat = null;
        this.removeChat(this.getWsMessage.data.userId);
      },
    };
  },
},
methods: {
  ...mapActions([
    "getChats",
    "createChat",
    "deleteChat",
```

```
  "addToBlacklist",
  "sendWsMessage",
 ]),
 toggleSideBar() {
  this.sideBarHidden = !this.sideBarHidden;
 },
 debounce(func, wait) {
  let timeoutId;
  return function debounced(...args) {
   clearTimeout(timeoutId);
   timeoutId = setTimeout(() => {
    func.apply(this, args);
   }, wait);
  };
 },
 checkScroll() {
  const messageContainer = this.$refs.messageContainer;
  const body = this.$refs.body;
  this.lastTopPosition = messageContainer.offsetTop + 1 - body.offsetTop;
  if (body.scrollTop <= messageContainer.offsetTop + 1 - body.offsetTop) {
   this.getMoreMessagesOfCurrChat();
  }
 },
 scrollToLastUnreadMessage() {
  this.$nextTick(() => {
   if (this.lastUnreadMessageIndex != -1) {
    const container = this.$refs.messageContainer;
    const lastUnreadMessage =
     container.children[this.lastUnreadMessageIndex];

    if (lastUnreadMessage) {
     const scrollOffset =
      lastUnreadMessage.offsetTop - lastUnreadMessage.offsetHeight;
     this.$refs.body.scrollTop = scrollOffset;
    }
   }
  });
 },
 chooseChat(chat) {
  this.currentChat = chat;
  this.sideBarHidden = true;
 },
 async sendNewMessage(message) {
  if (this.currentChat) {
   const payload = {
    event: "sendMessage",
    data: {
     receiverId: this.receiver.userId,
     content: message,
    },
   };
   await this.sendWsMessage(payload);
  }
```

```
},
makeChatsOnline(userIds) {
 this.chats.forEach((chat) => {
  if (
   userIds?.includes(chat.user1.userId) ||
   userIds?.includes(chat.user2.userId)
  ) {
   chat.online = true;
  }
 });
},
makeChatOnlineOrOffline(userId, onlineStatus) {
 const chatIndex = this.chats.findIndex(
  (chat) => chat?.user1.userId == userId || chat?.user2.userId == userId
 );
 if (chatIndex != -1) this.chats[chatIndex].online = onlineStatus;
},
async appendNewMessage(message) {
 const chatIndex = this.chats.findIndex(
  (chat) => chat.chatId == message.chatId
 );
 if (chatIndex != -1) {
  this.chats[chatIndex].messages.push(message);
  if (message.receiverId == this.getPayload.userId) {
   this.chats[chatIndex].unreadCount =
    this.chats[chatIndex].unreadCount + 1;
   if (this.currentChat?.chatId == this.chats[chatIndex]?.chatId) {
    await this.readAllUserUnreadMessagesInChat();
   }
  }
  let removedElement = this.chats.splice(chatIndex, 1)[0];
  this.chats.unshift(removedElement);
 }
},
appendNewMessages(messages) {
 if (this.currentChat) {
  this.currentChat.messages = [
   ...messages.reverse(),
   ...this.currentChat.messages,
  ];
  this.$nextTick(() => {
   this.$refs.body.scrollTop = this.lastTopPosition;
  });
 }
},
async setFirstMessages(messages) {
 if (this.currentChat) {
  this.currentChat.messages = messages.reverse();
  this.scrollToLastUnreadMessage();
  await this.readAllUserUnreadMessagesInChat();
 }
},
appendNewChat(chat) {
```

```
  if (chat) {
    const findChat = this.chats.find((item) => item.chatId == chat.chatId);
    if (!findChat) {
      chat["online"] = true;
      chat["unreadCount"] = 1;
      this.chats.unshift(chat);
    }
  }
},
removeChat(userId, chatId = null) {
  if (!userId && !chatId) return;
  if (userId) {
    const foundChat = this.chats.find(
      (chat) => chat.user1.userId == userId || chat.user2.userId == userId
    );
    this.changeUnreadMessagesCount("-", foundChat?.unreadCount);
    this.chats = this.chats.filter(
      (chat) => chat.user1.userId !== userId && chat.user2.userId !== userId
    );
  } else if (chatId) {
    this.chats = this.chats.filter((chat) => chat.chatId != chatId);
  }
},
async getMoreMessagesOfCurrChat() {
  if (this.currentChat) {
    await this.sendWsMessage({
      event: "getMessages",
      data: {
        chatId: this.currentChat.chatId,
        lastMessageTimeCreated:
          this.currentChat?.messages?.at(0)?.timeCreated,
        first: false,
      },
    });
  }
},
async getFirstMessagesOfCurrChat() {
  if (this.currentChat)
    await this.sendWsMessage({
      event: "getMessages",
      data: {
        chatId: this.currentChat.chatId,
        first: true,
      },
    });
},
async addToBlacklistEmit(userId) {
  this.currentChat = null;
  this.removeChat(userId);
  await this.addToBlacklist(userId);
  await this.sendWsMessage({
    event: "removeChatForUser",
    data: { userId },
```

```
    });
  },
  changeUnreadMessagesCount(operator, number = 1) {
    if (operator == "+") {
      this.$store.commit(
        "setUnreadMessagesCount",
        this.getUnreadMessagesCount + number
      );
    } else {
      this.$store.commit(
        "setUnreadMessagesCount",
        this.getUnreadMessagesCount - number
      );
    }
  },
  readMessagesInChatByUserId(chatId, userId) {
    const foundChat = this.chats.find((chat) => chat.chatId == chatId);
    if (foundChat) {
      foundChat.messages.forEach((message) => {
        if (message.receiverId == userId && !message.read) {
          message.read = true;
          this.changeUnreadMessagesCount("-");
        }
      });
    }
  },
  async readAllUserUnreadMessagesInChat() {
    if (this.currentChat) {
      await this.sendWsMessage({
        event: "readMessagesInChat",
        data: { chatId: this.currentChat.chatId },
      });
      this.currentChat.unreadCount = 0;
    }
  },
  leaveLastMessageOnLeaveChat(chat) {
    if (chat) {
      chat.messages = chat.messages.slice(-1);
    }
  },
  async deleteChatClick() {
    if (this.currentChat) {
      await this.sendWsMessage({
        event: "removeChatForUser",
        data: { userId: this.receiver.userId },
      });
      const chatId = this.currentChat.chatId;
      this.removeChat(null, chatId);
      await this.deleteChat(chatId);
      this.currentChat = null;
    }
  },
  async onEditMessage(data) {
```

```
    if (this.messageToEdit == null) return;
    await this.sendWsMessage({ event: "editMessage", data });
    this.messageToEdit = null;
  },
  async onDeleteMessage(data) {
    if (!data) return;
    const result = confirm("Do you want to delete this message?");
    if (result) await this.sendWsMessage({ event: "deleteMessage", data });
  },
  openEditModal(message) {
    this.messageToEdit = message;
    this.editMessageModalOpened = true;
  },
  editLocalMessage(message) {
    if (!message) return;
    const foundChat = this.chats.find(
      (chat) => chat.chatId == message?.chatId
    );
    if (!foundChat) return;
    const messageIndex = foundChat.messages.findIndex(
      (item) => item.messageId == message.messageId
    );
    if (messageIndex != -1) foundChat.messages[messageIndex] = message;
  },
  removeLocalMessage(message) {
    if (!message) return;
    const foundChat = this.chats.find(
      (chat) => chat.chatId == message?.chatId
    );
    if (!foundChat) return;
    const messageIndex = foundChat.messages.findIndex(
      (item) => item.messageId == message.messageId
    );
    if (messageIndex != -1) foundChat.messages.splice(messageIndex, 1);
  },
  async handleWsEvent(event) {
    const eventName = event;
    if (Object.prototype.hasOwnProperty.call(this.eventHandlers, eventName)) {
      await this.eventHandlers[eventName]();
    }
  },
  callUser() {
    if (this.currentChat)
      this.$router.push(`/calls?chatId=${this.currentChat.chatId}`);
  },
},

watch: {
  async getWsMessage() {
    await this.handleWsEvent(this.getWsMessage.event);
  },
  async currentChat(newValue, oldValue) {
    this.leaveLastMessageOnLeaveChat(oldValue);
```

```
    this.lastTopPosition = 0;
    await this.getFirstMessagesOfCurrChat();
   },
  },

  async mounted() {
   let chat = null;
   if (this.$route.query.userId) {
    const queryUserId = this.$route.query.userId;
    this.$router.replace({ query: null });
    if (queryUserId != this.getPayload?.userId) {
     chat = await this.createChat({
      newFriendUserId: queryUserId,
     });
     await this.sendWsMessage({ event: "userCreatedChat", data: { chat } });
    }
   }
   this.chats = await this.getChats();
   if (chat) {
    this.currentChat = this.chats.find((item) => item.chatId == chat.chatId);
   } else {
    this.sideBarHidden = false;
   }
   await this.sendWsMessage({ event: "getOnlineUsers" });
  },
};
</script>
```

Файл CallView.vue

```
<template>
 <div class="call-view">
  <div class="view-block">
   <div class="call-message-container" v-if="message">
    <h3>{{ message }}</h3>
   </div>
   <div class="videos-container">
    <div
     class="video-block iniciator"
     :class="{ 'small-video': callInfo?.status == 'inProgress' }"
    >
     <div class="no-video" v-if="!localStream || !videoEnabled">
      <div class="profile-photo-block">
       <img
        v-if="getSelf?.profilePhotoURL"
        :src="getOwnProfilePhotoUrl"
        alt="Initiator photo"
       />
       <div v-else class="no-photo-circle">
        {{ ownUsernameFirstLetter }}
       </div>
      </div>
      <h3>{{ getSelf?.username }}</h3>
```

```
      </div>

      <video
        v-if="localStream"
        ref="localVideo"
        id="local"
        class="video-player"
        autoplay
        playsinline
      />
    </div>
    <div
      class="video-block recipient main-video"
      v-if="callInfo?.status == 'inProgress'"
    >
      <div class="no-video" v-if="!remoteVideoEnabled || !remoteStream">
        <div class="profile-photo-block">
          <img
            v-if="getSecondUser?.profilePhotoURL"
            :src="getsecondUserProfilePhotoUrl"
            alt="Recipient photo"
          />
          <div v-else class="no-photo-circle">
            {{ secondUserUsernameFirstLetter }}
          </div>
        </div>

        <h3>{{ getSecondUser?.username }}</h3>
      </div>

      <video
        v-if="remoteStream"
        ref="remoteVideo"
        id="remote"
        class="video-player"
        autoplay
        playsinline
      />
    </div>
  </div>
</div>
<div class="controls">
  <CallControllButton
    class="media-btn"
    @click="toggleAudioEnabled"
    :class="{ disabled: !audioEnabled }"
  >
    <img src="../assets/icons/microphone.png" alt="toggle microphone" />
  </CallControllButton>
  <CallControllButton
    class="media-btn"
    @click="toggleVideoEnabled"
    :class="{ disabled: !videoEnabled }"
```

```
      >
        <img src="../assets/icons/video-camera.png" alt="toggle video" />
      </CallControllButton>
      <CallControllButton class="call-end" @click="handleEndCallClick">
        <img src="../assets/icons/phone-call.png" alt="end call" />
      </CallControllButton>
    </div>
  </div>
</template>

<script>
import { mapActions, mapGetters } from "vuex";
import { isNumberWithoutDecimal } from "@/utils/isNumber";
import CallControllButton from "../components/Buttons/CallControllButton.vue";

export default {
  components: { CallControllButton },
  name: "CallView",
  data() {
    return {
      chatId: null,
      message: "",
      callInfo: null,
      callingTimeout: null,
      localStream: null,
      remoteStream: null,
      videoEnabled: true,
      audioEnabled: true,
      peerConnection: null,
      remoteVideoEnabled: false,
      stunServers: {
        iceServers: [
          {
            urls: [
              "stun:stun.l.google.com:19302",
              "stun:stun1.l.google.com:19302",
              "stun:stun2.l.google.com:19302",
              "stun:stun3.l.google.com:19302",
              "stun:stun4.l.google.com:19302",
            ],
          },
        ],
      },
      constraints: {
        video: {
          width: { min: 640, ideal: 1920, max: 1920 },
          height: {
            min: 480,
            ideal: 1080,
            max: 1080,
          },
        },
        audio: true,
```

```
    },
    iceCandidates: [],
  };
},
computed: {
  ...mapGetters(["getWS", "getPayload", "getWsMessage"]),
  getSelf() {
    return this.callInfo?.recipient.userId == this.getPayload.userId
      ? this.callInfo?.recipient
      : this.callInfo?.initiator;
  },
  getSecondUser() {
    return this.callInfo?.recipient.userId != this.getPayload.userId
      ? this.callInfo?.recipient
      : this.callInfo?.initiator;
  },
  ownUsernameFirstLetter() {
    return this.getSelf?.username[0]?.toUpperCase();
  },
  secondUserUsernameFirstLetter() {
    return this.getSecondUser?.username[0]?.toUpperCase();
  },
  getOwnProfilePhotoUrl() {
    return `${process.env.VUE_APP_SERVER_URL}/static/images/${this.getSelf?.profilePhotoURL}`;
  },
  getsecondUserProfilePhotoUrl() {
    return `${process.env.VUE_APP_SERVER_URL}/static/images/${this.getSecondUser?.profilePhotoURL}`;
  },
  eventHandlers() {
    return {
      error: () => {
        this.$router.push("/505");
      },
      remoteVideoToggled: () => {
        this.remoteVideoEnabled = this.getWsMessage.data.enabled;
      },
      noChatFound: () => {
        this.redirectToChats();
      },
      callEnded: () => {
        this.callEndedLogic();
      },
      callRejected: () => {
        this.callRejectedLogic();
      },
      recipientAnswered: async () => {
        await this.onRecipientAnswer();
      },
      callDetails: () => {
        this.setCallInfo();
      },
      dialing: async () => {
        await this.dialingLogic();
```

```
      },

      offerCreated: async () => {
        await this.onOfferCreated();
      },
      answerCreated: async () => {
        await this.addAnswer();
      },
      newIceCandidate: async () => {
        await this.addIceCandidate();
      },
      connectedToCall: async () => {
        await this.onCallConnect();
      },
    };
  },
},
methods: {
  ...mapActions(["sendWsMessage"]),
  isNumberWithoutDecimal,

  async init() {
    try {
      this.localStream = await navigator.mediaDevices.getUserMedia(
        this.constraints
      );
      this.$nextTick(() => {
        this.$refs.localVideo.srcObject = this.localStream;
      });
    } catch (err) {
      await this.endCall();
      alert("Can't get your camera! You will be redirected immediately!");
      this.redirectToChats();
    }
  },
  async createPeerConnection() {
    this.peerConnection = new RTCPeerConnection(this.stunServers);
    this.remoteStream = new MediaStream();
    this.$nextTick(() => {
      this.$refs.remoteVideo.srcObject = this.remoteStream;
    });

    if (!this.localStream) await this.init();

    this.localStream.getTracks().forEach((track) => {
      this.peerConnection.addTrack(track, this.localStream);
    });

    this.peerConnection.ontrack = (event) => {
      event.streams[0].getTracks().forEach((track) => {
        this.remoteStream.addTrack(track);
      });
    };
```

