Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка вебсистеми для знайомства людей з використанням фреймворку Angular»

Виконав: студент 4 курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Куксін Д.О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Черноволик Г.О.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Тарновський М. Г.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Куксіну Денису Олексійовичу

1. Тема роботи – «Розробка вебсистеми для знайомства людей з використанням фреймворку Angular»

Керівник роботи: Черноволик Г.О., к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи:.

4. Зміст розрахунково-пояснювальної записки: вступ; fналіз стану розвитку вебсистем для знайомства людей; розробка структури та алгоритмів програмного застосунку; розробка вебсистеми; тестування програмного забезпечення; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Черноволик Г.О., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку вебсистем для знайомства людей	25.03.25 - 05.04.25	вик.
2	Розробка структури та алгоритмів програмного застосунку	06.04.25 - 18.04.25	вик.
3	Варіантний аналіз та вибір мови програмування розробки	19.04.25 - 21.04.25	вик.
4	Розробка вебсистеми	22.04.25 - 17.05.25	вик.
5	Тестування програмного забезпечення	18.05.25 - 25.05.25	вик.
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25	вик.

Студент (підпис) Куксін Д.О. (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи (підпис) Черноволик Г.О. (прізвище та ініціали)

АНОТАЦІЯ

УДК 004:005.9

Куксін Д.О. Розробка вебсистеми для знайомства людей з використанням фреймворку Angular : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 94 с.

У бакалаврській кваліфікаційній роботі було розроблено вебсистему для знайомства людей з використанням Angular. Розроблена вебсистема дозволяє користувачам безпечно реєструватися та входити в обліковий запис, створювати й редагувати власні профілі з можливістю завантаження фотографій у хмарне сховище, швидко знаходити сумісних співрозмовників за допомогою механізму «лайків» і миттєвого «матчу», а також обмінюватися повідомленнями в реальному часі.

Було проведено аналіз інформаційного забезпечення системи, описано, які дані використовуються та генеруються при роботі системи. Розроблено архітектуру системи. Розроблено метод формування репутації користувача, метод вказання точок зустрічі. Побудовано загальний алгоритм роботи системи, продемонстровано його у вигляді відповідної блок-схеми, на якій відображено увесь процес роботи системи.

Програмне забезпечення розроблено із використанням середовища розробки Visual Studio Code та мови програмування Javascript. У процесі розробки використовувались сучасні бібліотеки та інструменти, такі як Angular, ASP.NET, SignalR. Під час тестування було перевірено працездатність основного функціоналу системи та підтверджено коректність реалізованих методів.

Ключові слова: знайомства, Angular, вебсистема.

ABSTRACT

UDC 004:005.9

Kuksin D.O. Development of a web system for meeting people using the Angular framework: bachelor's qualification work in specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 94 p.

In the bachelor's qualification work, a web system for meeting people using Angular was developed. The developed web system allows users to securely register and log in to an account, create and edit their own profiles with the ability to upload photos to cloud storage, quickly find compatible interlocutors using the "likes" and instant "match" mechanism, as well as exchange messages in real time.

An analysis of the information support of the system was carried out, it was described what data is used and generated during the system's operation. The system architecture was developed. A method for forming a user's reputation, a method for specifying meeting points were developed. A general algorithm for the system's operation was built and demonstrated in the form of a corresponding flowchart, which displays the entire process of the system's operation.

The software was developed using the Visual Studio Code development environment and the Javascript programming language. Modern libraries and tools such as Angular, ASP.NET, SignalR were used in the development process. During testing, the performance of the main functionality of the system was checked and the correctness of the implemented methods was confirmed.

Keywords: dating, Angular, web system.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Зростання популярності онлайн-знайомств обумовлене низкою соціокультурних і технологічних чинників. По-перше, зміни в стилі життя і культурі знайомств: сучасні люди відкладають вступ у шлюб і більше часу проводять у професійній діяльності або навчанні, тому мають менше можливостей для традиційних знайомств [1]. Онлайн-платформи допомагають заповнити цю прогалину, надаючи зручний спосіб зустріти нових людей поза межами свого звичного оточення. По-друге, технологічний прогрес зробив свій внесок: майже кожен має доступ до смартфона та інтернету, що спрощує використання додатків для знайомств будь-де і будь-коли.

Поширення соціальних мереж також зняло стигму з онлайн-знайомств – спілкування через інтернет стало звичним і прийнятним. В економічному плані великі інвестиції в цю галузь (ринок оцінюється в мільярди доларів) спонукали появу численних сервісів та рекламу, що підвищує обізнаність користувачів про такі платформи. У сукупності ці фактори – зміна суспільних норм, технологічна доступність та економічна привабливість – сприяють стрімкому поширенню онлайн-систем знайомств у всьому світі.

Онлайн-знайомства стали масовим явищем у сучасному світі. Мільйони людей шукають партнерів через додатки та вебсайти для знайомств, і ця аудиторія невпинно зростає. За прогнозами, кількість користувачів сервісів онлайн-знайомств у світі досягне приблизно 462,5 мільйонів осіб до 2029 року, що свідчить про стабільну тенденцію зростання [2].

В економічному вимірі індустрія онлайн-знайомств також демонструє значні показники: глобальний ринок оцінили майже у 9,65 мільярда доларів США у 2022 році, і очікується середньорічне зростання ~7,4% протягом 2023–2030 років [3]. Ці статистичні дані підтверджують, що розробка вебсистеми знайомств є надзвичайно актуальною та має значну цільову аудиторію.

Водночас існуючі платформи часто мають певні недоліки: складні

алгоритми підбору, що не завжди враховують індивідуальні потреби користувачів, недосконалі системи безпеки, високу вартість преміум-функцій, а також обмежені можливості для створення якісних профілів. Крім того, багато сервісів орієнтовані переважно на західні ринки і не враховують культурні особливості та потреби локальних спільнот. Постійне зростання попиту на більш персоналізовані, безпечні та доступні рішення створює простір для інноваційних підходів у розробці нових вебсистем знайомств.

Додатково, пандемія COVID-19 суттєво прискорила цифровізацію міжособистісного спілкування, що призвело до ще більшого зростання інтересу до онлайн-знайомств серед різних вікових груп. Молоді фахівці, які активно використовують цифрові технології у всіх сферах життя, очікують від платформ знайомств високої функціональності, інтуїтивного інтерфейсу та інноваційних можливостей для самовираження та пошуку сумісних партнерів.

Отже, актуальною є розробка сучасної вебсистеми знайомств.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є збільшення рівня безпеки, сихронізація змін користувачів за рахунок спеціалізованої вебсистеми для знайомства людей.

Основними задачами дослідження є:

- розробити архітектуру вебзастосунку для знайомства людей;
- розробити інтерфейс користувача застосунку;
- розробити метод формування репутації користувача;
- розробити метод вказання точок зустрічі;
- розробити алгоритми роботи застосунку;
- розробити функціонал застосунку;
- провести тестування розробленого застосунку.

Об'єкт дослідження – процес розробки вебсистеми для знайомства людей з використанням фреймворку Angular.

Предмет дослідження – методи та засоби розробки вебсистеми для знайомства людей з використанням фреймворку Angular.

Методи дослідження. У процесі досліджень використовувались такі методи дослідження: аналіз аналогів та порівняння їх з власною розробкою для визначення та постановки задач розробки; методи геолокації для пошуку людей; теорія користувацького досвіду та принципів UX/UI дизайну; теорія веброзробки для створення вебінтерфейсів; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень; методи тестування вебзастосунків.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод формування репутації користувача, який, на відміну від відомих, забезпечує комплексне оцінювання поведінки користувачів через багатофакторний підхід із урахуванням як позитивних (взаємні лайки, повідомлення, успішні зустрічі), так і негативних взаємодій (скарги, блокування), динамічне оновлення репутаційного балу в режимі реального часу, автоматичну модерацію з можливістю ручного коригування та інтеграцію з алгоритмом підбору, що дозволяє підвищити якість рекомендацій та рівень безпеки користувачів.

2. Подальшого розвитку отримав метод вказання точок зустрічі, який, на відміну від відомих, поєднує в собі інтерактивний вибір локації через інтеграцію з картографічними сервісами, валідацію безпечності обраних точок, можливість додавання текстових описів та фотографій для кращої ідентифікації місця, миттєве оновлення, що дозволяє синхронізацію змін між користувачами, автоматичну генерацію маршрутів до місця зустрічі та аналіз популярних локацій для майбутніх рекомендацій.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що створена вебсистема забезпечує платформу для швидкого встановлення контактів між користувачами.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У роботі [4],

виконаній у співавторстві, автору належать: порівняльний аналіз можливостей сучасних додатків для знайомств, переваги та недоліки додатків для знайомств.

Апробація результатів роботи. Описані у бакалаврській кваліфікаційній роботі положення доповідались на конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025)».

Публікації. Результати роботи були опубліковані в матеріалах конференції «LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025)» [4].

1 АНАЛІЗ СТАНУ РОЗВИТКУ ВЕБСИСТЕМ ДЛЯ ЗНАЙОМСТВА ЛЮДЕЙ

1.1 Аналіз стану питання розробки вебсистем для знайомства людей

Вебсистеми знайомств виникли як відповідь на конкретні потреби та проблеми, з якими стикаються люди в процесі пошуку партнерів. Географічні та соціальні обмеження значно послаблюються: через інтернет людина може познайомитися з партнерами за межами свого кола спілкування чи міста, розширюючи таким чином вибір потенційних пар.

Онлайн-знайомства роблять пошук більш ефективним і зручним – користувачі можуть швидко переглядати анкети, застосовувати фільтри за інтересами чи цінностями і знаходити людей зі схожими характеристиками. Дослідження підтверджують, що користувачі цінують легкість, швидкість та мінімальні зусилля при онлайн-пошуку партнера.

Алгоритми підбору пар дозволяють знаходити більш сумісних партнерів за заданими критеріями, що було б важко реалізувати офлайн. Крім того, такі системи частково вирішують проблему сором’язливості чи браку часу – люди, які занадто зайняті або не схильні до активних знайомств у реальному житті, отримують зручний інструмент для налагодження особистого життя. Онлайн-платформи також надають контроль користувачу над взаємодією – наприклад, можна обмежувати коло осіб, які можуть писати повідомлення, або блокувати небажаних співрозмовників.

Ще одним ключовим аспектом є безпека та приватність: сучасні платформи активно впроваджують механізми верифікації профілів, такі як підтвердження фотографій або інтеграція з соціальними мережами, щоб знизити кількість фейкових акаунтів і шахрайства. Крім того, застосовуються алгоритми автоматичної модерації контенту та чат-боти, що перевіряють повідомлення на наявність образ чи небажаного контенту. Спеціальні налаштування приватності дозволяють користувачам контролювати, хто бачить їхній профіль і які дані передаються іншим учасникам. Такі заходи підвищують

довіру до сервісу і сприяють безпечнішому знайомству онлайн.

Індустрія знайомств постійно розвивається, адаптуючись до потреб користувачів та технологічних трендів. Останніми роками спостерігається інтеграція нових форматів спілкування: все популярнішими стають відео- та аудіодзвінки безпосередньо в застосунках. Майже половина користувачів (близько 48 %) заявляють, що воліють більше спілкуватися через відеочат перед особистою зустріччю [5].

Відповідно, сервіси впровадили функції відеочату та голосових повідомлень, щоб пари могли поспілкуватися “вживу” онлайн і встановити комфорт до офлайн-побачення.

Друга тенденція – зростання уваги до штучного інтелекту та алгоритмів: платформи все більше вдосконалюють підбір сумісних партнерів, використовуючи машинне навчання для аналізу вподобань, поведінки користувачів і навіть їхнього стилю листування. З’являються навіть ШІ-чатботи, що можуть пропонувати теми для розмов чи допомагати заповнити профіль.

Отже, системи знайомств вирішують одразу кілька проблем: розширюють можливості пошуку, підвищують його результативність і забезпечують комфорт та безпеку користувача в процесі знайомства.

1.2 Порівняльний аналіз аналогів

Розглянемо існуючі платформи для знайомств:

- Tinder;

- Badoo;

-Bumble.

Tinder [6] – це одна з найвідоміших мобільних платформ для знайомств, що з’явилася у 2012 році. Її основна особливість полягає у системі свайпання, коли користувач переглядає профілі інших учасників і проводить пальцем праворуч, якщо зацікавився, або ліворуч, якщо ні. Завдяки використанню геолокації додаток допомагає знаходити потенційних партнерів поблизу, що значно спрощує процес знайомств та робить його більш інтерактивним і

динамічним.

Популярність Tinder полягає не лише у простоті використання, а й у широкій аудиторії користувачів з усього світу. Система «match» дозволяє обом сторонам ініціювати спілкування після взаємного інтересу, що створює сприятливе середовище для нових знайомств. Додаткові функції, такі як «Super Like», допомагають виразити особливу зацікавленість, а постійні оновлення додатку гарантують сучасність та адаптацію до потреб користувачів.

Інтерфейс Tinder наведено на рисунку 1.1.