```
  this.peerConnection.onicecandidate = async (event) => {
    if (event.candidate) {
      await this.sendWsMessage({
        event: "iceCandidate",
        data: { candidate: event.candidate, chatId: this.chatId },
      });
    }
  };
},

async createOffer() {
  await this.createPeerConnection();

  let offer = await this.peerConnection.createOffer();
  await this.peerConnection.setLocalDescription(offer);
  return offer;
},

async createAnswer() {
  const offer = this.getWsMessage.data.offer;
  await this.createPeerConnection();
  if (this.localStream) {
    this.toggleVideoEnabled();
    this.toggleAudioEnabled();
    this.remoteVideoEnabled = true;
  }
  this.peerConnection.setRemoteDescription(offer);
  let answer = await this.peerConnection.createAnswer();
  this.peerConnection.setLocalDescription(answer);
  await this.sendWsMessage({
    event: "callAnswer",
    data: { answer, chatId: this.chatId },
  });
},

async onOfferCreated() {
  await this.createAnswer();
},

async addAnswer() {
  const answer = this.getWsMessage.data.answer;
  if (!this.peerConnection.currentRemoteDescription) {
    this.peerConnection.setRemoteDescription(answer);
  }
},
async addIceCandidate() {
  const candidate = this.getWsMessage.data.candidate;
  this.iceCandidates.push(candidate);

  if (this.peerConnection.currentRemoteDescription) {
    // If the currentRemoteDescription is set, add the candidate immediately
    this.peerConnection.addIceCandidate(candidate);
```

```
  } else {
    // Process stored iceCandidates when currentRemoteDescription is set
    this.peerConnection.onaddstream = () => {
      // Iterate through the stored iceCandidates and add them to the peerConnection
      for (const storedCandidate of this.iceCandidates) {
        this.peerConnection.addIceCandidate(storedCandidate);
      }
      // Clear the iceCandidates array after processing
      this.iceCandidates = [];
    };
  }
},
stopStream(videoStream) {
  if (videoStream) {
    const tracks = videoStream.getTracks();
    tracks.forEach((track) => track.stop());
  }
},
async toggleVideoEnabled() {
  this.videoEnabled = !this.videoEnabled;
  if (this.localStream) {
    let videoTrack = this.localStream
      .getTracks()
      .find((track) => track.kind === "video");

    videoTrack.enabled = this.videoEnabled;
    await this.toggleLocalVideoEnabled();
  }
},
toggleAudioEnabled() {
  this.audioEnabled = !this.audioEnabled;
  if (this.localStream) {
    let audioTrack = this.localStream
      .getTracks()
      .find((track) => track.kind === "audio");

    if (audioTrack) audioTrack.enabled = this.audioEnabled;
  }
},
callTimeOuted() {
  this.message =
    "Calling time out! You will be redirected to chats page in 5!";
  const timeoutId = setTimeout(() => {
    this.$router.push("/chats");
  }, 5 * 1000);
  this.setCallingTimeout(timeoutId);
},
setCallingTimeout(newTimeout) {
  clearTimeout(this.callingTimeout);
  this.callingTimeout = newTimeout;
},
async onRecipientAnswer() {
  clearTimeout(this.callingTimeout);
```

```
    this.changeCallStatus();
    const offer = await this.createOffer();
    this.message = "";
    await this.sendWsMessage({
      event: "callOffer",
      data: { offer, chatId: this.chatId },
    });
  },
  async onCallConnect() {
    this.setCallInfo();
    await this.init();
  },
  async dialingLogic() {
    this.setCallInfo();
    this.message = `Calling ${this.getSecondUser?.username}`;
    const timeoutId = setTimeout(this.callTimeOuted, 60 * 1000);
    this.setCallingTimeout(timeoutId);
    await this.init();
  },
  setCallInfo() {
    this.callInfo = this.getWsMessage.data.call;
  },
  changeCallStatus() {
    if (this.callInfo) {
      this.callInfo.status = this.getWsMessage.data.status;
    }
  },
  callRejectedLogic() {
    this.changeCallStatus();
    this.stopStreams();
    this.message = "Call rejected! You will be redirected in 5 seconds!";
    const timeoutId = setTimeout(() => {
      this.redirectToChats();
    }, 5 * 1000);
    this.setCallingTimeout(timeoutId);
  },
  callEndedLogic() {
    if (this.chatId == this.getWsMessage.data.chatId) {
      this.changeCallStatus();
      this.message = "Call ended! You will be redirected in 5 seconds!";
      this.stopStreams();
      const timeoutId = setTimeout(() => {
        this.redirectToChats();
      }, 5 * 1000);
      this.setCallingTimeout(timeoutId);
    }
  },
  stopStreams() {
    this.stopStream(this.localStream);
    this.stopStream(this.remoteStream);
    this.localStream = null;
    this.remoteStream = null;
  },
```

```
async endCall() {
  if (this.callInfo && this.chatId) {
    this.stopStreams();
    await this.sendWsMessage({
      event: "endCall",
      data: { chatId: this.chatId },
    });
  }
},
redirectToChats() {
  this.stopStreams();
  this.$router.push("/chats");
},

async toggleLocalVideoEnabled() {
  if (this.remoteStream && this.callInfo && this.chatId) {
    await this.sendWsMessage({
      event: "toggleVideo",
      data: { enabled: this.videoEnabled, chatId: this.chatId },
    });
  }
},

async handleEndCallClick() {
  await this.endCall();
  clearTimeout(this.callingTimeout);
  this.redirectToChats();
},

async handleRouteEnter() {
  const chatId = this.$route.query.chatId;
  if (this.isNumberWithoutDecimal(chatId)) {
    await this.sendWsMessage({
      event: "callEnter",
      data: { chatId },
    });
    this.chatId = +chatId;
  } else this.redirectToChats();
},

async handleRouteUpdate() {
  clearTimeout(this.callingTimeout);
  await this.handleRouteEnter();
  this.peerConnection = null;
  this.audioEnabled = true;
  this.videoEnabled = true;
  this.stopStreams();
  this.message = null;
},

async handleWsEvent(event) {
  const eventName = event;
  if (Object.prototype.hasOwnProperty.call(this.eventHandlers, eventName)) {
```

```
      await this.eventHandlers[eventName]();
    }
   },
  },

  async beforeRouteUpdate(to) {
   if (this.chatId != to.query.chatId) {
    await this.endCall();
   }
  },

  async beforeRouteLeave() {
   await this.endCall();
  },

  watch: {
   async $route() {
    await this.handleRouteUpdate();
   },
   async getWsMessage() {
    await this.handleWsEvent(this.getWsMessage.event);
   },
  },

  async mounted() {
   await this.handleRouteEnter();
   if (this.chatId) window.addEventListener("beforeunload", this.endCall);
  },
  beforeUnmount() {
   window.removeEventListener("beforeunload", this.endCall);
   this.stopStream();
  },
};
</script>
```

Файл UpdateProfile.vue

```
<template>
 <div class="update-profile">
  <section>
   <h2>Options</h2>
   <div class="user-settings">
    <div class="mode">
     Hidden mode:
     <ToggleButton v-model="hidden" />
    </div>
    <div class="mode">
     <router-link to="/profile/blacklist">Blacklist</router-link>
    </div>
   </div>
  </section>
  <section>
   <div class="upload-photo">
```

```
    <UploadImage
      :imageUrl="profilePhotoURL"
      @imageUpdated="updateProfilePhoto"
    />
    <RemoveButton v-if="profilePhotoURL" @click="removeProfilePhoto" />
  </div>
</section>
<section>
  <div class="user-data">
    <div class="header">
      <h2>Update</h2>
      <p>
        If you update your account information, your account will be flagged
        for verification. During the verification process, your account may
        be limited in some way.
      </p>
      <span class="error-msg" v-if="getErrorMessage">
        {{ getErrorMessage }}
      </span>
    </div>
    <form @submit.prevent="updateUserSubmit" autocomplete="on">
      <section class="short-personal-info">
        <h3>Short personal info</h3>
        <div class="inputs-container">
          <FormInput title="First Name" v-model.trim="updateData.firstName">
            <RequiredError :v="v.updateData.firstName" />
            <MinMaxError :v="v.updateData.firstName" :min="2" :max="15" />
          </FormInput>
          <FormInput
            title="Second Name"
            v-model.trim="updateData.secondName"
          >
            <RequiredError :v="v.updateData.secondName" />
            <MinMaxError :v="v.updateData.secondName" :min="2" :max="15" />
          </FormInput>
          <FormInputSlot title="Birthday">
            <template #input>
              <input
                class="input"
                type="date"
                v-model="updateData.birthdayDate"
                :min="minDate"
                :max="maxDate"
              />
            </template>
            <template #error>
              <RequiredError :v="v.updateData.birthdayDate" />
            </template>
          </FormInputSlot>
          <FormInputSlot title="Sex">
            <template #input>
              <VueMultiselect
                v-model="sex"
```

```
            :options="getSexList"
            :close-on-select="true"
            :clear-on-select="false"
            placeholder="Select"
            label="title"
            track-by="id"
            :allow-empty="false"
            selectLabel=" "
            deselectLabel=" "
           />
          </template>
          <template #error>
           <RequiredError :v="v.sex" />
          </template>
         </FormInputSlot>
         <FormInputSlot title="Country">
          <template #input>
           <VueMultiselect
            v-model="updateData.country"
            :options="getCountriesList"
            :close-on-select="true"
            :clear-on-select="false"
            placeholder="Select your country"
            label="countryName"
            track-by="countryId"
            :allow-empty="false"
            selectLabel=" "
            deselectLabel=" "
           />
          </template>
          <template #error>
           <RequiredError :v="v.updateData.country" />
          </template>
         </FormInputSlot>
        </div>
       </section>
       <section class="user-description">
        <h3>About you</h3>
        <p class="section-description">
         In this section you should write information about you and what
         are you loking for on this website. (The second is optional)
        </p>
        <div class="inputs-container">
         <FormInputSlot title="Bio">
          <template #input>
           <textarea
            v-model="updateData.bio"
            :maxlength="1000"
            placeholder="Type something about you (at least 100 symbols)"
           />
          </template>
          <template #error>
           <RequiredError :v="v.updateData.bio" />
```

```
        <MinMaxError :v="v.updateData.bio" :min="100" :max="1000" />
      </template>
    </FormInputSlot>
    <FormInputSlot title="What are you looking for?">
      <template #input>
        <textarea
          v-model="updateData.lookingForText"
          :maxlength="1000"
          placeholder="Your expectations (at least 100 symbols)"
        />
      </template>
      <template #error>
        <MinMaxError
          :v="v.updateData.lookingForText"
          :min="100"
          :max="1000"
        />
      </template>
    </FormInputSlot>
  </div>
</section>
<section class="user-languages">
  <h3>Languages</h3>
  <p class="section-description">
    Choose languages you are interested in (at least 1)
  </p>
  <div class="inputs-container">
    <FormInputSlot title="Hobbies and interests">
      <template #input>
        <VueMultiselect
          v-model="updateData.languages"
          placeholder="Search language"
          label="language"
          track-by="languageId"
          :options="getLanguagesList"
          :multiple="true"
          :allow-empty="false"
        />
      </template>
      <template #error>
        <RequiredError :v="v.updateData.languages" />
      </template>
    </FormInputSlot>
  </div>
</section>
<section class="user-hobbies">
  <h3>Hobbies and interest</h3>
  <p class="section-description">
    Choose your favourite hobbies and interests (at least 1)
  </p>
  <div class="inputs-container">
    <FormInputSlot title="Hobbies and interests">
      <template #input>
```

```
            <VueMultiselect
              v-model="updateData.hobbies"
              placeholder="Search hobby/interest"
              label="hobby"
              track-by="hobbyId"
              :options="getHobbiesList"
              :multiple="true"
              :allow-empty="false"
            />
          </template>
          <template #error>
            <RequiredError :v="v.updateData.hobbies" />
          </template>
        </FormInputSlot>
      </div>
    </section>
    <section class="user-passwords"></section>
    <CustomButton btnType="submit">Update</CustomButton>
  </form>
  <form @submit.prevent="updateUserPasswordSubmit" autocomplete="on">
    <section class="user-passwords">
      <h2>Update password</h2>
      <div class="inputs-container">
        <FormInput
          title="Password"
          v-model.trim="updatePasswrod.password"
          :isPassword="!showPassword"
        >
          <RequiredError :v="v.updatePasswrod.password" />
          <MinMaxError
            :v="v.updatePasswrod.password"
            :min="6"
            :max="18"
          />
        </FormInput>
        <FormInput
          title="Password confirmation"
          v-model="passwordConfirm"
          :isPassword="!showPassword"
        >
          <SameAsError
            :v="v.passwordConfirm"
            text="Passwords have to be the same"
          />
        </FormInput>
      </div>
      <ShowPassword v-model="showPassword" />
    </section>
    <CustomButton btnType="submit">Change Password</CustomButton>
  </form>
 </div>
</section>
<section class="delete-user">
```

```
    <h2>Delete account</h2>
    <DeleteMaxButton @click="deleteProfile" />
   </section>
  </div>
</template>

<script>
import { mapActions, mapGetters } from "vuex";

import useVuelidate from "@vuelidate/core";
import { required, minLength, maxLength, sameAs } from "@vuelidate/validators";

import FormInput from "../../components/Inputs/FormInput.vue";
import FormInputSlot from "../../components/Inputs/FormInputSlot.vue";
import RequiredError from "../../components/InputErrors/RequiredError.vue";
import SameAsError from "../../components/InputErrors/SameAsError.vue";
import MinMaxError from "../../components/InputErrors/MinMaxError.vue";
import ShowPassword from "../../components/Inputs/ShowPassword.vue";
import CustomButton from "../../components/Buttons/CustomButton.vue";
import VueMultiselect from "vue-multiselect";
import ToggleButton from "../../components/Buttons/ToggleButton.vue";
import DeleteMaxButton from "../../components/Buttons/DeleteMaxButton.vue";
import UploadImage from "../../components/UploadImage.vue";
import RemoveButton from "../../components/Buttons/RemoveButton.vue";

export default {
 name: "UpdateProfile",
 components: {
   FormInput,
   FormInputSlot,
   RequiredError,
   SameAsError,
   MinMaxError,
   ShowPassword,
   CustomButton,
   VueMultiselect,
   ToggleButton,
   DeleteMaxButton,
   UploadImage,
   RemoveButton,
 },
 setup() {
   return { v: useVuelidate() };
 },
 data() {
   return {
     updateData: {
       firstName: "",
       secondName: "",
       sexId: null,
       birthdayDate: null,
       country: null,
       languages: [],
```

```
      hobbies: [],
      bio: "",
      lookingForText: "",
    },
    updatePasswrod: {
      password: "",
    },
    profilePhotoURL: null,
    sex: null,
    hidden: false,
    passwordConfirm: "",
    showPassword: false,
  };
},
validations() {
  return {
    updateData: {
      firstName: {
        required,
        maxLength: maxLength(15),
        minLength: minLength(2),
      },
      secondName: {
        required,
        maxLength: maxLength(15),
        minLength: minLength(2),
      },
      birthdayDate: { required },
      country: { required },
      bio: {
        required,
        maxLength: maxLength(1000),
        minLength: minLength(100),
      },
      lookingForText: {
        maxLength: maxLength(1000),
        minLength: minLength(100),
      },
      languages: { required },
      hobbies: { required },
    },
    updatePasswrod: {
      password: {
        required,
        maxLength: maxLength(18),
        minLength: minLength(6),
      },
    },
    sex: { required },
    passwordConfirm: { sameAs: sameAs(this.updatePasswrod.password) },
  };
},
computed: {
```

```
  ...mapGetters([
    "getErrorMessage",
    "getSexList",
    "getCountriesList",
    "getHobbiesList",
    "getLanguagesList",
  ]),
  maxDate() {
    const today = new Date();
    today.setFullYear(today.getFullYear() - 12);
    return today.toISOString().slice(0, 10);
  },
  minDate() {
    const today = new Date();
    today.setFullYear(today.getFullYear() - 120);
    return today.toISOString().slice(0, 10);
  },
 },
 methods: {
  ...mapActions([
    "deleteUser",
    "updateUser",
    "getProfile",
    "getCHLLists",
    "updateUserHidden",
    "updateUserPassword",
    "deleteProfilePhoto",
  ]),
  async updateUserPasswordSubmit() {
    this.v.updatePasswrod.$touch();
    this.v.passwordConfirm.$touch();
    if (this.v.updatePasswrod.$error && this.v.passwordConfirm.$error) {
      return 0;
    }
    const { success } = await this.updateUserPassword(this.updatePasswrod);

    if (this.getErrorMessage)
      return window.scroll({ top: 0, behavior: "smooth" });

    if (success) {
      alert("Password updated succesfully");
      this.v.updatePasswrod.$reset();
      this.v.passwordConfirm.$reset();
      this.updatePasswrod.password = "";
      this.passwordConfirm = "";
    }
  },
  async updateUserSubmit() {
    this.v.updateData.$touch();
    if (this.v.updateData.$error) {
      return 0;
    }
    const { success } = await this.updateUser(this.updateData);
```

```
    if (this.getErrorMessage)
      return window.scroll({ top: 0, behavior: "smooth" });

    if (success) alert("Information updated succesfully");
   },
   async removeProfilePhoto() {
    if (this.profilePhotoURL) {
     await this.deleteProfilePhoto();
     this.profilePhotoURL = null;
    }
   },
   async deleteProfile() {
    const confirmationResult = confirm(
     "Do you want to delete your account? It will remove all data!"
    );

    if (confirmationResult) await this.deleteUser();
   },
   updateProfilePhoto(imageUrl) {
    this.profilePhotoURL = imageUrl;
   },
   fillUpdateData(user) {
    this.updateData = {
     firstName: user.firstName,
     secondName: user.secondName,
     sexId: user.sexId,
     birthdayDate: new Date(user.birthdayDate).toISOString().split("T")[0],
     country: user.country,
     languages: user.languages,
     hobbies: user.hobbies,
     bio: user.userDetails.bio,
     lookingForText: user.userDetails.lookingForText,
    };
    this.profilePhotoURL = user.profilePhotoURL;
    this.sex = this.getSexList.find((sex) => sex.id == user.sexId);
    this.hidden = user.hidden;
   },
  },
  watch: {
   sex() {
    this.updateData.sexId = this.sex.id;
   },
   async hidden() {
    await this.updateUserHidden({ hidden: this.hidden });
   },
  },
  async mounted() {
   if (
    this.getCountriesList.length == 0 ||
    this.getHobbiesList.length == 0 ||
    this.getLanguagesList.length == 0
   )
```

```
    await this.getCHLLists();
    this.$data.updateData.sex = this.getSexList[0];
    const { user, success } = await this.getProfile();
    if (!success) this.$router.push("/");
    this.fillUpdateData(user);
  },
};
</script>
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

СОЦІАЛЬНА МЕРЕЖА ДЛЯ ПОШУКУ ДРУЗІВ ТА СПІЛКУВАННЯ

Програма та методика тестування

КПІ.ІП-9226.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_Ірина ЗЕНІВ___

Нормоконтроль:
Катерина ЛІЩУК_

Виконавець:
Олексій ХАРЕНКО_

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є веб-застосунок соціальної мережі для пошуку друзів та спілкування між ними у режимі реального часу. Програмне забезпечення було розроблено для використання на мобільних пристроях та комп’ютерах. Chrome та Edge – браузери, під які було розроблене веб-застосунок.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Edge);

– перевірка роботи веб-застосунку на мобільних пристроях;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– тестування на мобільних пристроях з різною роздільною здатністю екрану;

– тестування на мобільних пристроях з різною версією операційної системи;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування зміни орієнтації екрану;

– тестування інтерфейсу користувача;

– тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

СОЦІАЛЬНА МЕРЕЖА ДЛЯ ПОШУКУ ДРУЗІВ ТА СПІЛКУВАННЯ

Керівництво користувача

КПІ.ІП-9226.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_Ірина ЗЕНІВ___

Нормоконтроль:
Катерина ЛІЩУК

Виконавець:
Олексій ХАРЕНКО

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Соціальна мережа FriendlyGlobe надає можливість знайти собі нових друзів, незважаючи на кордони. Можна подивитись інформацію про користувача, чим він цікавиться, які мови знає або вивчає. А дізнатись ще більше про людину ви зможете розпочавши спілкування в вбудованому в веб-застосунок чаті, який працює в реальному часі. Ще краще пізнати один одного вам дозволить функціонал відео та аудіо дзвінка.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 5 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 30 мегабіт.

2.2 Завантаження застосунку

Для отримання доступу до веб-застосунку, необхідно мати активне підключення до мережі Інтернет і перейти за посиланням https://friendlyglobe.onrender.com/ .

2.3 Перевірка коректної роботи

Після переходу за посиланням відкриється головна сторінка соціальної мережі FriendlyGlobe, після чого користувач зможе розпочати роботу.

3 ВИКОНАННЯ ПРОГРАМИ

3.1 Керівництво користувача

При запуску веб-застосунку, користувач побачить перед собою головну сторінку, на якій буде можливість перейти на екран реєстрації чи входу (рисунок 3.1).

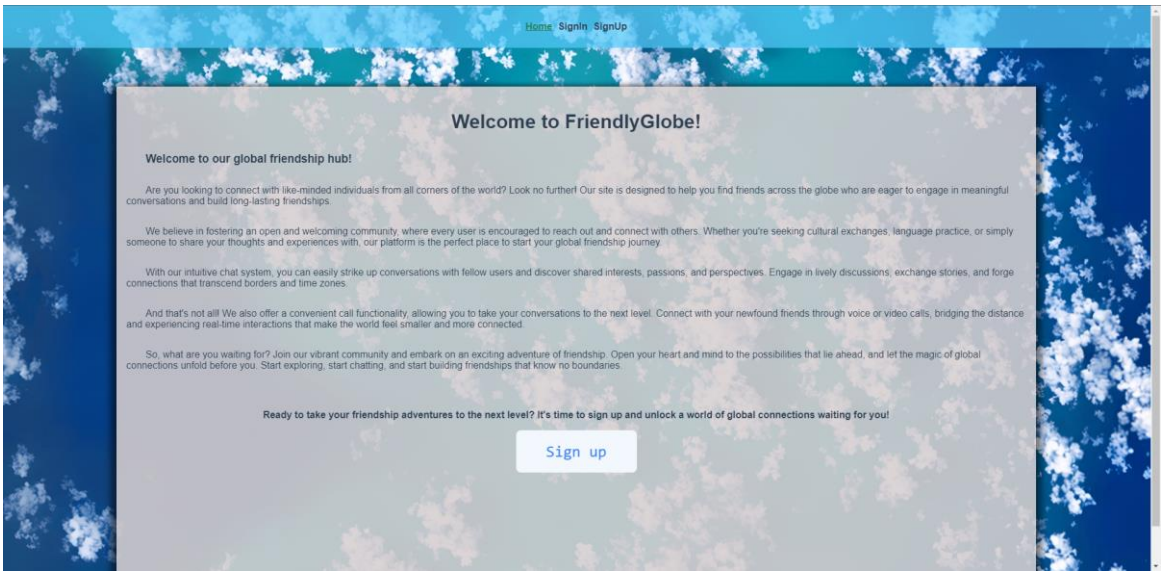


Рисунок 3.1 – Початкова сторінка веб-застосунку

Якщо користувач не має облікового запису, то він переходить на сторінку реєстрації та заповнює дані (рисунок 3.2).

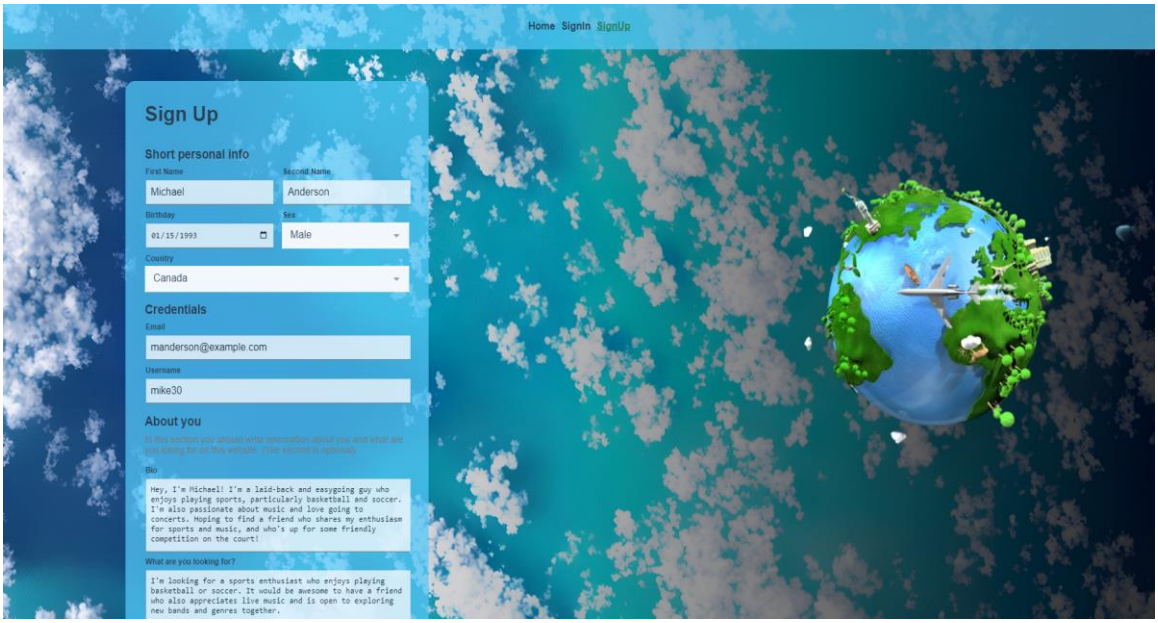


Рисунок 3.2 – Сторінка реєстрації

Після успішної реєстрації користувача буде переправлено на сторінку входу (рисунок 3.3).

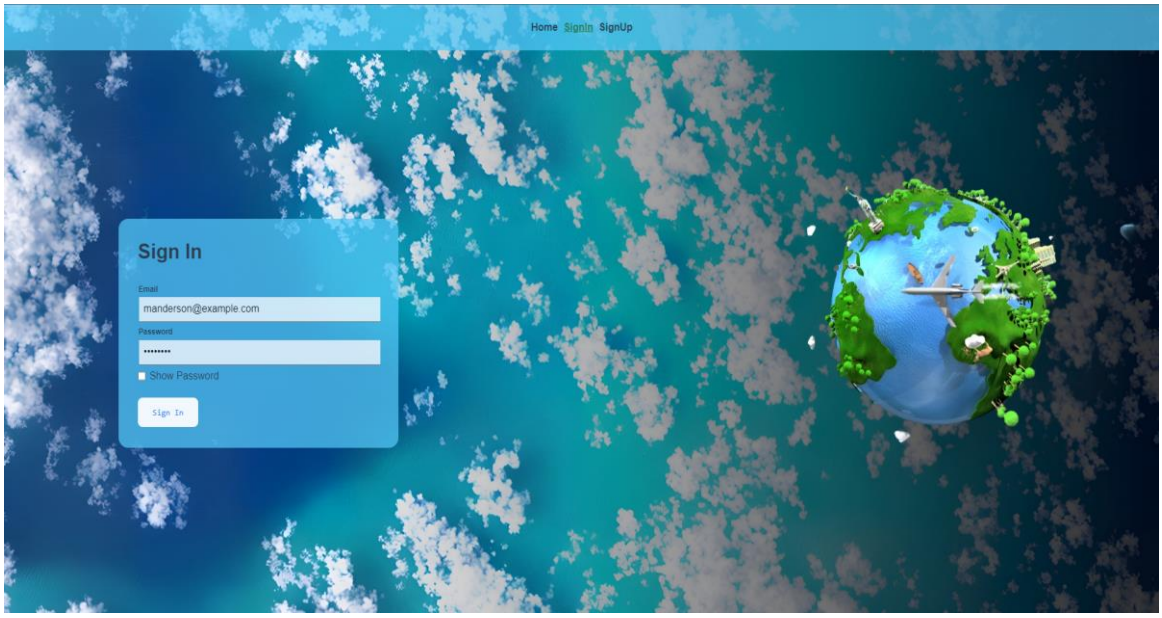


Рисунок 3.3 – Сторінка входу

Після успішного входу в систему, користувачу буде продемонстровано сторінку особистого профілю (рисунок 3.4). Якщо адміністратор ще не провів перевірку нового клієнта системи, то відповідний запис буде зображено. Функціонал обмежений. Користувач не зможе спілкуватись з іншими та шукати людей. Також після входу можна буде завантажити фото облікового запису.