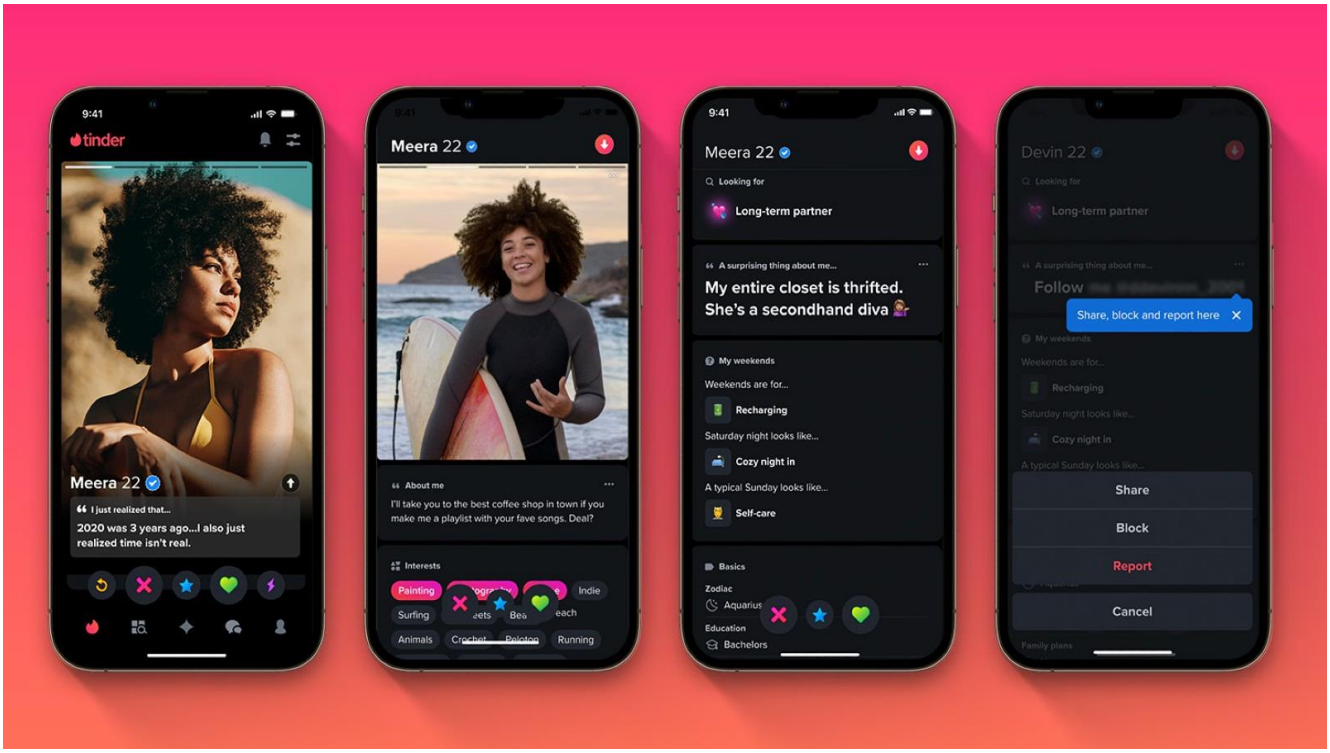

Рисунок 1.1 — Інтерфейс Tinder

Badoo [7] – ще один гравець на ринку онлайн-знайомств, який стартував ще у 2006 році. Ця платформа має глобальне охоплення та орієнтується як на випадкові зустрічі, так і на пошук серйозних стосунків. Завдяки різноманітним фільтрам та рекомендаціям користувачі можуть знаходити співрозмовників як на локальному рівні, так і серед представників інших країн. Особлива увага приділяється безпеці: система верифікації профілів сприяє зниженню ризиків та підвищенню довіри до спільноти.

Функціонал Badoo охоплює не лише традиційний обмін повідомленнями, а й можливість публікувати фотографії, брати участь у спільних обговореннях та ділитися враженнями. Це створює атмосферу соціальної мережі, де кожен користувач може відчути себе частиною великої спільноти, знайти однодумців чи навіть нових друзів. Завдяки постійному розвитку та впровадженню нових можливостей, Badoo залишається актуальним та цікавим для широкої аудиторії.

Інтерфейс Badoo наведено на рисунку 1.2.

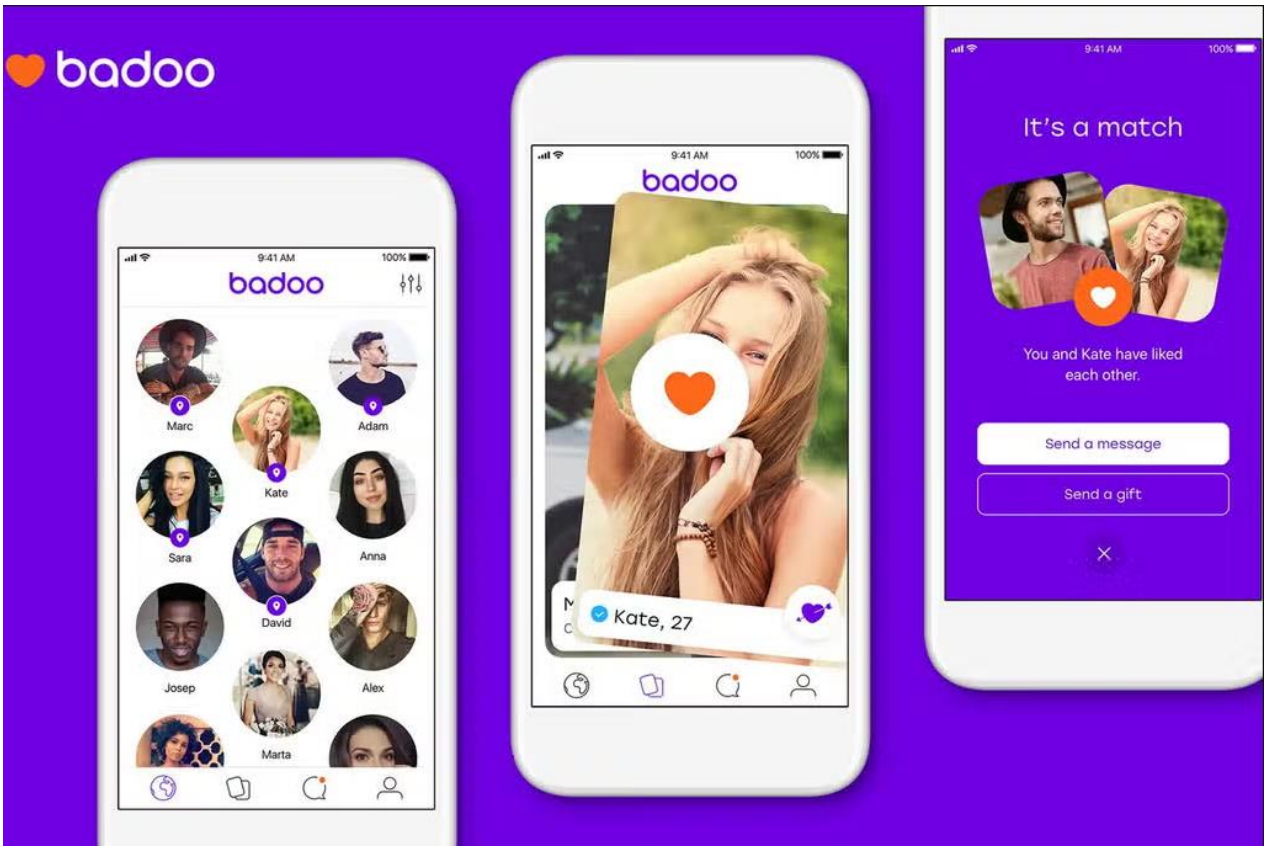

Рисунок 1.2 — Інтерфейс Badoo

Bumble [8] вирізняється серед інших додатків для знайомств своєю унікальною концепцією: тут саме жінки роблять перший крок. Заснований у 2014 році колишньою співзасновницею Tinder, Bumble спрямований на створення безпечнішого та комфортнішого середовища для жінок, дозволяючи їм контролювати процес початку спілкування. Цей підхід допоміг додатку здобути популярність серед тих, хто шукає більш прогресивний і сучасний

формат знайомств.

Окрім традиційного пошуку партнерів, Bumble пропонує розширені функції, такі як Bumble BFF для знаходження друзів і Bumble Bizz для налагодження професійних контактів. Це робить платформу багатофункціональною і привабливою для різних категорій користувачів. Завдяки акценту на безпеку, підтримку рівноправності та зручний інтерфейс, Bumble продовжує розширювати свою аудиторію і впливати на сучасну культуру знайомств.

Інтерфейс Bumble наведено на рисунку 1.3.

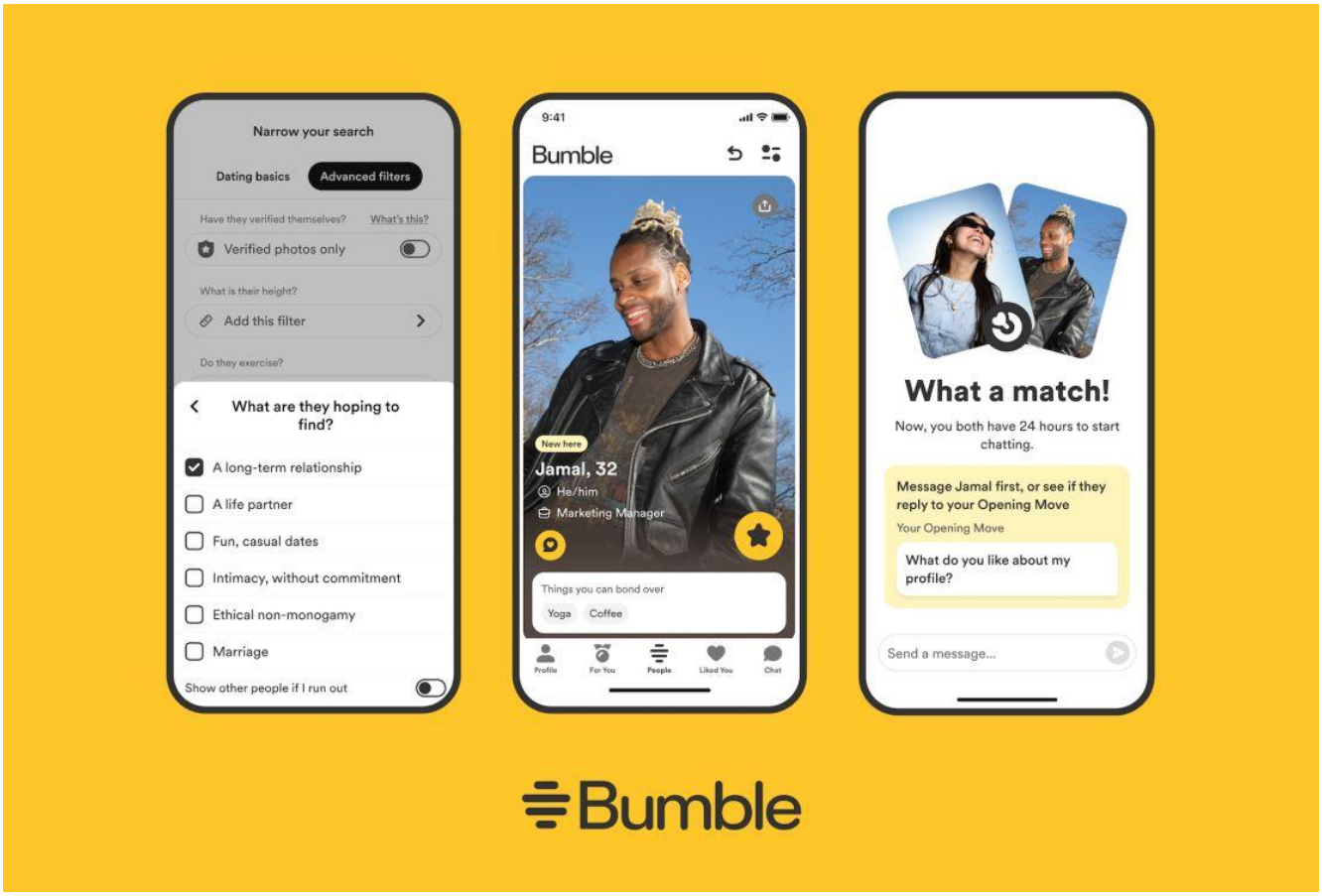

Рисунок 1.3 — Інтерфейс Bumble

Для демонстрації відмінностей між розглянутими застосунками для знайомств, та власною розробкою, було сформовано порівняльну таблицю (таблиця 1.1).

Таблиця 1.1 — Порівняльний аналіз аналогів

Характеристика	Tinder	Badoo	Bumble	Власна розробка
Свайпи	+	+	+	+
Показ активності користувача	+	+	-	+
Система взаємних лайків	+	+	+	+
Можливість відеодзвінків	+	+	+	+
Верифікація	+	+	+	+
Репутація користувача	-	-	-	+
Вказання точок для зустрічі	-	-	-	+
Загальний бал	5	5	4	7

Таким чином, актуальною є власна розробка, оскільки вона має вищий сумарний бал за аналоги.

1.3 Аналіз методів розв’язання задачі

Необхідно створити вебсистему, яка дозволяє людям знайомитися, спілкуватися та взаємодіяти через інтернет з використанням сучасних технологій, зокрема Angular.

Основні функціональні вимоги до системи:

1. Реєстрація та авторизація користувачів.
2. Створення та управління профілями користувачів.
3. Пошук користувачів за параметрами (вік, інтереси, локація тощо).
4. Встановлення контактів (лайки, взаємні симпатії).
5. Чат або система повідомлень.
6. Можливість завантаження медіафайлів (фото/відео).

Для аналізу розглянемо три основні підходи до реалізації цієї вебсистеми.

Перший метод передбачає створення вебсистеми для знайомств у вигляді

односторінкового додатку (Single Page Application) на Angular із використанням REST API для взаємодії між клієнтською частиною та сервером [9]. При цьому фронтенд повністю будується на Angular, який керує маршрутизацією, станом, формами та обробкою HTTP-запитів через модуль HttpClient. Back-end може бути створений за допомогою технологій на кшталт Node.js (Express чи NestJS), Spring Boot або Django [10]. Цей метод відзначається відносно простою архітектурою, хорошою продуктивністю інтерфейсу та зручністю у подальшій підтримці. Недоліками є підвищені вимоги до безпеки API та можливо повільне початкове завантаження, пов'язане з необхідністю завантаження усіх фронтенд-ресурсів на старті.

Другий метод реалізації вебсистеми знайомств полягає у поєднанні Angular із технологією GraphQL [11]. В такому випадку фронтенд взаємодіє із сервером за допомогою спеціалізованого GraphQL API, що дозволяє клієнту запитувати саме ту інформацію, яка потрібна у конкретній ситуації, мінімізуючи передачу зайвих даних та кількість запитів. В ролі клієнтського фреймворку найчастіше застосовується Apollo Angular, а на бекенді—Apollo Server, Hasura чи інші сервери з GraphQL-підтримкою. Серед переваг цього підходу—гнучкість роботи з даними, висока швидкість передачі інформації та зменшення навантаження на мережу. Серед недоліків—необхідність початкового налаштування більш складної архітектури та певна складність в управлінні кешуванням і станом на стороні клієнта.

Третій метод — це використання технології Angular Universal, яка дозволяє реалізувати серверний рендеринг (Server-Side Rendering, SSR) [12]. У цьому підході перша сторінка формується та рендериться на сервері, що забезпечує швидке початкове завантаження та кращі результати для SEO-просування системи знайомств у пошукових системах. Основними перевагами цього підходу є поліпшене індексування контенту пошуковими роботами та значно менший час завантаження першої сторінки. Водночас він має складнішу архітектуру реалізації, вимагає потужнішого сервера для підтримки навантаження, а також може виникати потреба у додаткових налаштуваннях для сумісності деяких бібліотек

Angular з технологією SSR.

1.4 Постановка задач дослідження

Провівши аналіз методів розв’язання поставленої задачі, аналіз стану застосунків для знайомства людей, порівняння аналогів було поставлено такі задачі дослідження:

- розробити архітектуру вебзастосунку для знайомства людей;
- розробити інтерфейс користувача застосунку;
- розробити метод формування репутації користувача;
- розробити метод вказання точок зустрічі;
- розробити алгоритми роботи застосунку;
- розробити функціонал застосунку;
- провести тестування розробленого застосунку.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз стану питання розробки вебсистем для знайомства людей, в результаті якого було доведено актуальність розробки такої системи. Розглянуто існуючі аналоги: Badoo, Tinder, Bumble, та проведено порівняльний аналіз. В результатів аналізу було продемонстровано переваги власної розробки, а саме: репутація користувача, вказання точок для зустрічі, через що власна розробка є актуальною. Проведено аналіз методів розв’язання задачі, проведено постановку задач дослідження.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Аналіз інформаційного забезпечення

Під час роботи системи, для забезпечення її функціоналу, збираються, використовуються та формуються такі дані:

1. Реєстраційні дані, які збираються при реєстрації користувача. До них належить:

- Ім’я, прізвище (необов’язково).

- Електронна адреса (обов'язково, необхідно для реєстраці та відновлення паролю).

- Пароль (необхідний для авторизованого доступу, при збереженні хешується та зберігається у вигляді хешу).

- Номер телефону (необов’язково, може використовуватися для додаткової верифікації).

- Дата народження (необхідно для перевірки вікових обмежень).

- Стать / гендерна ідентичність (для точності алгоритму підбору).

- Місцезнаходження (геолокація користувача, необхідне для роботи алгоритмів підбирання людей).

- Фото профілю / аватар (для персоналізації).

- Погодження з умовами конфіденційності та використання сервісу.

Також, при авторизації формуються унікальні JWT-токени, які необхідні для створення сесії користувача в системі. Вони є тимчасовими, через певний час необхідна повторна авторизація та створення нового токена.

2. Профіль користувача, редагування, інтереси, хобі:

- Основна інформація (короткий опис особистості, статус).

- Інтереси та хобі (список, теги, категорії).

- Фотографії та відеоматеріали (для профілю, галереї).

- Дані про освіту, професію, місце роботи (необов’язково, може використовуватись для уточнення підбору людей).

- Сімейний статус (необов’язково).

Також, по мірі заповнення профілю формуються метадані, в які входить дата створення та останнього оновлення профілю; рейтинг заповнення профілю, який являє собою відсоток заповненості для підвищення ефективності алгоритмів пошуку; активність профілю, у що входить кількість переглядів, кількість запитів на знайомство, взаємні лайки.

3. Меседжер (чат, листування):

- Текстові повідомлення (зберігаються на сервері, з шифруванням end-to-end).

- Зображення, відео, аудіоповідомлення (зберігаються на сервері).

- Таймштампи повідомлень (для хронології листування).

- Статус прочитання повідомлень (переглянуто, доставлено, набирає текст тощо).

- Список контактів, активні чати.

Також, формуються службові дані чату, такі як логування активності (вхід/вихід із чату), видалення повідомлень (таймштамп видалення), історія взаємодії між двома користувачами (лайки, блокування, скарги).

4. Репутація користувача:

- Оцінки інших користувачів (відгуки, рейтинги).

- Кількість скарг/позитивних відгуків.

- Бали репутації.

- Історія порушень та штрафів (наприклад, за неналежну поведінку).

5. Алгоритм підбору:

- Анкети користувачів (стать, вік, інтереси, хобі, цілі знайомств).

- Локації (географічне положення та близькість).

- Репутація користувачів (для більш якісного підбору).

- Історія взаємодій (перегляди, лайки, листування).

- Налаштування пошуку користувачів (переваги щодо віку, інтересів, місця проживання тощо).

В результаті обробки цих даних, формується список рекомендованих

профілів з коефіцієнтом відповідності, та відповідності за параметрами, які задав користувач (наприклад, категорії спільних інтересів).

6. Точки переміщення зустрічі:

- Точки на карті (координати GPS).

- Адреси/назви місць (кафе, ресторани, парки тощо).

- Час і дата запланованих зустрічей.

- Історія проведених зустрічей (необов’язково, може формуватися для статистики та репутації).

- Коментарі до зустрічі (відгуки, обговорення деталей).

7. Верифікація через соціальні мережі:

- Ідентифікатор користувача соцмережі (Facebook, Instagram, Google тощо).

- Дані із соцмереж (ім'я, аватар, контакти, друзі).

- Статус підтвердження особи через соцмережі (валідність акаунту, дата перевірки).

- Список підключених соцмереж (для відображення у профілі).

- Додаткові службові дані (адміністративні, аналітичні):

Крім цього, записуються логи доступу, помилок та продуктивності застосунку для аналітики роботи системи, її працездатності, стійкості до відмов та помилок.

Для покращення роботи системи також збираються дані аналітики користувацької поведінки (кількість активних користувачів, тривалість сесій, конверсії тощо).

Отже, для забезпечення функціоналу вебсистеми для знайомства людей збираються, генеруються та обробляються значні обсяги даних.

2.2 Розробка архітектури системи

Застосунок побудований за багатошаровою архітектурою, у якій кожен шар має чітко окреслені обов’язки та мінімальні залежності від інших.

На рівні інтерфейсу працює Angular-SPA: у межах одного застосунку

ізолюються окремі бізнес-домени (керування анкетами, обмін повідомленнями, модуль адміністрування). Усі маршрути, Guard’и та Resolver’и знаходяться у feature-модулях, а компоненти ніколи не звертаються до мережі безпосередньо. Усю взаємодію з бекендом інкапсулюють сервіс-класи, що повертають RxJS-потоки — вони реалізують авторизацію через JWT-токен, оновлюють статуси присутності й підписуються на push-повідомлення SignalR. Завдяки такому підходу інтерфейс залишається тонким і легко тестується окремо від серверної частини.

З боку сервера фронтенд спілкується з ASP.NET Core API, де кожен ресурс відображено окремим контролером: користувачі, лайки й повідомлення мають свої REST-ендпоінти, які повертають DTO-об’єкти у форматі JSON. Для реального часу задіяно два SignalR-хаби: один відповідає за онлайн-статуси, другий — за миттєву доставку приватних повідомлень.

Бізнес-правила зосереджені в application-шарі. Саме тут знаходяться сервіси для генерації й підпису JWT, завантаження й трансформації зображень через Cloudinary, а також клас UnitOfWork, який об’єднує кілька репозиторіїв в одну атомарну транзакцію. Репозиторії приховують від решти коду деталі EF Core й працюють лише з доменними сутностями або їхніми проєкціями за допомогою AutoMapper. Така інкапсуляція дозволяє, наприклад, виконати весь бізнес-процес створення повідомлення, оновлення статистики лайків та надсилання нотифікації в межах одного SaveChangesAsync, гарантуючи узгодженість даних.

Доменний шар представлений агрегатами: AppUser, Photo, Message і UserLike. Усі вони містять власну поведінку (методи, що розраховують вік, визначають взаємність лайків тощо) і не знають нічого про те, як саме їх буде серіалізовано або збережено.

За фізичне зберігання даних відповідає infrastructure-шар. Контекст бази даних конфігурується для PostgreSQL у продакшні й для SQLite під час локальної розробки. Додаткові інтеграції — Cloudinary для фото, Fly.io для розгортання container-образу — обгорнуті власними абстракціями, тому заміну

зовнішнього сервісу можна виконати, не торкаючись бізнес-логіки. У production секрети передаються через Fly.io Secrets, а Docker-pipeline GitHub Actions автоматично збирає multi-stage-image та виконує деплой.

Архітектура застосунку наведена на рисунку 2.1.

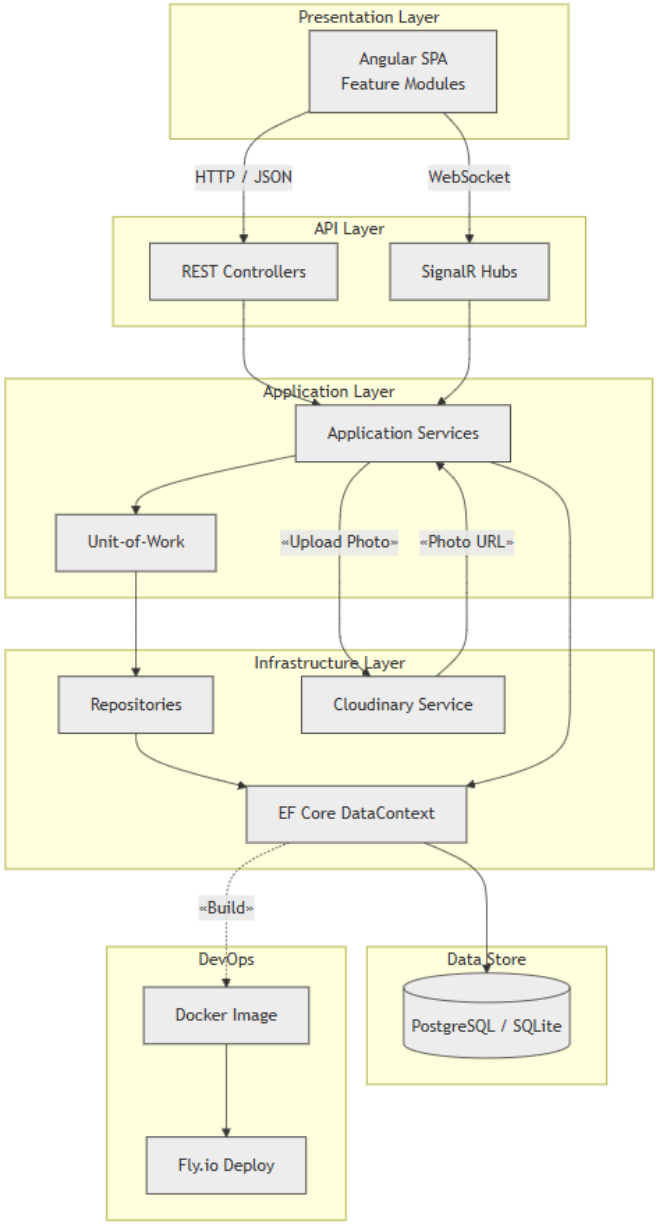

Рисунок 2.1 — Архітектура застосунку

Крос-катингові задачі вирішуються централізовано. JWT-аутентифікація реалізована симетричним ключем і ролями в claim’ах, які Angular перехоплює через HTTP-Interceptor. CORS-політика дозволяє лише один origin, указаний у

конфігурації середовища. Початкове наповнення БД забезпечує Seeder, що створює тестових користувачів, фото й повідомлення — це скорочує час до «першого кліку» під час локального запуску. Окремий «Buggy»-контролер ілюструє типові помилки 400 / 500 і допомагає фронтенду уніфікувати обробку винятків.