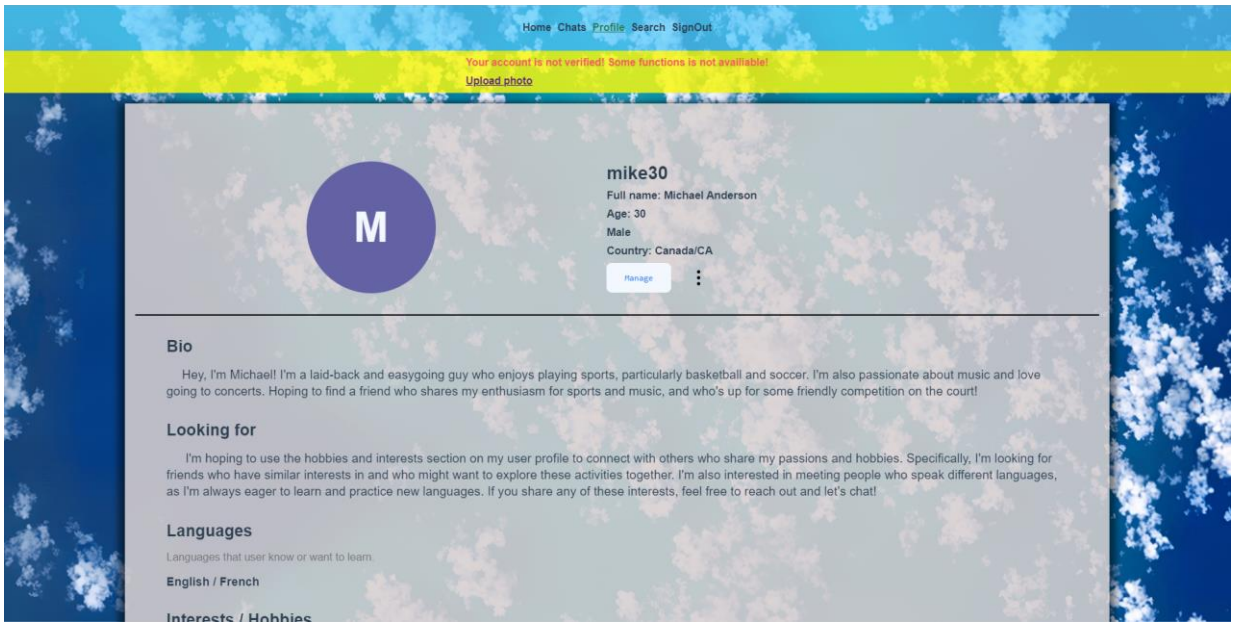


Рисунок 3.4 – Сторінка профілю неверифікованого користувача

Користувач може змінити дані про себе, для цього він натискає кнопку “Manage” на сторінці профіля і він потрапляє на сторінку редагування (рисунок 3.5).

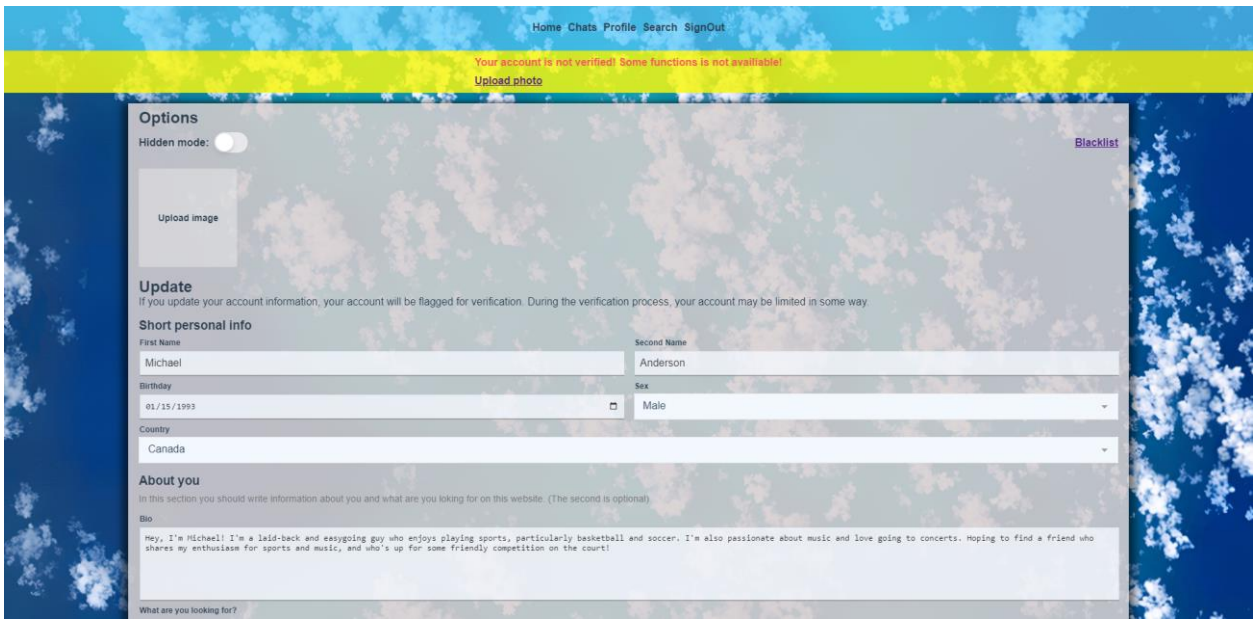


Рисунок 3.5 – Сторінка редагування профілю

Можна завантажити фото, керувати прихованим режимом, перейти на сторінку чорного списку, змінити дані про себе та пароль, чи взагалі видалити аккаунт. Також після завантаження фото профілю, буде можливість його змінити чи видалити (рисунок 3.6).

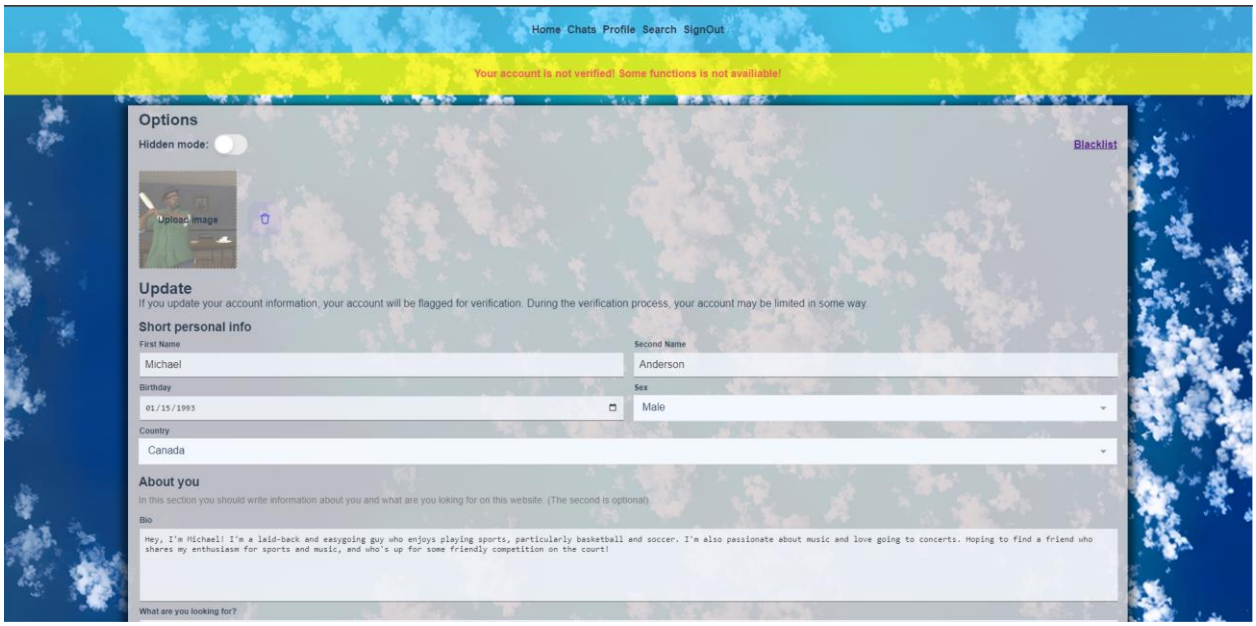


Рисунок 3.6 – Сторінка редагування профілю з завантаженим фото профілю

Якщо аккаунт пройшов перевірку, то користувачу буде доступний весь функціонал системи, зникне відповідний напис.

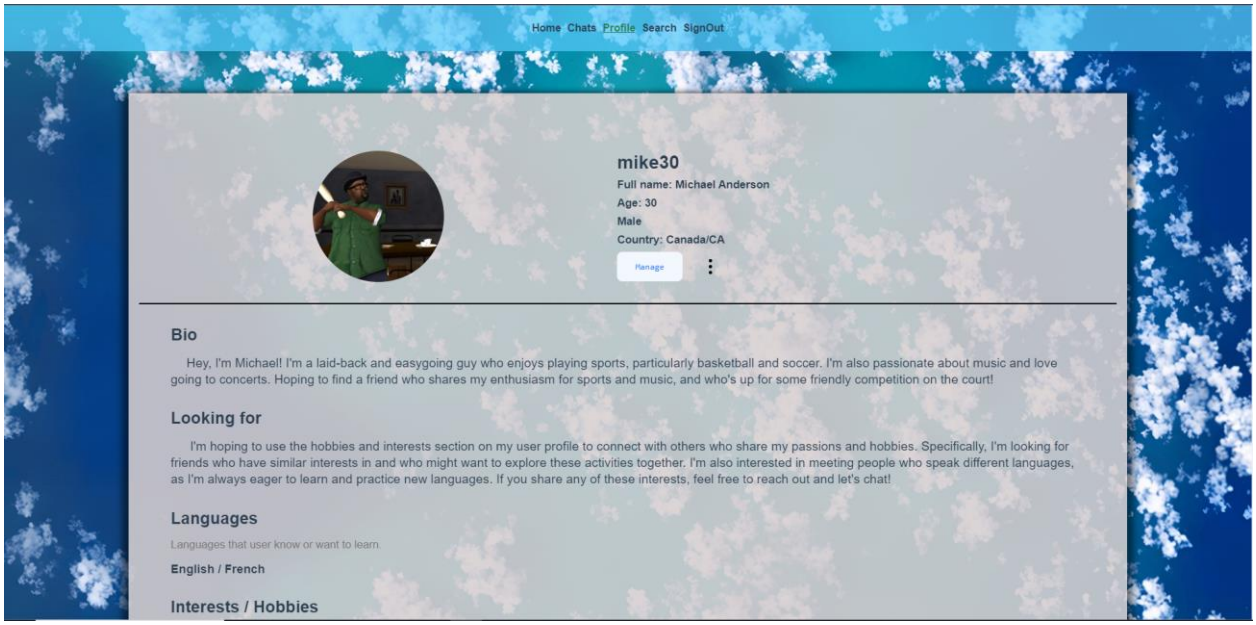


Рисунок 3.6 – Сторінка профілю, що пройшов перевірку

Для того щоб розпочати пошук друзів, достатньо перейти на сторінку пошуку (рисунок 3.7). Користувачу буде продемонстровано одразу людей, що мають нові облікові записи.

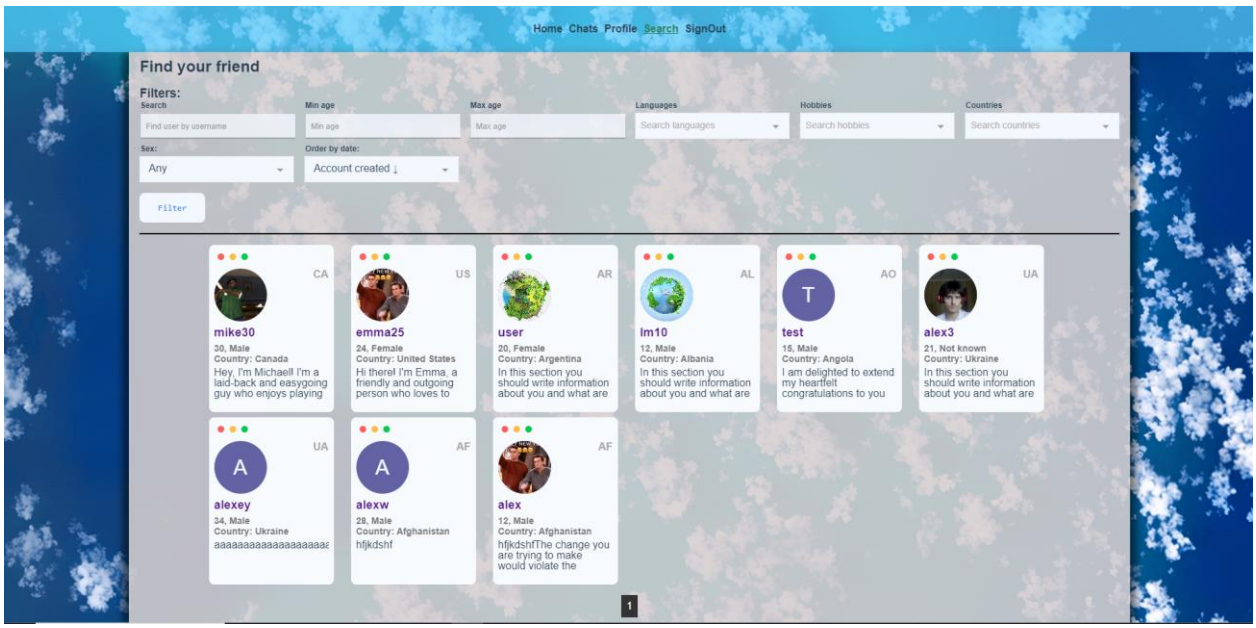


Рисунок 3.7 – Сторінка пошуку

Користувач може ввести потрібні параметри, щоб знайти собі однодумця. Серед фільтрів є можливість вказати логін, мінімальний та максимальний вік, обрати мови, хобі, країни, що цікавлять. Вказати стать та

відсортувати за найновішими чи найстарішими обліковими записами. На рисунку 3.8 продемонстровано фільтрацію за мінімальним та максимальним віком.

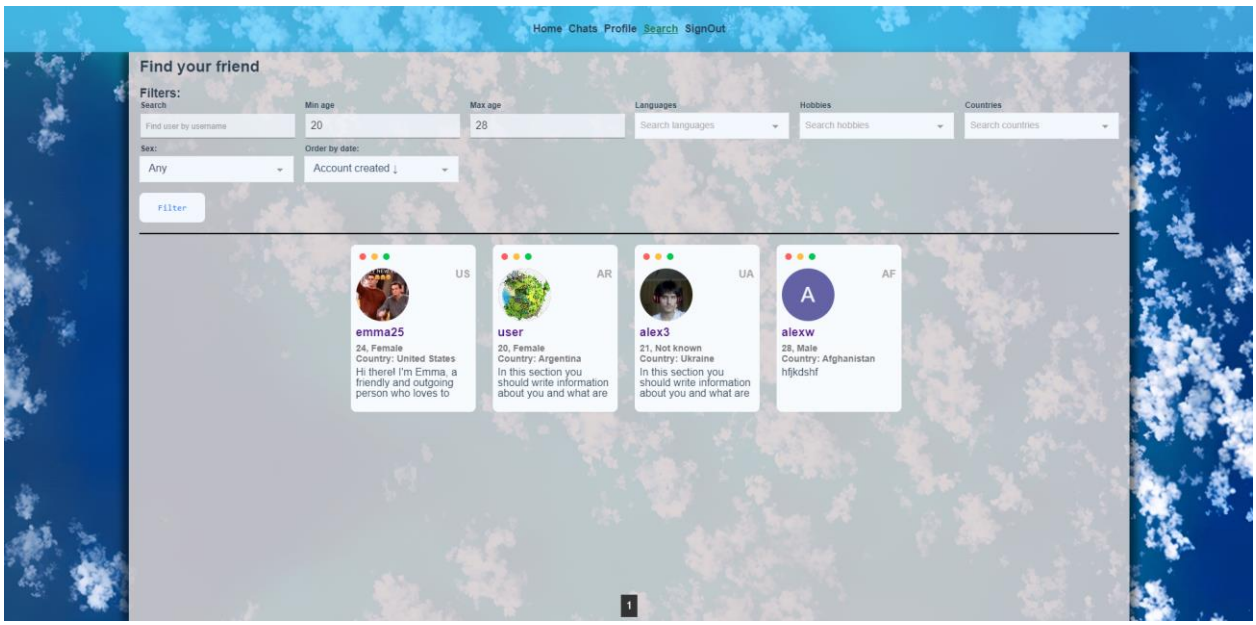


Рисунок 3.8 – Сторінка пошуку за фільтрацією за віком

Для того щоб подивитись детальну інформацію про користувача достатньо натиснути на його логін. Буде продемонстровано сторінку профілю (рисунок 3.9.