У підсумку така архітектура дає три переваги. По-перше, низьке зчеплення й висока зв’язність: Angular знає тільки HTTP-контракт, а бізнес-код лише абстракції репозиторіїв та сервісів. По-друге, тестованість: бізнес-правила можна покривати unit-тестами без реальної БД чи HTTP-перекликів. По-третє, масштабованість: будь-яка зміна, від переходу на іншу систему збереження файлів до заміни СУБД, вимагає мінімальних правок і не зачіпає інші рівні.

2.3 Розробка методу формування репутації користувача

Метод формування репутації користувача є необхідним в системі для знайомства людей з таких причин:

1. Підвищує безпеку користувачів шляхом зменшення ризику недобровісного використання системи шахраями. Вони використовують платформи для знайомств, створюючи фейкові профілі для завоювання довіри інших користувачів, яку використовують для збору банківських даних з метою викрадення грошей, вимагання грошей різного роду маніпуляціями, організацією підставних зустрічей. Модерація не завжди є ефективною у протидії таким діям, тому формування репутації користувачами є додатковим способом пересвідчитися, що профіль справжній та користувач добросовісно використовує систему.

2. Стимулювання позитивної поведінки, тобто заохочування користувачів до конструктивних взаємодій, оскільки вони підвищують репутацію користувача, що може давати свої переваги при використанні системи, наприклад, більшу довіру з боку інших користувачів.

3. Алгоритм підбору використовує репутаційні дані для сортування та фільтрації профілів, забезпечуючи високу якість взаємодії між користувачами.

4. Система репутації дозволяє оперативно реагувати на скарги або негативні дії, забезпечуючи належний рівень модерації та можливість апеляції для користувачів.

Блок-схема алгоритму роботи цього методу наведена на рисунку 2.2.

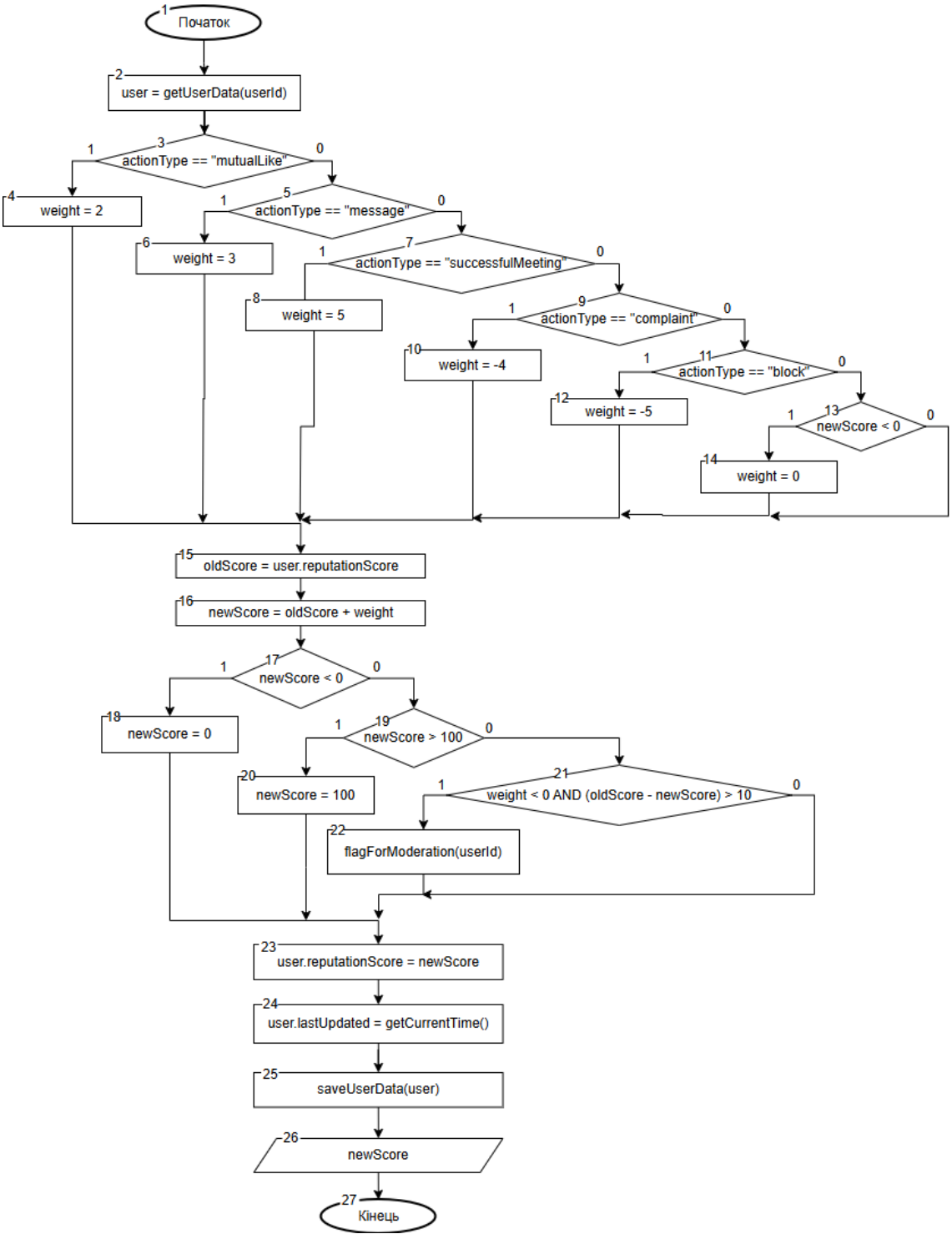

Рисунок 2.2 — Блок-схема алгоритму методу формування репутації користувача

Метод формування репутації користувача складається з таких етапів:

1. Збір початкових даних. Під час реєстрації збираються базові дані (ім'я, вік, місцезнаходження, фото, верифікація через соцмережі). Це створює «базову довіру», що може бути підвищено після успішного проходження верифікації.

Відстежуються дії користувача, такі як відправка повідомлень, лайки, відповіді, перегляди профілів, а також дії негативного характеру: скарги, блокування, відхилення спаму.

2. Визначення ключових показників (KPI). Сюди входять позитивні взаємодії, такі як: кількість лайків, позитивних відгуків, взаємні повідомлення (чат, відповідні дії), завершені успішні зустрічі (якщо є можливість зворотного зв’язку). Також входять сюди і негативні взаємодії: кількість скарг, повідомлень про небажану поведінку, блокування іншими користувачами, невдале проходження верифікації чи фейкові дані.

3. Присвоєння вагових коефіцієнтів.

Ваги для позитивних дій: кожен взаємний лайк додає 2 бали, повідомлення – 3 бали, успішно завершена зустріч – 5 балів.

Ваги для негативних дій: кожна скарга віднімає 4 бали, блокування – 5 балів.

Верифікований користувач може отримувати додаткові бонусні бали, що посилює довіру до його профілю.

4. Розрахунок репутаційного балу. Проводиться підрахунок сумарних позитивних та негативних балів за певний період часу.

Для отримання зручного діапазону (наприклад, від 0 до 100) застосовують нормалізаційні формули, які дозволяють порівнювати користувачів між собою.

Оцінка може включати не лише сумарні бали, а й динаміку змін - чи покращується поведінка користувача з часом, чи відзначається різке зниження активності через негативні дії.

5. Оновлення репутації в режимі реального часу. При кожній новій взаємодії (як позитивній, так і негативній) виконується оновлення репутаційного балу.

Система може періодично проводити аналіз (щоденно або щотижнево) для коригування репутації з урахуванням змін у поведінці.

6. Використання алгоритму модерації. Алгоритми автоматично виявляють підозрілу або негативну поведінку (наприклад, раптове зниження репутації через збільшення скарг) і можуть тимчасово обмежувати функціонал користувача або відправляти дані на ручну перевірку модераторам.

Адміністратори або модератори можуть вручну коригувати репутацію на підставі додаткової інформації (наприклад, після розгляду скарг або при виявленні фейкових профілів).

7. Інтеграція репутації у функціонал системи. Алгоритм підбору використовує репутаційний бал для сортування профілів: користувачі з вищою репутацією можуть отримувати більше показів і пріоритет у списку рекомендацій.

Система надає бонуси чи спеціальні можливості користувачам із стабільно високим рейтингом, стимулюючи підтримувати позитивну поведінку.

8. Зворотний зв’язок та апеляції. Користувачі мають можливість оскаржити негативні оцінки або попросити перегляд репутації у разі помилок.

Хоча деталі алгоритму можуть бути частково прихованими для запобігання зловживань, користувачам можуть надаватися загальні рекомендації щодо підвищення свого рейтингу.

2.4 Розробка методу вказання точок зустрічі

Метод вказання точок переміщення зустрічі необхідний для організації та оптимізації процесу узгодження місця зустрічі між користувачами. Його призначення полягає у:

1. Зручності планування, яка дозволяє інтерактивно обирати та уточнювати місце зустрічі за допомогою карт, що спрощує процес організації.

2. Підвищенні безпеки шляхом використання перевірених геоданих та інтеграції з картографічними сервісами, що дозволяє обирати безпечні та доступні локації.

3. Адаптивності до змін, яка дозволяє в режимі реального часу змінювати точку зустрічі.

4. Інтеграції з навігаційними інструментами, що дозволяє створювати маршрути до місця зустрічі.

Метод складається з таких етапів:

1. Збір початкових даних.

Система запитує дозвіл на використання геолокаційних даних користувача (через браузер чи мобільний пристрій) для визначення поточної позиції. Користувач може ввести адресу або вибрати місто/район, де він/вона бажає зустрітися.

2. Інтеграція з картографічним сервісом - інтеграція з Google Maps для відображення карти безпосередньо в додатку. Інтерактивна карта дозволяє користувачу переглядати доступну територію, бачити дороги, визначні місця та інші об’єкти.

3. Інтерактивне позначення точки зустрічі. Користувач може натискати на карту або перетягувати маркер для позначення конкретної точки зустрічі. Додатково доступна функція пошуку, що дозволяє знайти потрібну адресу або місце (наприклад, кафе, парк, заклад тощо).

4. Додавання додаткової інформації - можливість додати текстовий опис, що включає особливості місця, наприклад, "затишне кафе біля метро" або "парк з гарним видом на місто". Додаткові дані, як от фотографії або короткі відгуки, можуть допомогти іншій стороні зробити вибір, особливо якщо локація незнайома.

5. Валідація та збереження даних. Система перевіряє правильність координат та відповідність адреси за допомогою картографічного сервісу. Вибрана точка зустрічі зберігається у базі даних, пов’язана з конкретною зустріччю або подією у месенджері.

6. Повідомлення та підтвердження вибору. Після збереження локації система надсилає повідомлення іншому користувачу (через меседжер або push-сповіщення) з інформацією про вибрану точку зустрічі. Користувач може

переглянути запропоновану локацію, внести свої коригування або підтвердити вибір.

Блок-схема алгоритму роботи цього методу наведена на рисунку 2.3.

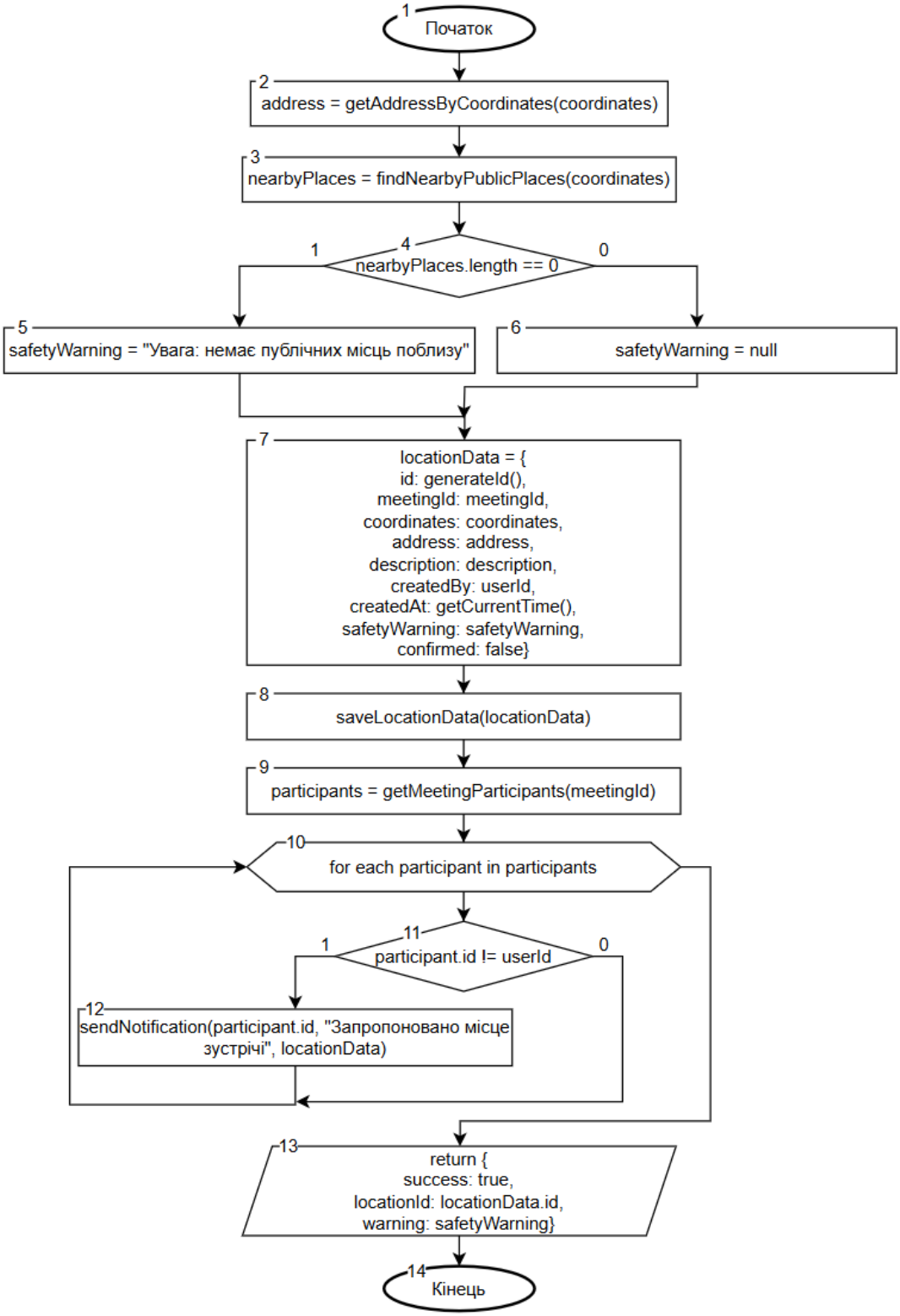

Рисунок 2.3 — Блок-схема роботи методу вказання точок зустрічі

У разі змін (зміна локації, часу зустрічі) дані миттєво оновлюються у системі, а обидва учасники отримують відповідні сповіщення. Система може надати посилання або інтегровану навігацію, що допоможе користувачам дістатися до обраної точки (генерація маршруту від поточної локації до точки зустрічі). Користувачі можуть змінювати точку зустрічі до остаточного підтвердження, що дозволяє гнучко реагувати на зміну обставин.

Збережені дані можуть використовуватися для статистики та аналізу майбутніх рекомендацій (наприклад, популярні місця зустрічей). Вбудовані механізми дозволяють перевірити, чи не обирається локація, що може бути потенційно небезпечною або недоступною для однієї зі сторін.

2.5 Розробка діаграм та алгоритмів роботи системи

Було розроблено діаграму класів, які відповідають за формування даних у об’єкти для їх подальшої обробки та зберігання у базі даних. Діаграму класів наведено на рисунку 2.4.

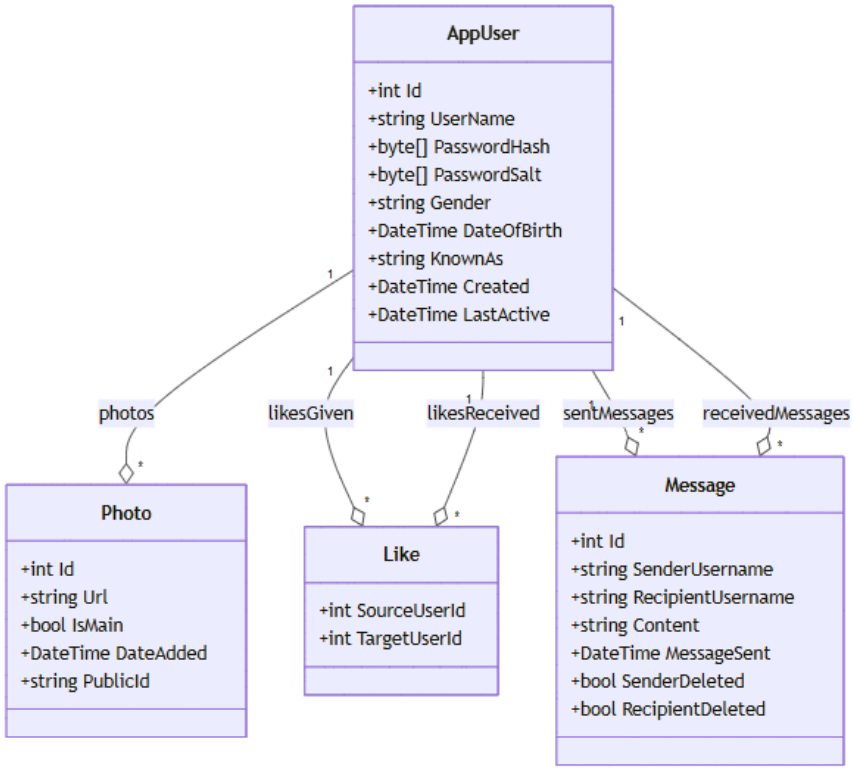

Рисунок 2.4 — Діаграма класів

AppUser є центральною сутністю, що відповідає за зберігання даних користувачів та їхніх профілів. Він містить атрибути для аутентифікації, базову інформацію про користувача та навігаційні властивості для зв’язків із фотографіями, лайками та повідомленнями.

Поля AppUser:

- Id: унікальний ідентифікатор (int).

- UserName: логін користувача (string).

- PasswordHash, PasswordSalt: хеш та сіль для збереження пароля (byte[]).

- Gender: стать користувача (string).

- DateOfBirth: дата народження (DateTime).

- KnownAs: ім’я/псевдонім для відображення (string).

- Created, LastActive: дати створення акаунту та останньої активності (DateTime).

- Photos: колекція об’єктів Photo.

- LikedUsers, LikedByUsers: колекції об’єктів UserLike (лайки, що поставив і отримав).

- MessagesSent, MessagesReceived: колекції об’єктів Message.

Клас Photo відповідає за представлення фотографії користувача, включаючи URL, статус основного фото та метадані. Використовується для відображення й керування фото в профілі.

Поля класу Photo:

- Id: унікальний ідентифікатор (int).

- Url: посилання на зображення (string).

- IsMain: прапорець основного фото (bool).

- DateAdded: дата завантаження (DateTime).

- PublicId: ідентифікатор у зовнішньому сервісі (string).

Об’єкт Like (UserLike) моделює відношення «користувач лайкнув іншого користувача». Використовується для реалізації механіки лайків та матчу.

Поля Like:

- SourceUserId: ідентифікатор автора лайку (int).

- TargetUserId: ідентифікатор цільового користувача (int).

- SourceUser, TargetUser: навігаційні властивості до AppUser.

Клас Message відповідає за текстові повідомлення між користувачами. Містить вміст повідомлення, часові мітки та статус видалення.

Поля класу Message:

- Id: унікальний ідентифікатор (int).

- SenderUsername, RecipientUsername: логіни відправника та отримувача (string).

- Content: текст повідомлення (string).

- MessageSent: дата/час відправлення (DateTime).

- SenderDeleted, RecipientDeleted: прапорці видалення (bool). Sender, Recipient: навігаційні властивості до AppUser.

Розроблено діаграму станів, яка моделює життєвий цикл сесії в інтерфейсі користувача.

Діаграма станів наведена на рисунку 2.5.

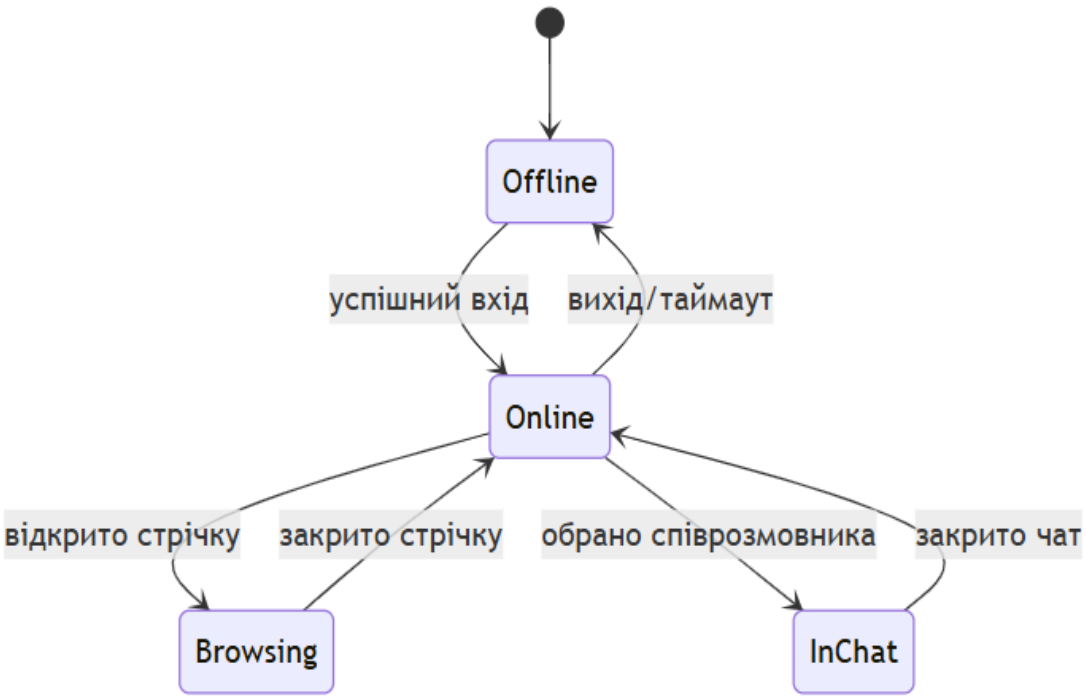

Рисунок 2.5 — Діаграма станів циклу сесії

Сюди входять такі стани:

- Offline (початковий стан);
- Online після успішного логіну;
- Browsing під час перегляду профілів;
- InChat у контексті відкритого чату;
- повернення Online при закритті чатів чи стрічки;
- зворотний перехід у Offline при виході або таймауті.

Діаграма послідовності реєстрації користувача відображає послідовність процесу, у якому клієнт (браузер чи мобільний додаток) відправляє на AccountController запит POST /register із об'єктом registerDto. Контролер викликає сервіс токенів (TokenService.CreateToken), який перевіряє облікові дані та формує JWT. Після отримання токена контролер повертає клієнту HTTP 201 зі структурою UserDto, що містить ім'я користувача, токен та базові профільні дані. Така послідовність гарантує, що користувач відразу отримує дійсний токен для наступних запитів.

Діаграма послідовності реєстрації користувача наведена на рисунку 2.6.

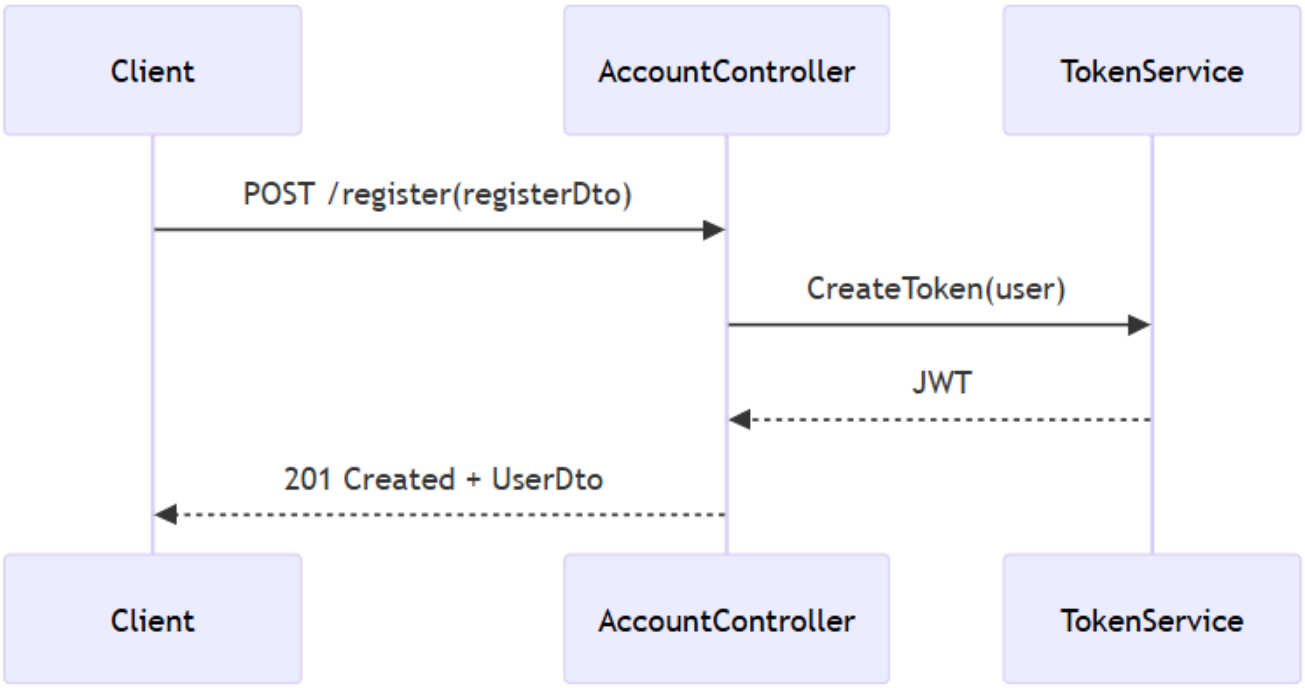

Рисунок 2.6 — Діаграма послідовності реєстрації користувача

Діаграма послідовності входу користувача в систему передбачає надсилання POST /login із loginDto до того ж контролера AccountController. Контролер зчитує користувача з бази з його фотографіями, перевіряє пароль через UserManager.CheckPasswordAsync, а потім генерує JWT через TokenService. У разі успіху повертається HTTP 200 з UserDto, який клієнт зберігає локально (LocalStorage) і використовує для авторизації всіх подальших викликів API. Діаграма послідовності входу користувача в систему наведена на рисунку 2.7.