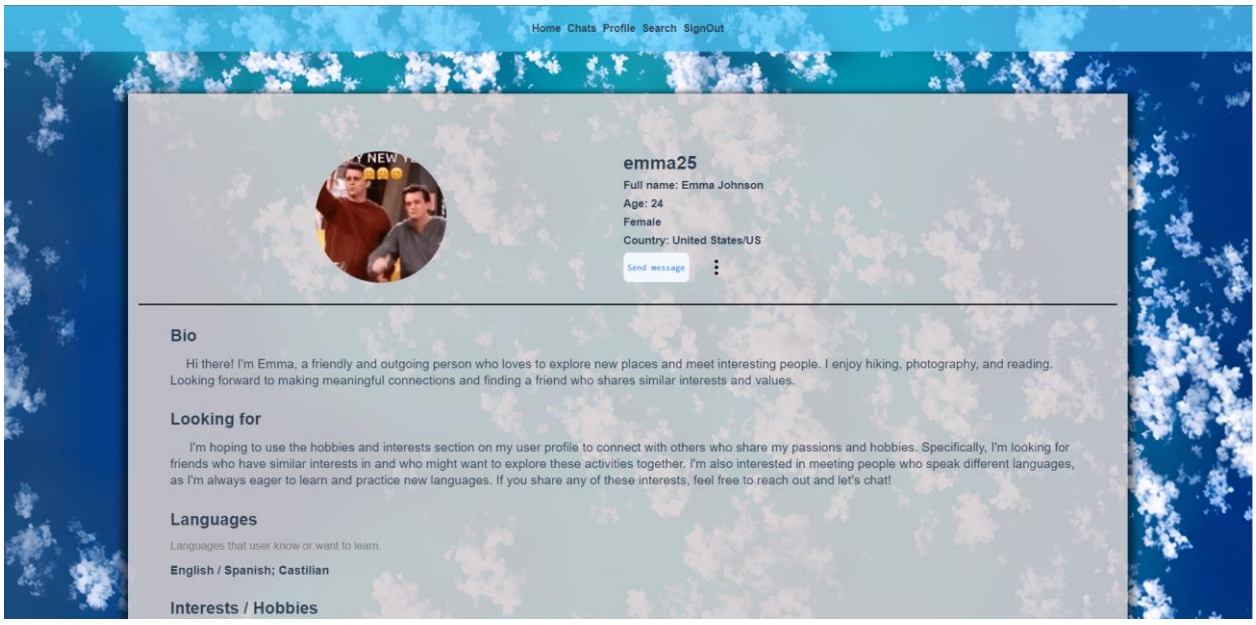


Рисунок 3.9 – Сторінка користувача з детальною інформацією

Щоб надіслати користувачу перше повідомлення, потрібно натиснути кнопку “Send message”, буде створено новий чат, якщо його ще немає,

надіслано автоматично згенероване повідомлення та переправлено на сторінку з чатами (рисунок 3.10).

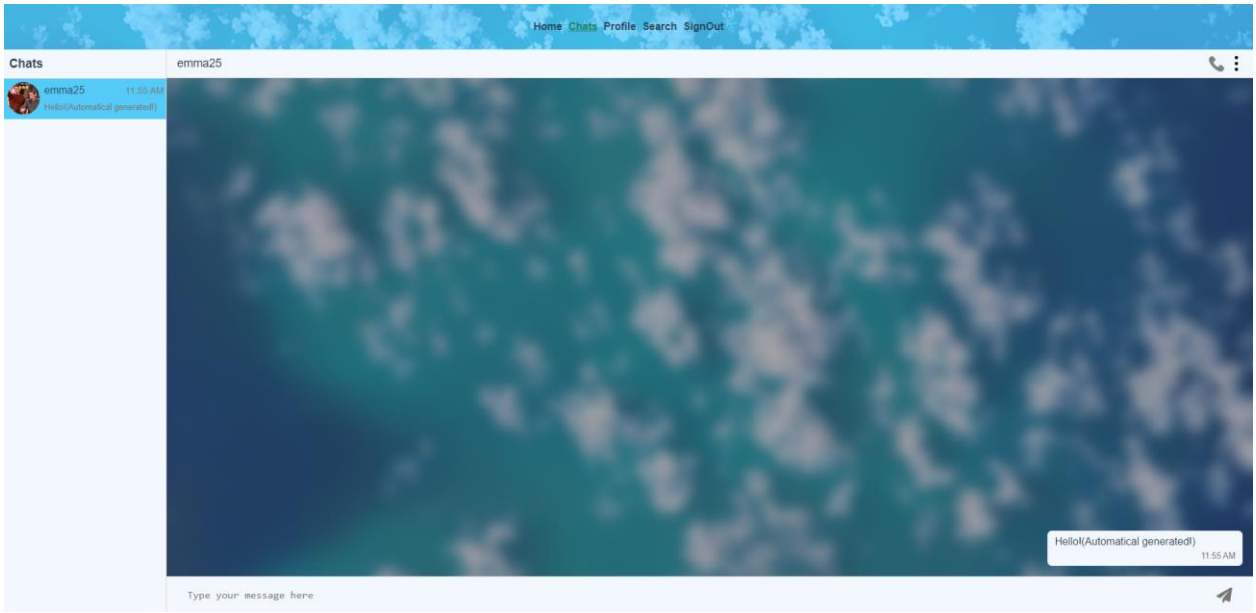


Рисунок 3.10 – Сторінка чатів після надсилання першого повідомлення

Для того щоб надіслати особисте повідомлення, потрібно його ввести в відповідне поле та натиснути іконку літачка. Повідомлення з’явиться в історії (рисунок 3.11).

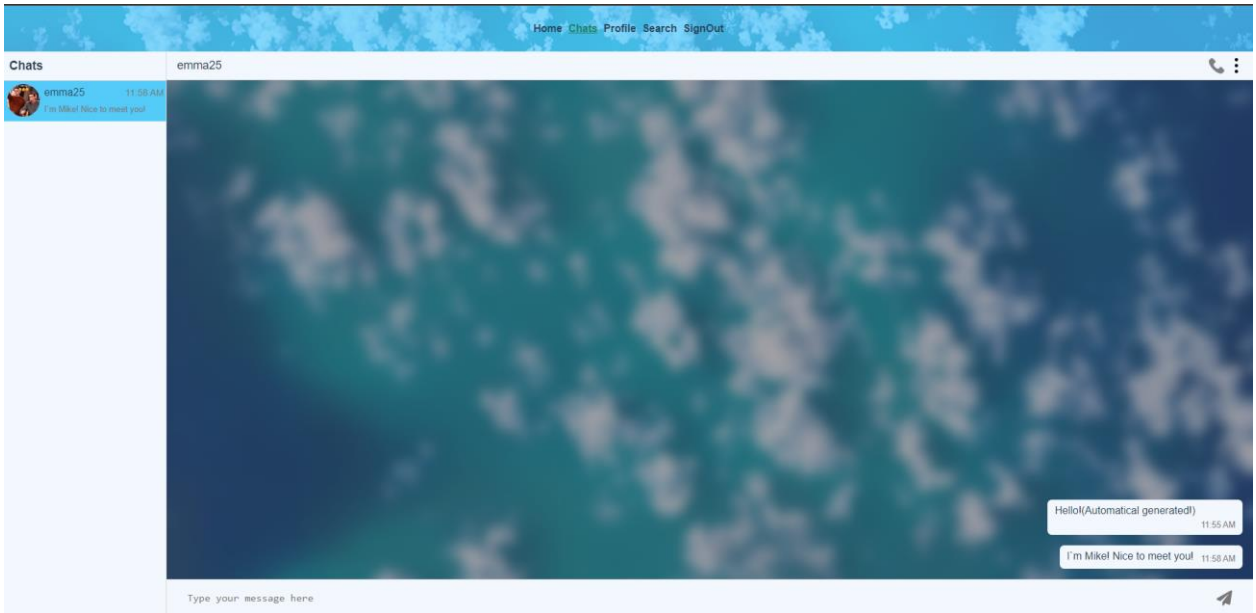


Рисунок 3.11 – Сторінка чатів після надсилання повідомлення

Для редагування власного повідомлення потрібно натиснути на нього та в контекстному меню (рисунок 3.12) обрати редагувати.

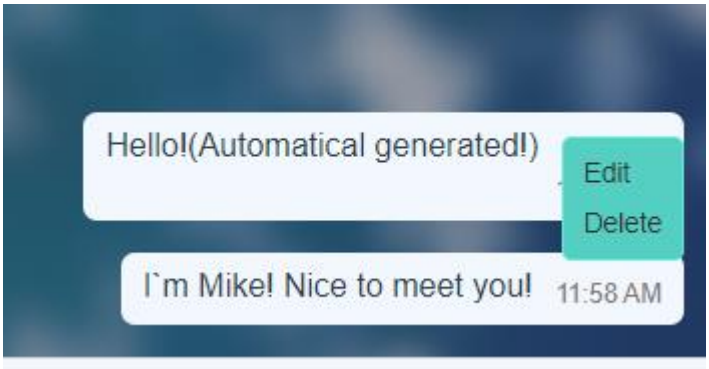


Рисунок 3.12 – Контекстне меню керування повідомленням

З’явиться нове вікно, в якому можна буде змінити зміст повідомлення (рисунок 3.13).

Рисунок 3.13 – Вікно редагування повідомлення

Після введення тексту, потрібно натиснути на кнопку “Edit”. Вікно закриється, відредаговане повідомлення буде помічене (рисунок 3.14).

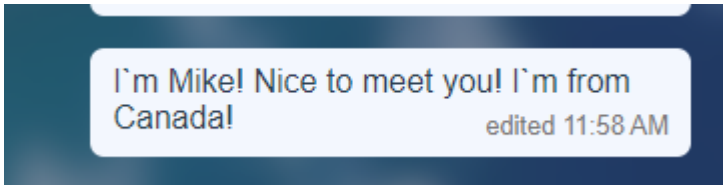


Рисунок 3.14 – Відредаговане повідомлення

Як тільки інший учасник чату увійде до системи, буде продемонстровано, що він з’явився в мережі і буде можливість зробити йому дзвінок (рисунок 3.15).

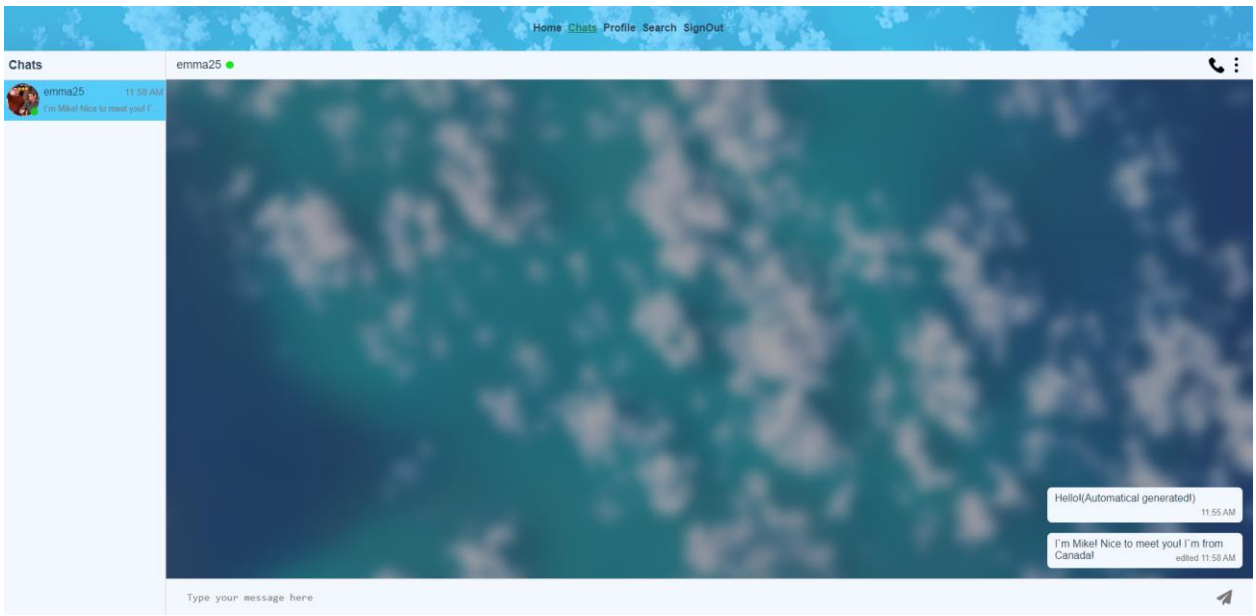


Рисунок 3.15 – Інший учасник чату увійшов в систему

Коли інший користувач відкриє спільний чат, то повідомлення будуть помічені прочитаними (рисунок 3.16).

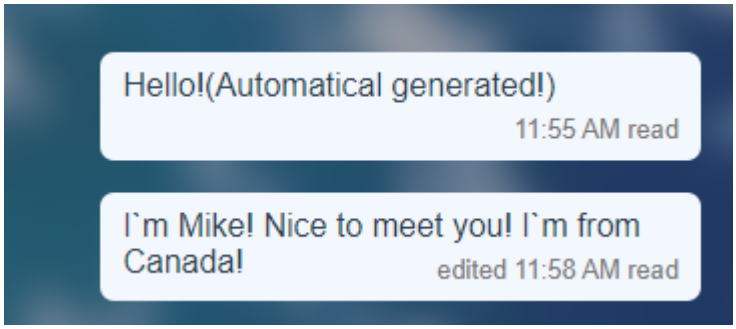


Рисунок 3.16 – Інший учасник чату прочитав надіслані повідомлення

Якщо інший учасник чату надішле повідомлення, то вони з'являться в історії (рисунок 3.17).