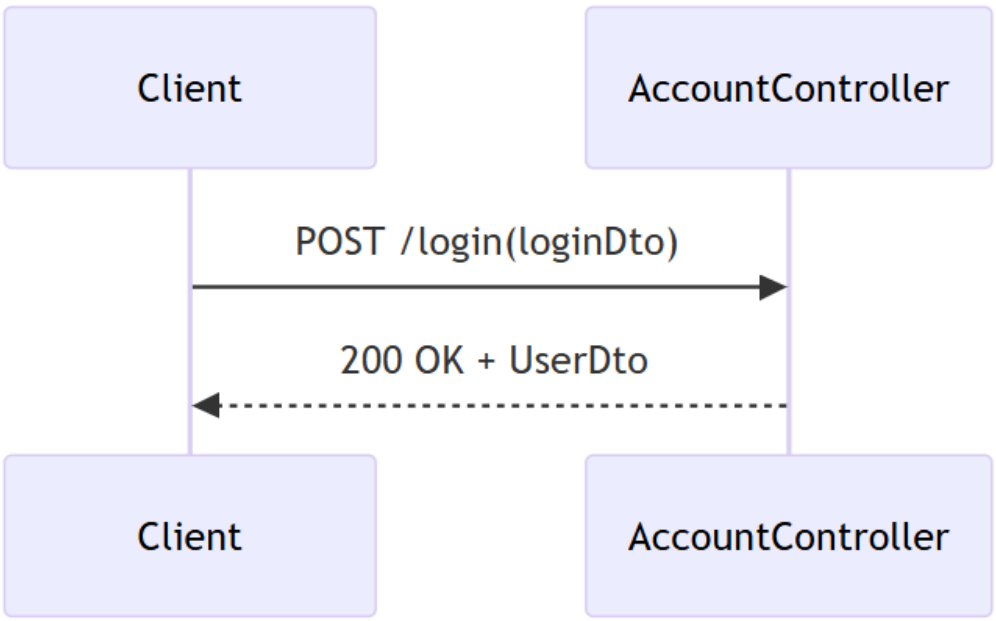

Рисунок 2.7 — Діаграма послідовності входу користувача в систему

У діаграма послідовності отримання списку профілів клієнт виконує GET /users?params до UsersController, передаючи параметри фільтрації й пагінації. Контролер звертається до IUnitOfWork.UserRepository.GetMembersAsync, отримує PagedList<MemberDto> із бази даних та додає в HTTP-заголовки інформацію про сторінки. У відповідь клієнт отримує список DTO користувачів та використовує його для рендерингу стрічки профілів. Діаграма послідовності отримання списку профілів наведена на рисунку 2.8.

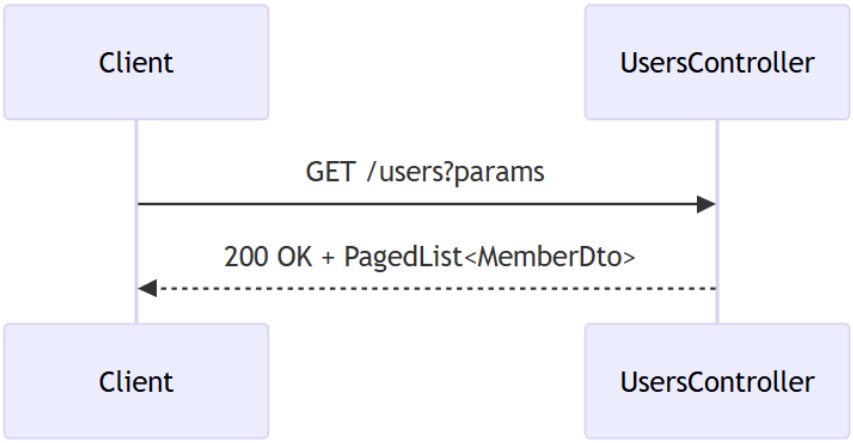

Рисунок 2.8 — Діаграма послідовності отримання списку профілів

Діаграма послідовності лайків. Коли користувач натискає кнопку «лайк», клієнт робить POST /likes/{username} до LikesController. Контролер перевіряє існування цільового користувача, викликає LikesRepository.AddLike() та зберігає відношення у БД. Якщо взаємний лайк вже існує, контролер через NotificationService генерує нотифікацію про матч. Нарешті повертається HTTP 200 (лише лайк) або HTTP 201 (матч, відкриття чату).

Діаграма послідовності лайків наведена на рисунку 2.9.

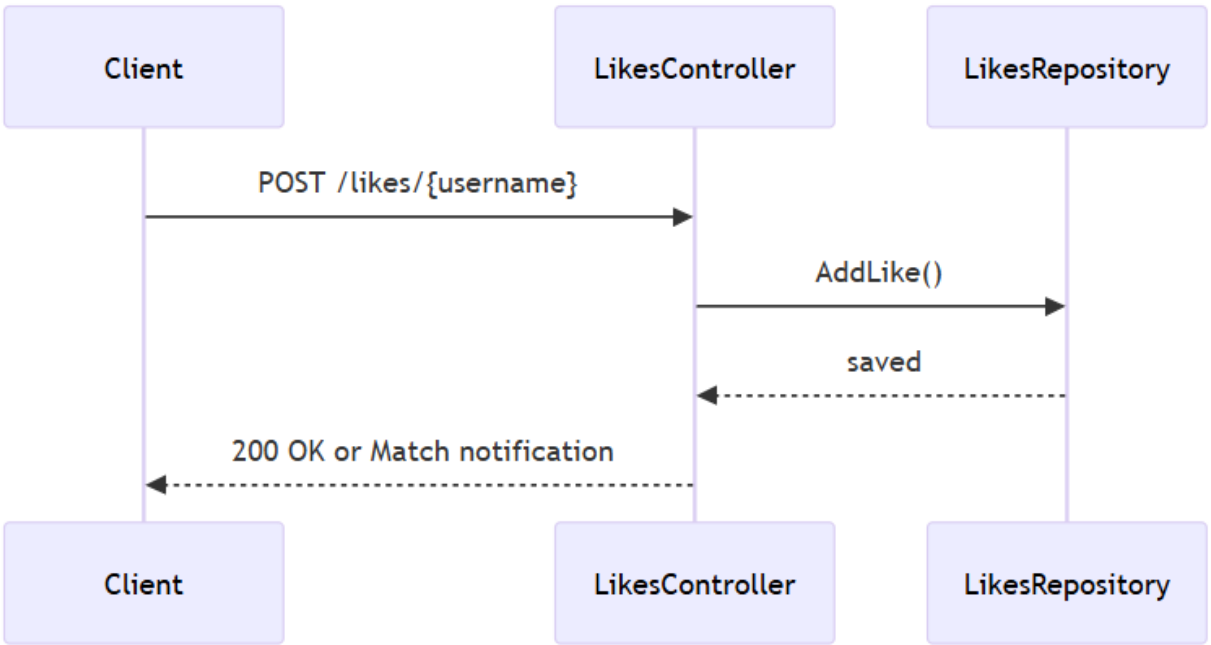

Рисунок 2.9 — Діаграма послідовності лайків

Під час процесу відправлення повідомлення через SignalR клієнт викликає метод Hub SendMessage(dto) на MessageHub. Hub перевіряє, що відправник і отримувач — різні користувачі, завантажує їх із репозиторію, створює сутність Message, додає її до MessageRepository та викликає Complete(). Після успішного збереження Hub перетворює повідомлення в MessageDto і розсилає його групі SignalR, щоб одержувач у реальному часі побачив нову повідомлення.

Діаграма послідовності процесу відправлення подовілмення через SignalR наведена на рисунку 2.10.

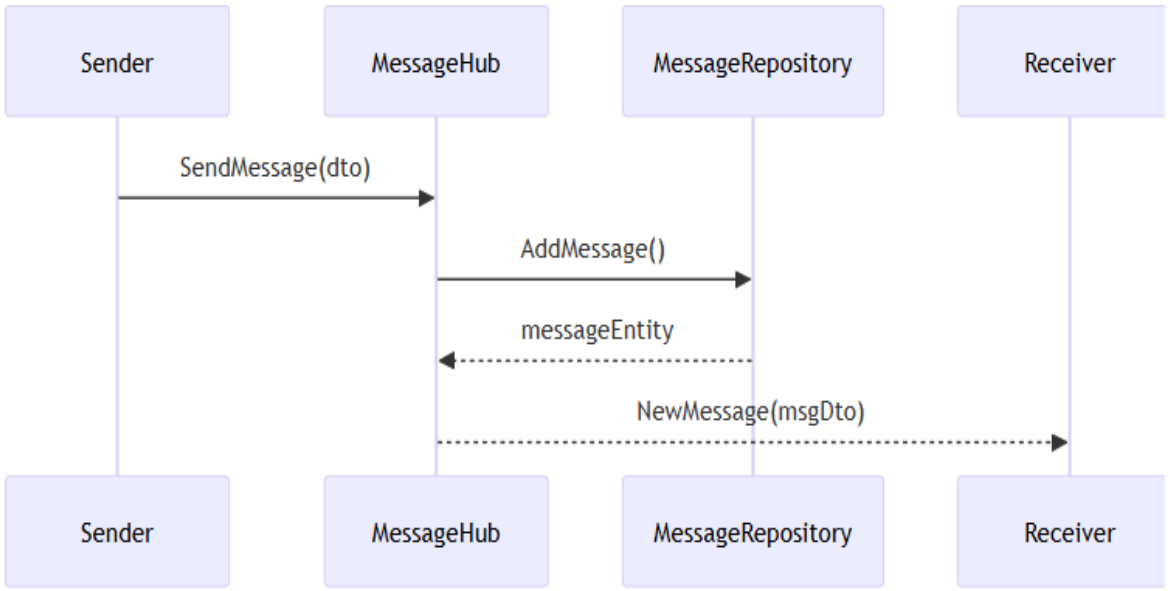

Рисунок 2.10 — Діаграма послідовності процесу відправлення подовілмення через SignalR

Під час процесу додавання фото клієнт надсилає POST /users/add-photo із файлом у FormData до UsersController. Контролер передає файл у PhotoService.AddPhotoAsync, який завантажує його в хмару (Cloudinary) й повертає uploadResult. Після перевірки помилок контролер створює сутність Photo, додає її до користувача і викликає Complete(). У випадку успіху повертається HTTP 201 з PhotoDto, а UI одразу відображає нову фотографію.

Діаграма процесу додавання фото наведена на рисунку 2.11.

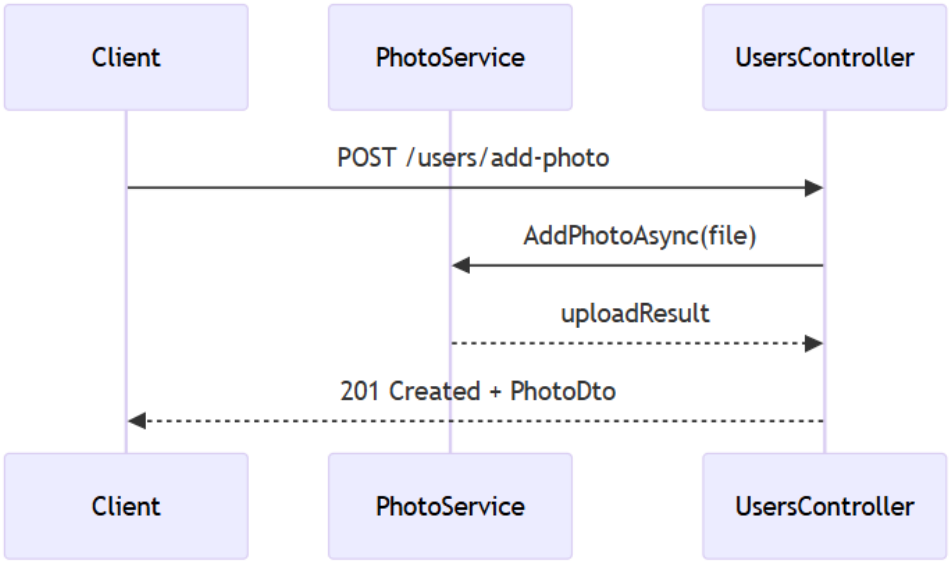

Рисунок 2.11 — Діаграма послідовності процесу додавання фото

Діаграма послідовності підключення до чату (OnConnectedAsync) передбачає встановлення SignalR-з'єднання з MessageHub і передачу параметру user у рядок запиту. У методі OnConnectedAsync() Hub дістає імена двох учасників, викликає AddToMessageGroup(), щоб зареєструвати їх у групі, і потім розсилає всім учасникам групи повідомлення UpdatedGroup із деталями поточного складу чату. Це забезпечує синхронізацію клієнтів у момент підключення до чату.

Діаграма послідовності підключення до чату наведена на рисунку 2.12.

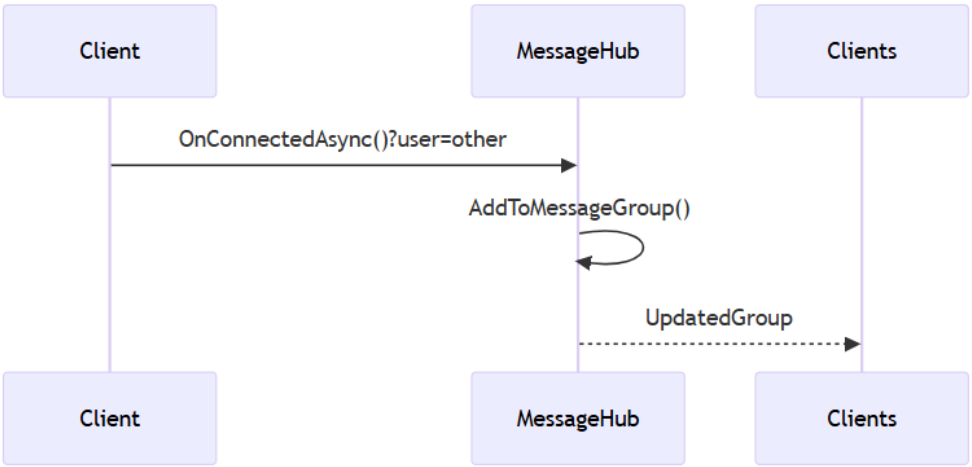

Рисунок 2.12 — Діаграма послідовності підключення до чату

Загальний алгоритм роботи системи описує усі можливі взаємодії користувача з системою.

Блок-схема загального алгоритму роботи системи наведена на рисунку 2.13.

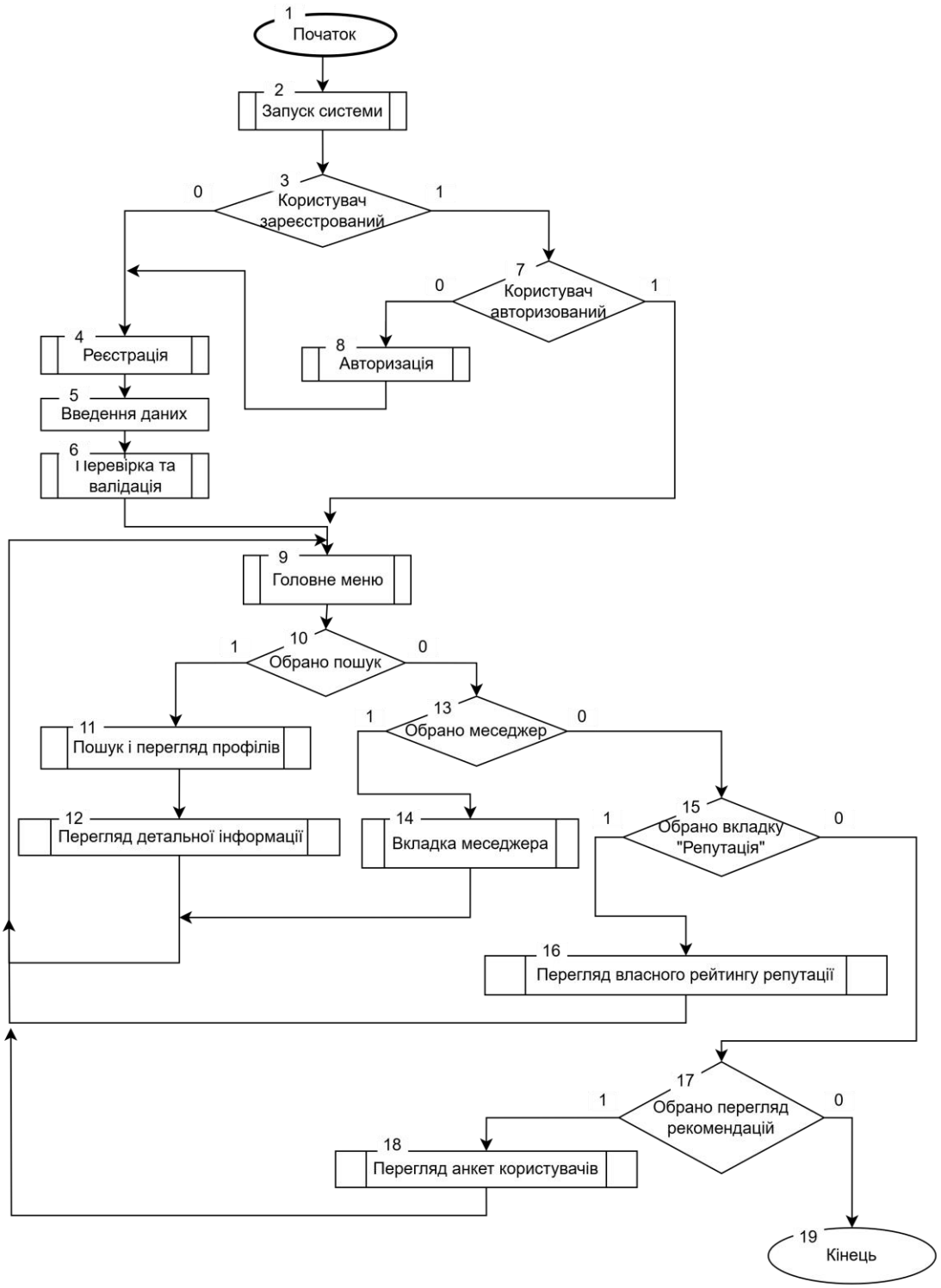

Рисунок 2.13 — Блок-схема загального алгоритму роботи системи

Після її запуску, відбувається перевірка, чи користувач зареєстрований у системі. У разі, якщо ні, відбувається перехід на сторінку реєстрації, де потрібно ввести свої дані. Якщо користувач зареєстрований, відбувається процес авторизації.

Після авторизації, користувачу доступне головне меню, в якому він залежно від свого вибору, може переглядати анкети користувачів, свій рівень репутації, користуватися меседжером, організувати зустріч з іншим користувачем, прив’язати свої сторінки в соцмережах.

Таким чином було розроблено діаграми послідовності, станів, та загальний алгоритм системи для знайомств. Розроблені алгоритми демонструють принцип роботи основного функціоналу системи.

2.5 Висновки

У другому розділі, було проведено аналіз інформаційного забезпечення системи, описано, які дані використовуються та генеруються при роботі системи. Розроблено архітектуру системи. Розроблено метод формування репутації користувача, метод вказання точок зустрічі. Побудовано загальний алгоритм роботи системи, продемонстровано його у вигляді відповідної блок-схеми, на якій відображено увесь процес роботи системи.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз та вибір мови програмування розробки

Для вибору мови програмування розробки вебсистеми для знайомства людей проведемо порівняння таких мов, як: JavaScript, C#, Kotlin.

JavaScript [13] — найпопулярніша мова веброзробки, що запускається безпосередньо в браузері й дозволяє створювати інтерактивні клієнтські інтерфейси. Сучасні фронтенд-фреймворки на JavaScript (React, Vue, Angular) забезпечують компонентний підхід, декларативний рендеринг і ефективне оновлення DOM, що дуже зручно для реалізації динамічної стрічки профілів, лайків та чатів у реальному часі. На сервері JavaScript в середовищі Node.js дає можливість використовувати одну й ту саму мову для бекенду й фронтенду, спрощуючи управління залежностями та логікою обох частин застосунку.

C# [14] у поєднанні з платформою .NET Core (або .NET 6/7) є потужним вибором для створення надійного бекенду зі строгим типізуванням і багатим набором стандартних бібліотек. Завдяки Entity Framework Core легко реалізувати шар доступу до даних із підтримкою міграцій, LINQ-запитів та відстеження змін, а SignalR дозволяє оперативно обробляти події реального часу (чат, повідомлення про матч). Крім того, ASP.NET Core має гнучку систему middleware, вбудовану підтримку аутентифікації/авторизації та широкі можливості для масштабування в хмарі.

Kotlin [15] — сучасна мова для розробки Android-додатків і мультиплатформених рішень (Kotlin Multiplatform Mobile). У клієнтській частині Android-версії застосунку Kotlin із Jetpack Compose забезпечує декларативне та лаконічне описання UI, а його повна сумісність із Java відкриває доступ до всього екосистеми Android. Kotlin Multiplatform дозволяє спільно використовувати бізнес-логіку (моделі, мережеві запити) між Android та iOS, що значно скорочує час розробки та підтримки мобільних клієнтів.

Сформуємо таблицю розглянутих мов програмування (таблиця 3.1).

Таблиця 3.1 — Порівняння мов програмування

Характеристика	JavaScript	C#	Kotlin
Вебінтерфейс на боці клієнта	+	–	–
API бекенду	+	+	+
Комунікація в режимі реального часу (WebSocket)	+	+	–
Мобільна версія	+	–	+
Спільний код (мультиплатформа)	+	–	+
Екосистема та бібліотеки веброзробки	+	+	+
Аутентифікація та авторизація (JWT, OAuth)	+	+	+
Пагінація та фільтрація списків	+	+	+
Сумарний бал	8	5	6

Таким чином, для розробки вебсистеми для знайомства людей було обрано мову програмування JavaScript, оскільки вона має вищий сумарний бал за C# та Kotlin.

3.2 Розробка бази даних

База даних дозволяє зберігати дані для потреб системи форматі, витягувати їх по запиту користувача, вносити зміни до наявних даних[16].

Таблиця User містить в собі дані автентифікації та базову інформацію профілю користувачів, таку як: ім’я користувача, посилання на фото профіля, стать, ролі в системі. Поля таблиці User наведені у таблиці 3.2.

Таблиця 3.2 — Поля таблиці User

Поле	**Тип**	**Опис**
username	рядок	Первинний ключ. Унікальний ідентифікатор кожного користувача.
token	рядок	Токен автентифікації для сесій користувача.
photoUrl	рядок	URL-посилання на фото профілю користувача.
knownAs	рядок	Відображуване ім'я або псевдонім користувача.
gender	рядок	Гендерна ідентичність користувача.

Таблиця Member містить розширену інформацію профілю користувачів:

вік, стать, активність, інтереси, інформацію про місто та країну, підтримуючи детальні профілі у додатку. Поля таблиці Member наведені у таблиці 3.3.

Таблиця 3.3 — Поля таблиці Member

Поле	Тип	Опис
id	ціле число	Первинний ключ. Унікальний ідентифікатор кожного профілю.
userName	рядок	Ім'я користувача, пов'язане з цим профілем.
photoUrl	рядок	URL основного зображення профілю.
age	ціле число	Вік користувача у роках.
knownAs	рядок	Відображуване ім'я або псевдонім.
created	рядок	Дата створення профілю.
lastActive	рядок	Часова мітка останньої активності користувача.
gender	рядок	Гендерна ідентичність користувача.
introduction	рядок	Самопредставлення або біографічний текст.
lookingFor	рядок	Опис того, що користувач шукає у зв'язках.
interests	рядок	Перелічені інтереси та хобі користувача.
city	рядок	Місто, де знаходиться користувач.
country	рядок	Країна, де знаходиться користувач.

Таблиця Photo керує фотографіями, пов'язаними з профілями користувачів, включаючи статус затвердження. Поля таблиці Photo наведені у таблиці 3.4.

Таблиця 3.4 — Поля таблиці Photo

Поле	Тип	Опис
id	ціле число	Первинний ключ. Унікальний ідентифікатор кожного фото.
url	рядок	URL-посилання на збережене фото.
isMain	логічний	Вказує, чи є це основним фото профілю користувача.
isApproval	логічний	Вказує, чи затверджено фото для відображення.
username	рядок	Ім'я користувача, власника фото.

Таблиця Message зберігає комунікації між користувачами, включаючи

метадані про статус повідомлень. Поля таблиці Message наведені у таблиці 3.5.

Таблиця 3.5 — Поля таблиці Message

Поле	Тип	Опис
id	ціле число	Первинний ключ. Унікальний ідентифікатор кожного повідомлення.
senderId	ціле число	ID-посилання на користувача, який надіслав повідомлення.
senderUsername	рядок	Ім'я користувача відправника.
senderPhotoUrl	рядок	URL фото профілю відправника.
recipientId	ціле число	ID-посилання на користувача, який отримує повідомлення.
recipientUsername	рядок	Ім'я користувача отримувача.
recipientPhotoUrl	рядок	URL фото профілю отримувача.
content	рядок	Текстовий вміст повідомлення.
dateRead	дата/час	Коли повідомлення було прочитане отримувачем.
messageSent	дата/час	Коли повідомлення було надіслане.

Таблиця Group визначає групи для комунікації в реальному часі або класифікації користувачів. Поля таблиці Group наведені у таблиці 3.6.

Таблиця 3.6 — Поля таблиці Group

Поле	Тип	Опис
name	рядок	Первинний ключ. Унікальний ідентифікатор кожної групи.

Таблиця Connection керує активними підключеннями користувачів до груп, ймовірно для комунікацій у реальному часі. Поля таблиці Connection наведені у таблиці 3.7.

Таблиця 3.7 — Поля таблиці Connection

Поле	Тип	Опис
connectionId	рядок	Первинний ключ. Унікальний ідентифікатор кожного підключення.
username	рядок	Ім'я користувача, який підключений.
groupName	рядок	Назва групи, до якої підключений користувач.

Таблиця Pagination підтримує пагінацію даних для ефективного завантаження та відображення інформації. Поля таблиці Pagination наведені у таблиці 3.8.

Таблиця 3.8 — Поля таблиці Pagination

Поле	Тип	Опис
currentPage	ціле число	Поточний номер сторінки, яка переглядається.
itemsPerPage	ціле число	Кількість елементів, що відображаються на сторінці.
totalItems	ціле число	Загальна кількість елементів на всіх сторінках.
totalPages	ціле число	Загальна кількість доступних сторінок.