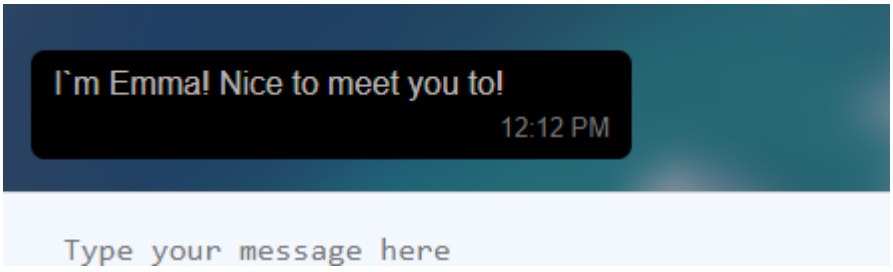


Рисунок 3.17 – Повідомлення від іншого користувача

Для того щоб видалити повідомлення потрібно обрати в контекстному меню пункт “Delete” (рисунок 3.12) та підтвердити свої дії. Воно зникне у двох користувачів одразу.

Для того щоб видалити чат чи додати користувача до чорного списку потрібно натиснути на кнопку з трьома крапками в верхньому правому кутку. З’явиться меню з пунктами (рисунок 3.18).

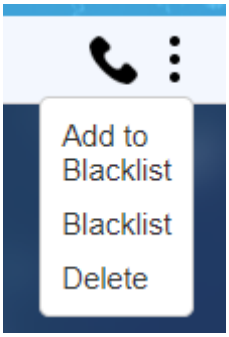


Рисунок 3.18 – Меню керування чатом

При додаванні користувача до чорного списку, чат з ним буде видалено.

Для здійснення дзвінка достатньо натиснути на кнопку з іконкою телефону в верхньому правому куті. Користувачу буде направлено на іншу сторінку (рисунок 3.19).

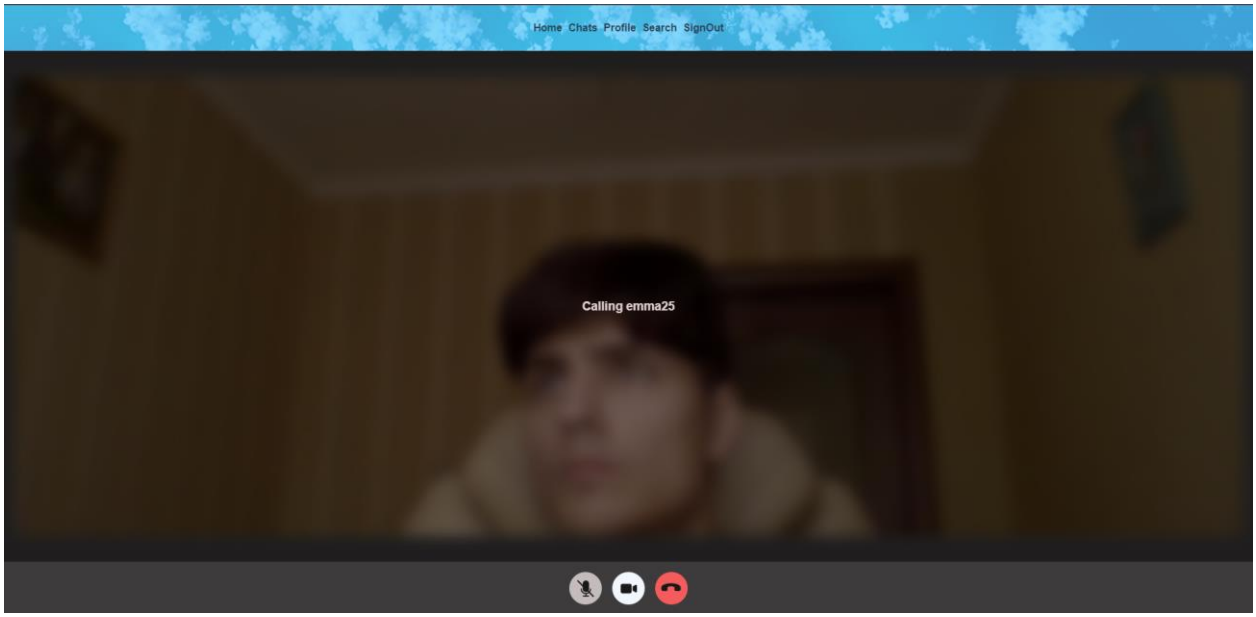


Рисунок 3.19 – Сторінка дзвінка після його створення

У іншого учасника чату з’явиться нове вікно, де він зможе відхилити чи прийняти дзвінок (рисунок 3.20).

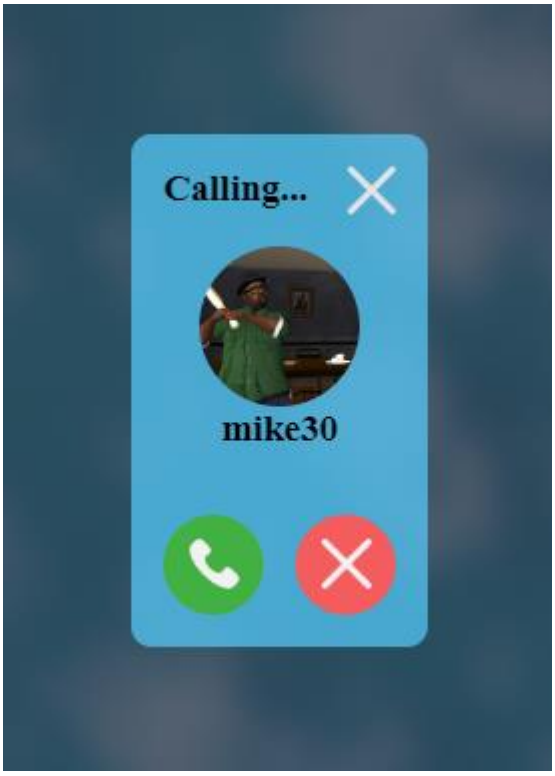


Рисунок 3.20 – Вікно користувача, якому дзвонять

Якщо отримувач відхилить дзвінок, то в нього зникне вікно, а в ініціатора виклика буде виведено відповідне повідомлення (рисунок 3.21) та його буде перенаправлено на сторінку з чатами через 5 секунд.

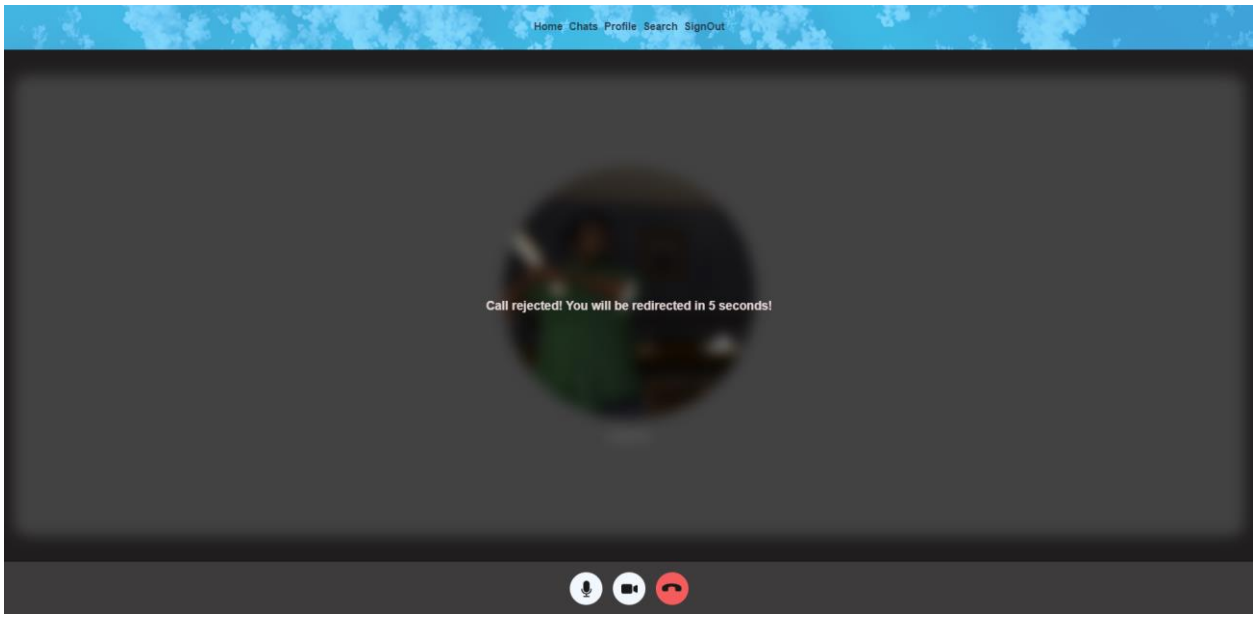


Рисунок 3.21 – Повідомлення, після відхилення виклику

Якщо ініціатор закінчить дзвінок до того, як інший користувач підтвердить чи відхилить виклик, то вікно зникне.

Якщо отримувач відповість на дзвінок, то його буде переправлено на сторінку виклику (рисунок 3.22).

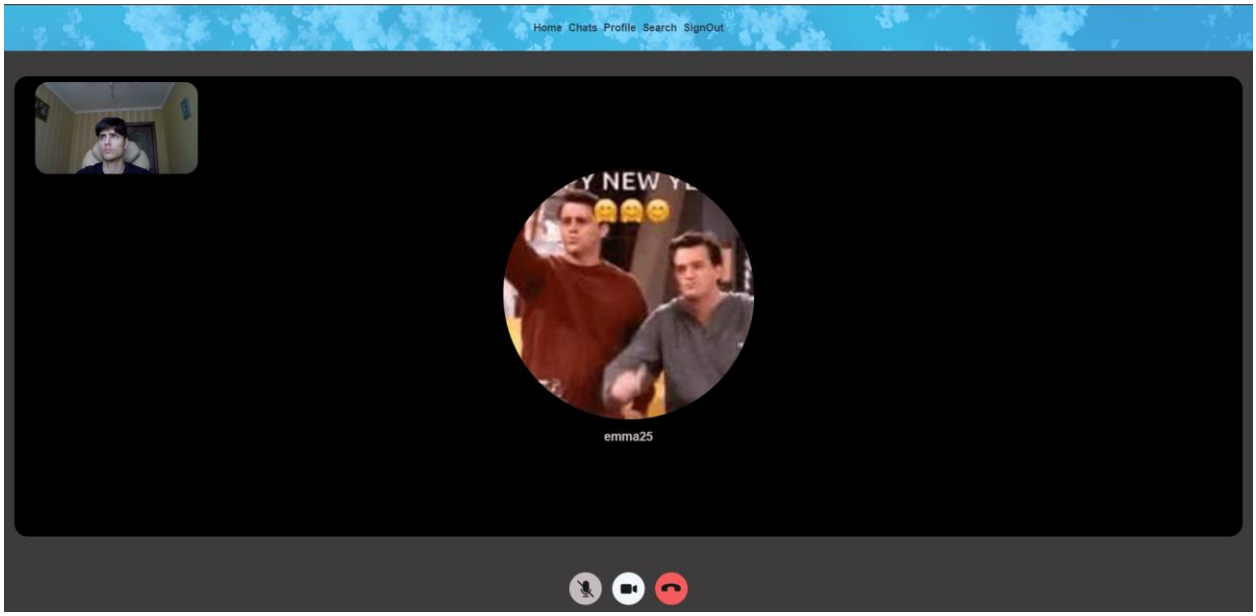


Рисунок 3.22 – Екран після приєднання до дзвінка

Користувач може керувати камерою та мікрофоном. Після завершення виклику, користувача, що натиснув на кнопку буде одразу перенаправлено на сторінку з чатом, а іншого учасника через 5 секунд. Також йому буде виведено відповідне повідомлення (рисунок 3.23).

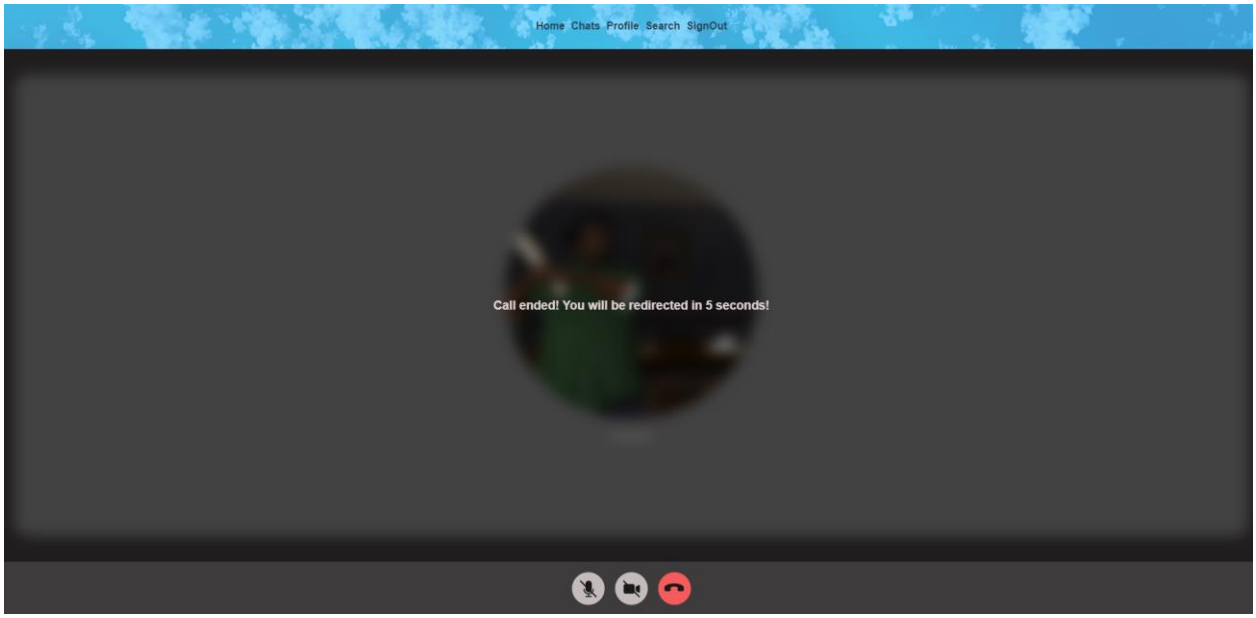


Рисунок 3.23 – Екран після завершення дзвінка

Для керування чорним списком, потрібно перейти на відповідну сторінку. Посилання є на наступних сторінках: профіль, редагування аккауну

та чати. Для видалення користувача зі списку, достатньо натиснути кнопку з іконкою смітника навпроти потрібного логіну (рисунок 3.24).

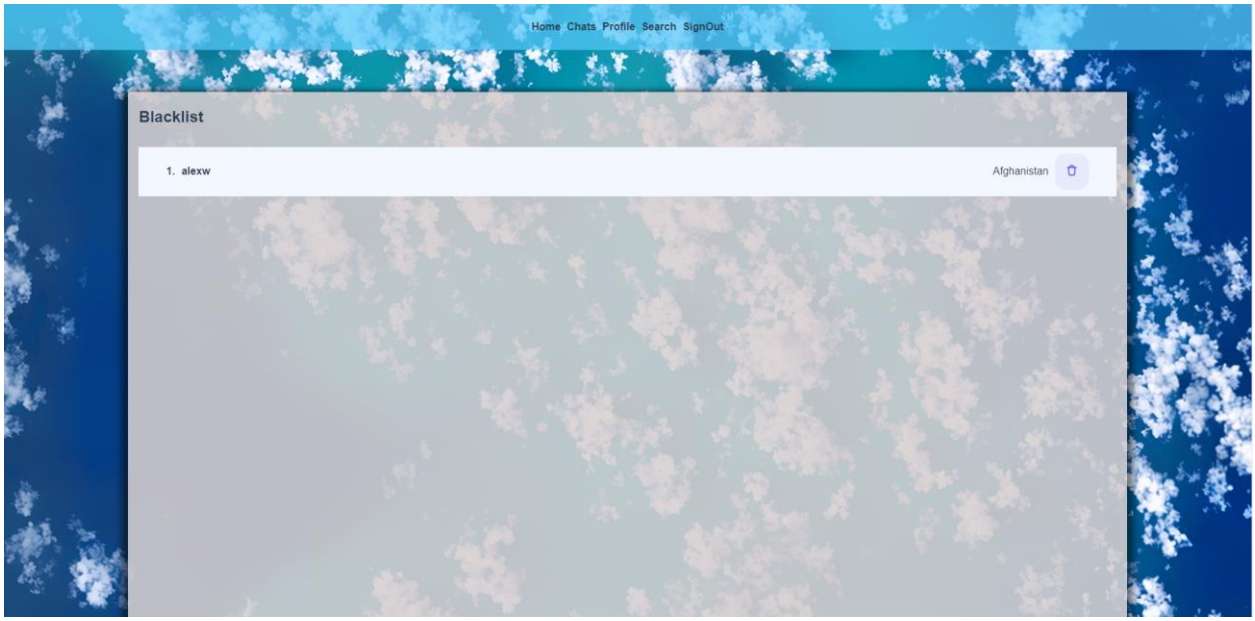


Рисунок 3.24 – Сторінка чорного списку

3.2 Керівництво адміністратора

Після входу в систему з аккаунту адміністратора, його буде перенаправлено на сторінку профілю. Де додатково наведено інформацію про статус блокування, верифікації, режиму прихованості користувача (рисунок 3.25).

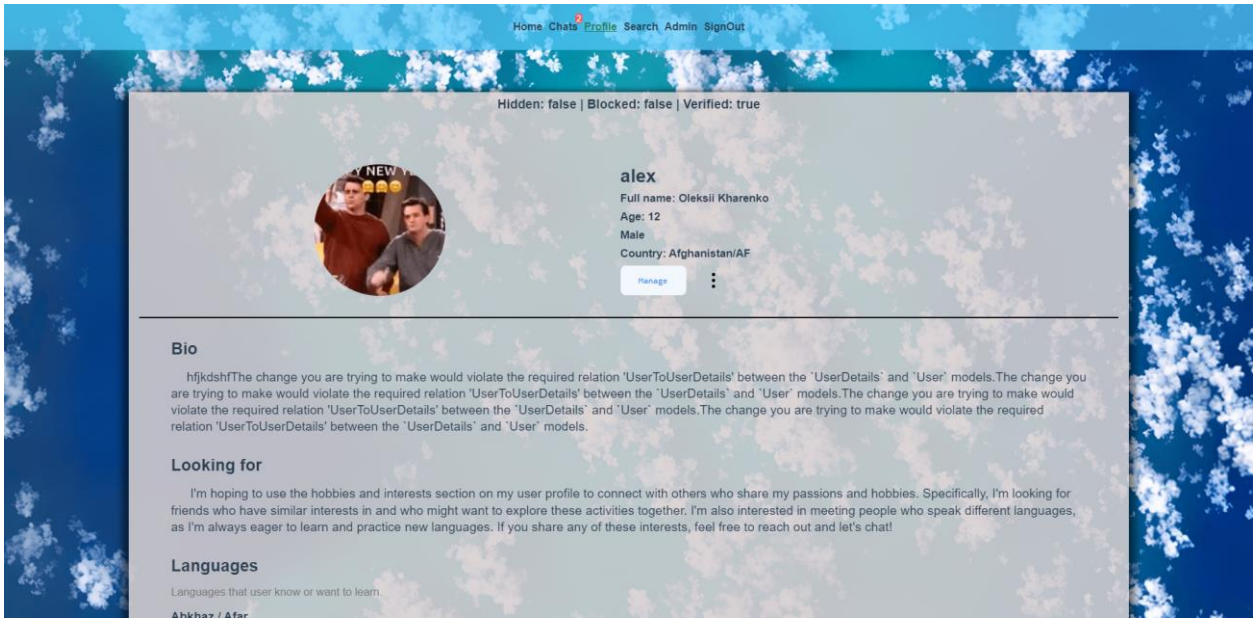


Рисунок 3.25 – Сторінка профілю адміністратора

Для керування користувачами потрібно перейти на сторінку адміністрування (рисунок 3.26).

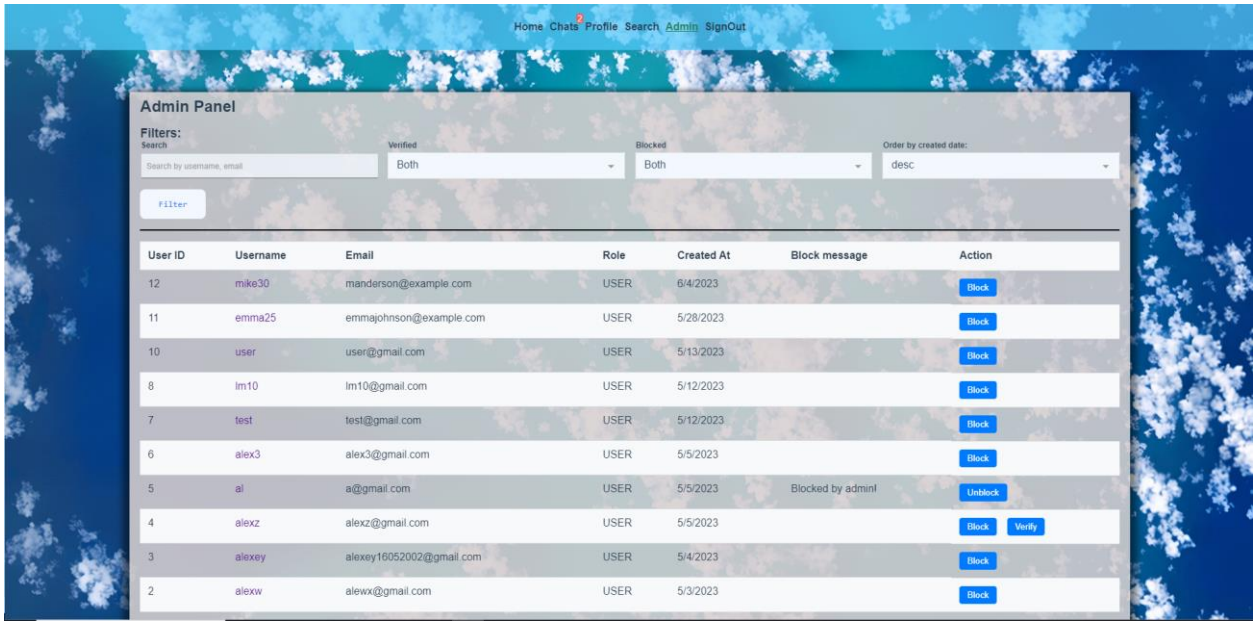


Рисунок 3.26 – Сторінка адміністрування

Є можливість фільтрації за статусом блокування чи верифікації, можна знайти за електронною адресою чи логіном.

Для того щоб верифікувати користувача, адміністратор повинен перейти на його сторінку та перевірити інформацію про нього. Потім повернутись до панелі керування та натиснути кнопку “Verify”, якщо все добре. Якщо ні, то потрібно надіслати листа на електрону пошту за допомогою стороннього сервісу.

Для того щоб заблокувати користувача, потрібно натиснути на кнопку “Block”. З’явиться нове вікно, де потрібно вказати причину блокування (рисунок 3.27).

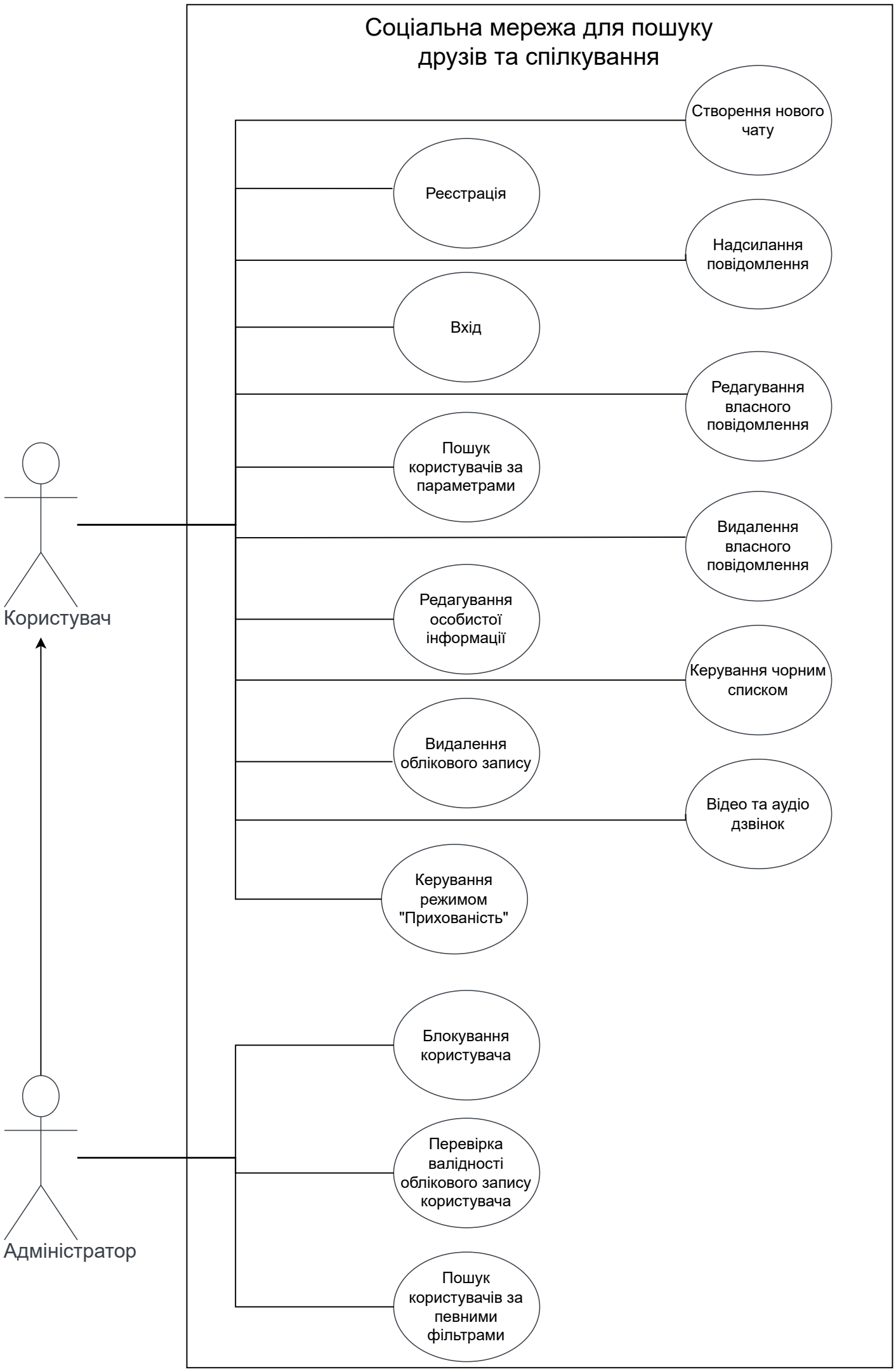
Рисунок 3.27 – Вікно введення причини блокування

Після підтвердження, користувачу буде заборонено доступ до системи. Він не зможе увійти до системи, всі чати з ним будуть видаленні. З’явиться кнопка розблокування (рисунок 3.28).

Рисунок 3.28 – Запис після блокування користувача

Для того щоб розблокувати користувача, достатньо натиснути кнопку “Unblock”. Причина блокування зникне (рисунок 3.29).

Рисунок 3.29 – Запис після розблокування користувача

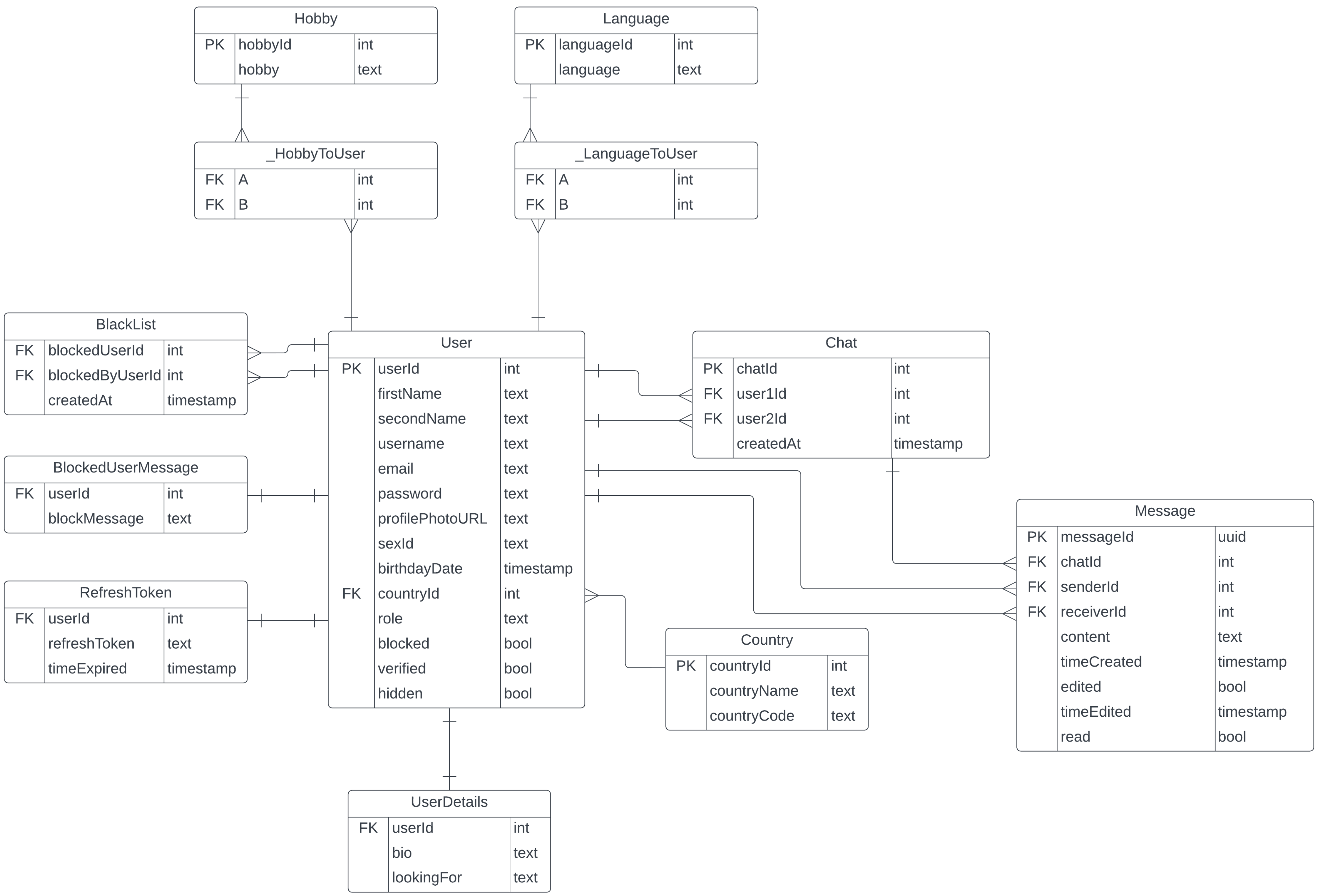


					КПІ.ІП-9226.045440.06.99.ССВ
Зм.	Арк.	№ докум.	Підп.	Дата	
Розроб.		Харенко О.С.			Схема структурна варіантів використань
Перев.		Зенів І.О.			
Т. Кон.					
Н. Кон.		Ліщук К.І.			Соціальна мережа для пошуку друзів та спілкування
Затв.		Жаріков Е.В.			

Лит.	Арк.	Аркушів
Аркуш	Аркушів	

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-92

КПІ.ІП-9226.045440.07.99.СБД



Зм.	Арк.	№ документа	Підпис	Дата
Розробив		Харенко О.С.		
Перевірив		Зенів І.О.		
Т. кон.				
Н. кон.		Ліщук К.І.		
Затвердив		Жаріков Е.В.		

КПІ.ІП-9226.045440.07.99.СБД

Схема бази даних

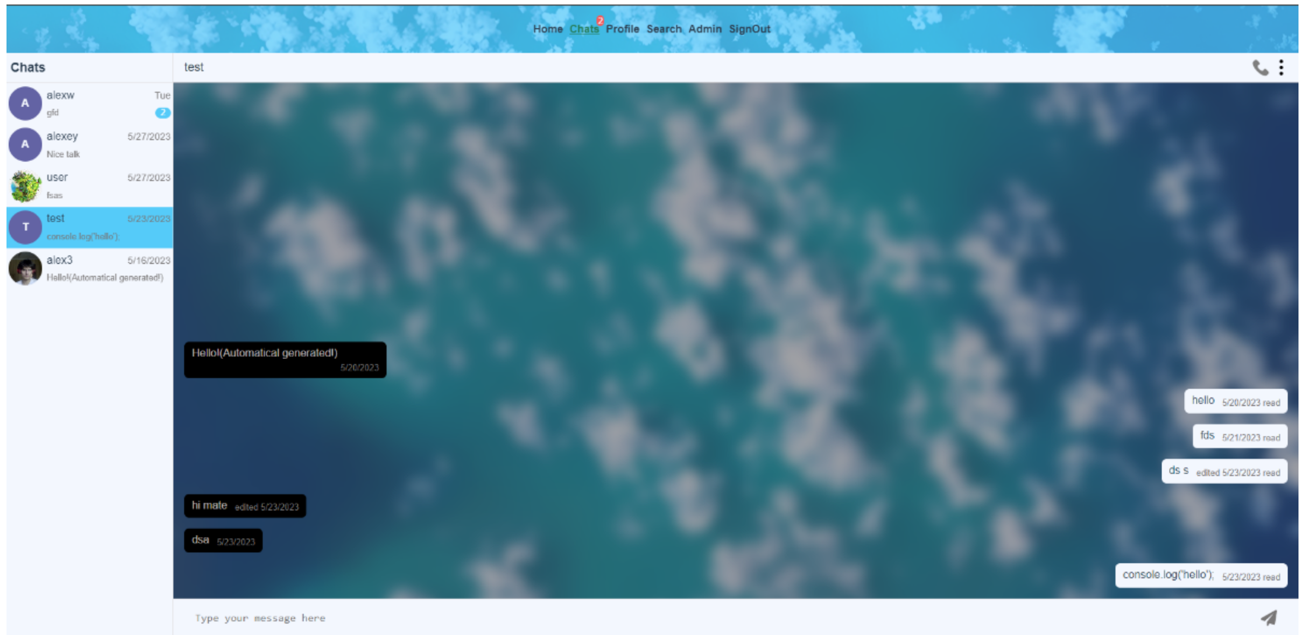
Літера	Маса	Масштаб

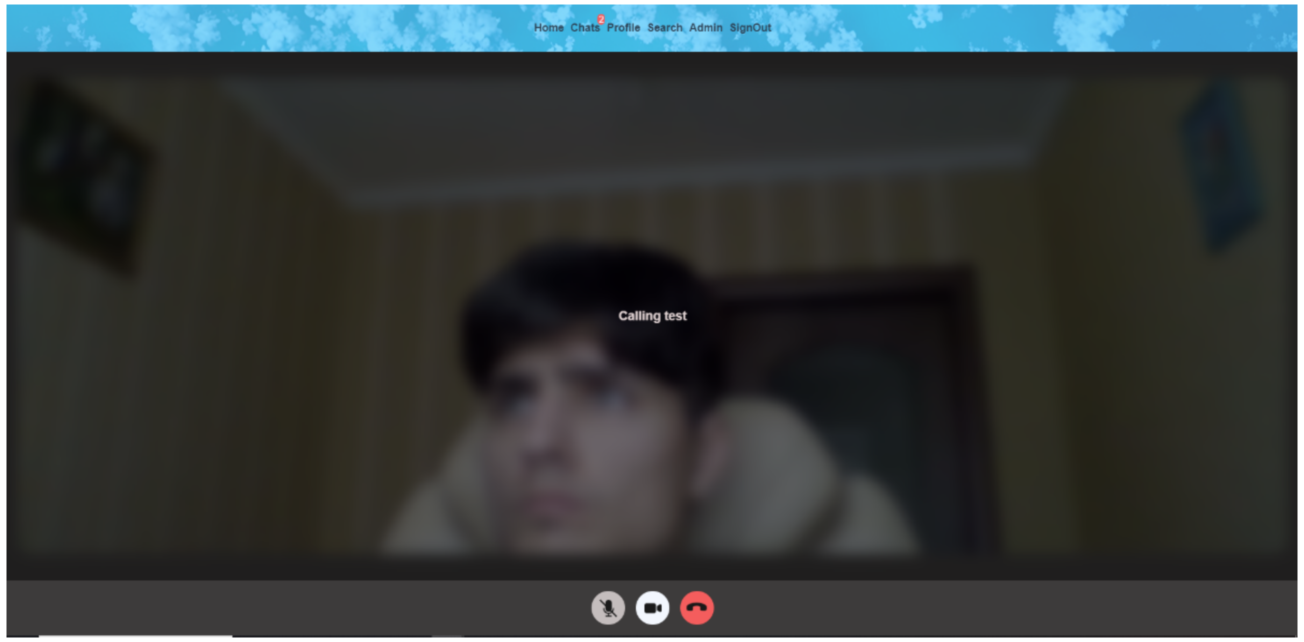
Аркуш	Аркушів

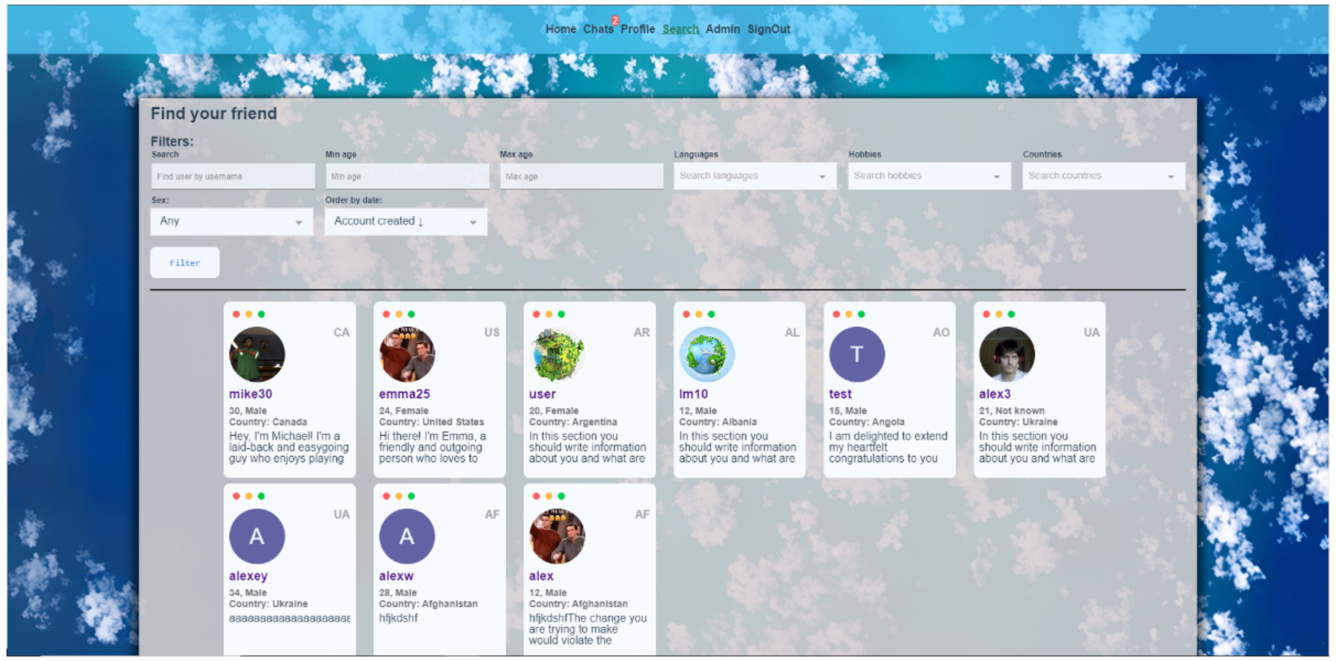
Соціальна мережа для пошуку друзів та спілкування

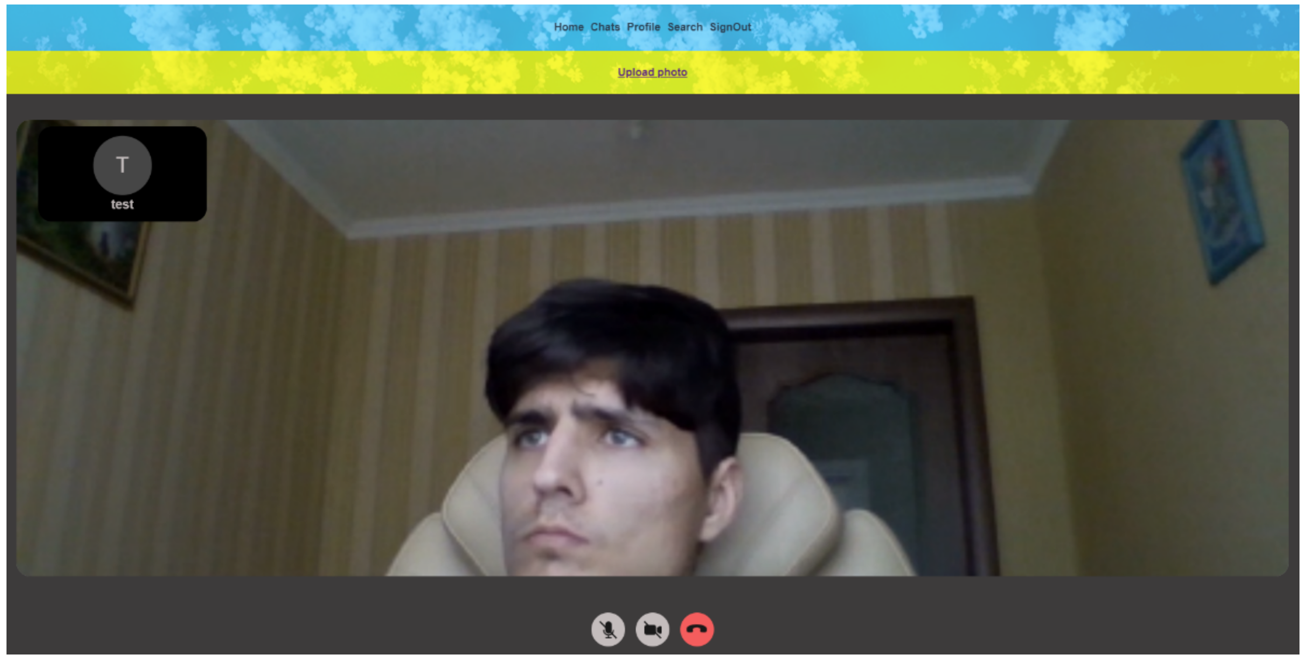
КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-92

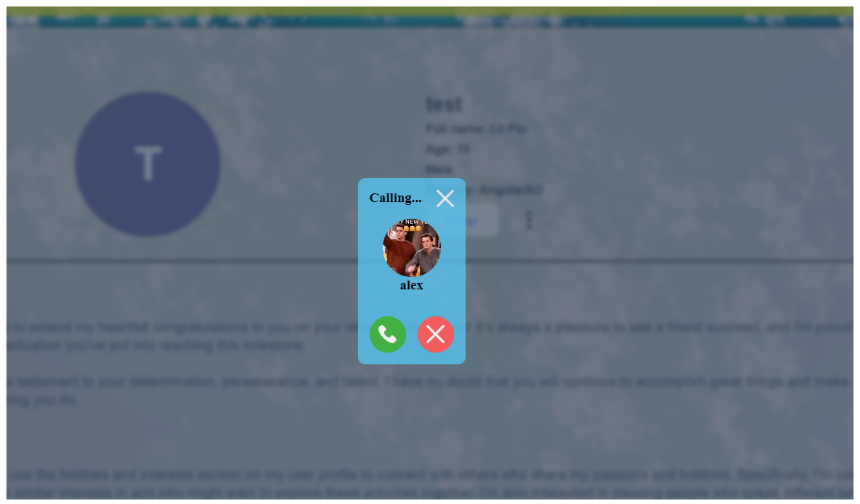
КПІ.ІП-9226.045440.08.99.КЕ











					КПІ.ІП-9226.045440.08.99.КЕ			
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера	Маса	Масштаб
Розробив		Харенко О.С.						
Перевірив		Зенів І.О.				Аркуш		Аркушів
Т. кон.					Соціальна мережа для пошуку друзів та спілкування	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-92		
Н. кон.		Ліщук К.І.						
Затвердив		Жаріков Е.В.						