Схема бази даних наведена на рисунку 3.1.

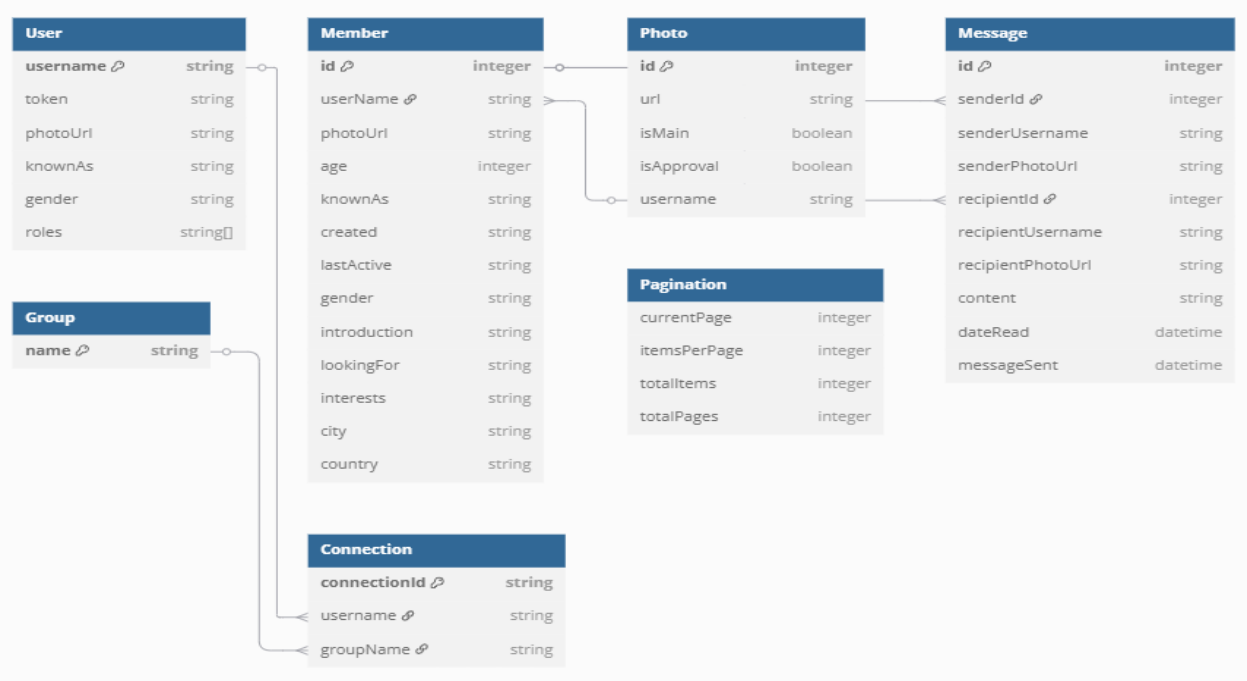

Рисунок 3.1 — Схема бази даних

Взаємозв'язки між таблицями:

•Профілі користувачів пов'язані з їхніми надісланими та отриманими повідомленнями.

•Фотографії пов'язані з конкретними іменами користувачів.

•Підключення пов'язують користувачів з групами, до яких вони приєдналися.

•Дані автентифікації користувачів пов'язані з їхніми записами підключень.

Ця схема бази даних підтримує соціальну мережу або додаток для знайомств з можливостями обміну повідомленнями, спільного використання фотографій та профілями користувачів.

Таким чином, було розроблено базу даних вебсистеми для знайомства людей, побудовано схему бази даних.

3.3 Розробка модулів системи

Модуль аутентифікації відповідає за реєстрацію нових користувачів та входу в систему. Він реалізований у контролері AccountController і взаємодіє з сервісом токенів (ITokenService), менеджером користувачів (UserManager<AppUser>) та мапером DTO (IMapper).

Основні функції:

- Register(RegisterDto registerDto) (рисунок 3.2). Дана функція забезпечує безпечне створення нового облікового запису. Після валідації вхідних даних RegisterDto у контролері AccountController з використанням IMapper формується об’єкт AppUser, логін приводиться до нижнього регістру, а через UserManager.CreateAsync виконується запис користувача та хешування пароля. У разі успіху додається роль «Member», і сервіс JWT (ITokenService) генерує токен, який разом із основними даними про користувача повертається клієнту. Відповідно до результату операції контролер віддає HTTP-код 201 або повідомлення про помилку у випадку конфлікту чи невдалої валідації.

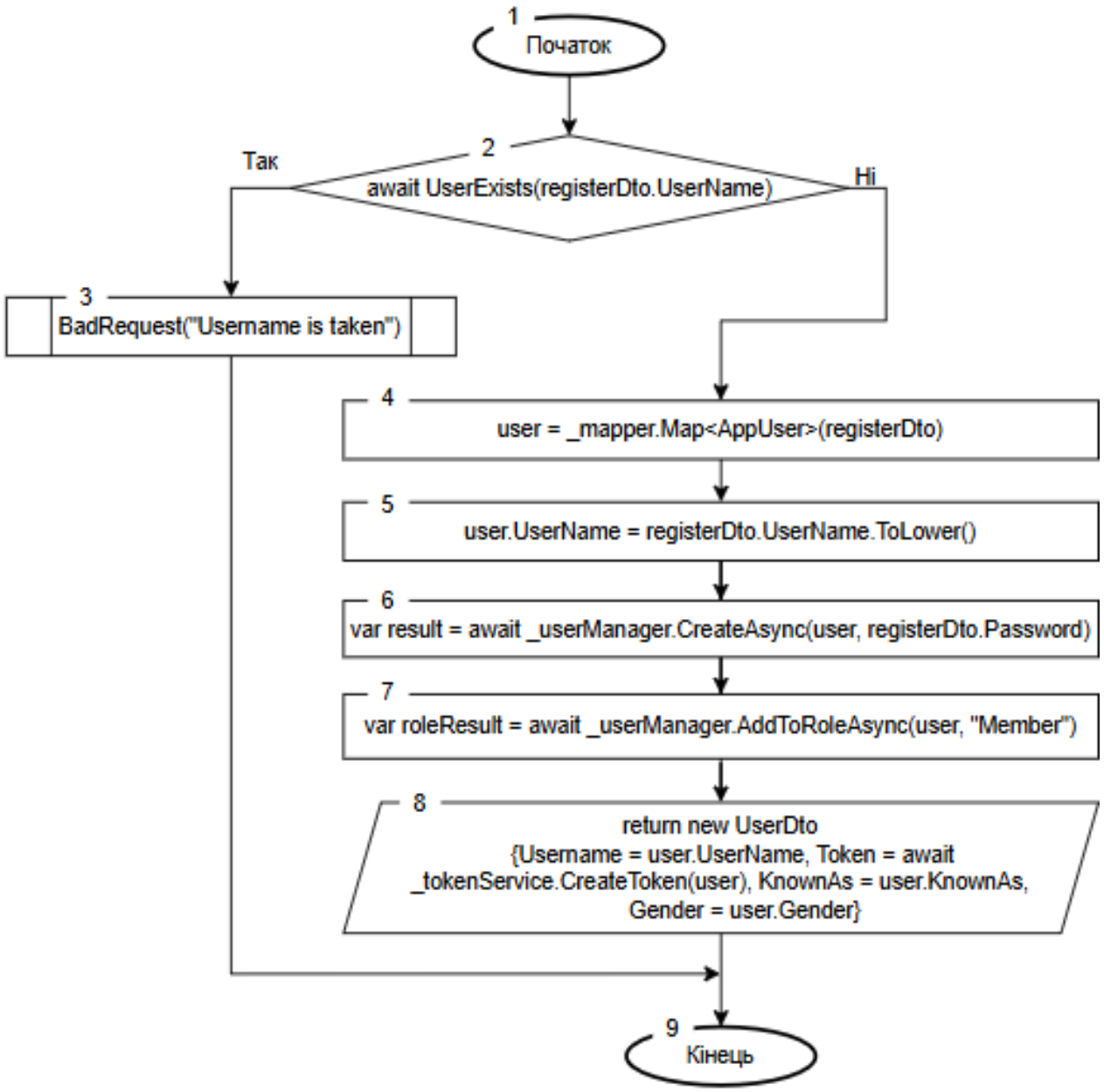

Рисунок 3.2 — Алгоритм функції Register

- Login(LoginDto loginDto) (рисунок 3.3). Алгоритм входу починається із пошуку користувача за UserName у базі разом із його фотографіями. Після того, як запис знайдено, викликається CheckPasswordAsync, щоб порівняти хеші паролів. У разі коректного пароля ITokenService.CreateToken формує новий JWT, а контролер повертає DTO, що містить логін, токен і URL головного фото. Якщо ім’я або пароль некоректні, клієнту надходить HTTP 401 із відповідним повідомленням.

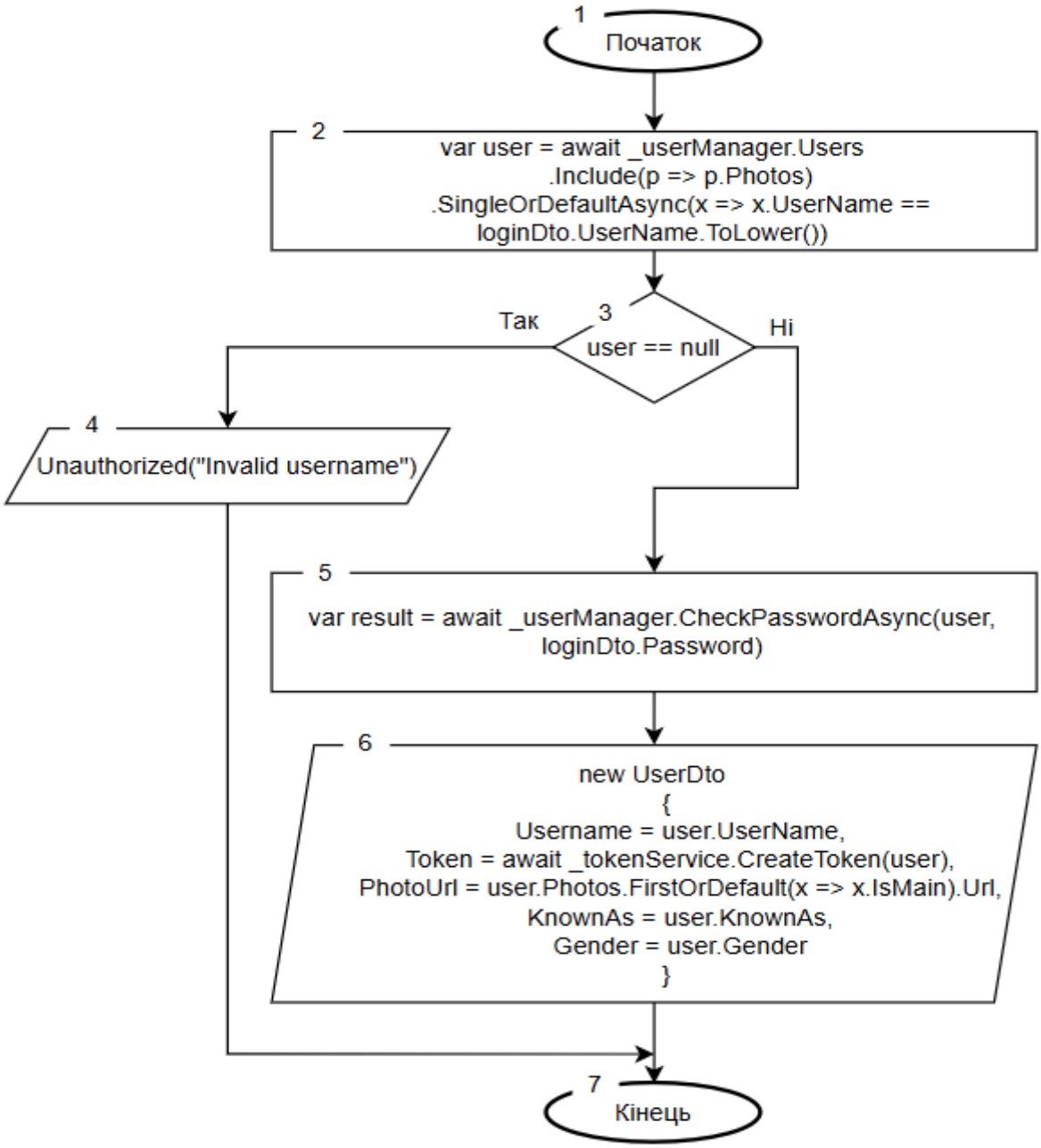

Рисунок 3.3 — Алгоритм функції Login

Модуль управління профілем користувача реалізовано у контролері UsersController. Він надає API для отримання списку користувачів, деталей профілю, оновлення власного профілю та керування фотографіями (додавання, виділення основної, видалення). Використовує IUnitOfWork, IPhotoService та IMapper.

Основні функції:

- GetUsers(UserParams userParams) (рисунок 3.4). Параметри запиту

UserParams спочатку доповнюються поточним ім’ям користувача й, за відсутності вказаної статі, автоматично встановлюють пошук протилежної статі. Далі через UserRepository.GetMembersAsync отримується список MemberDto. Контролер додає до HTTP-заголовків інформацію про пагінацію та повертає результат клієнту. Це дозволяє реалізувати нескінченну прокрутку або класичну пагінацію простим чином.

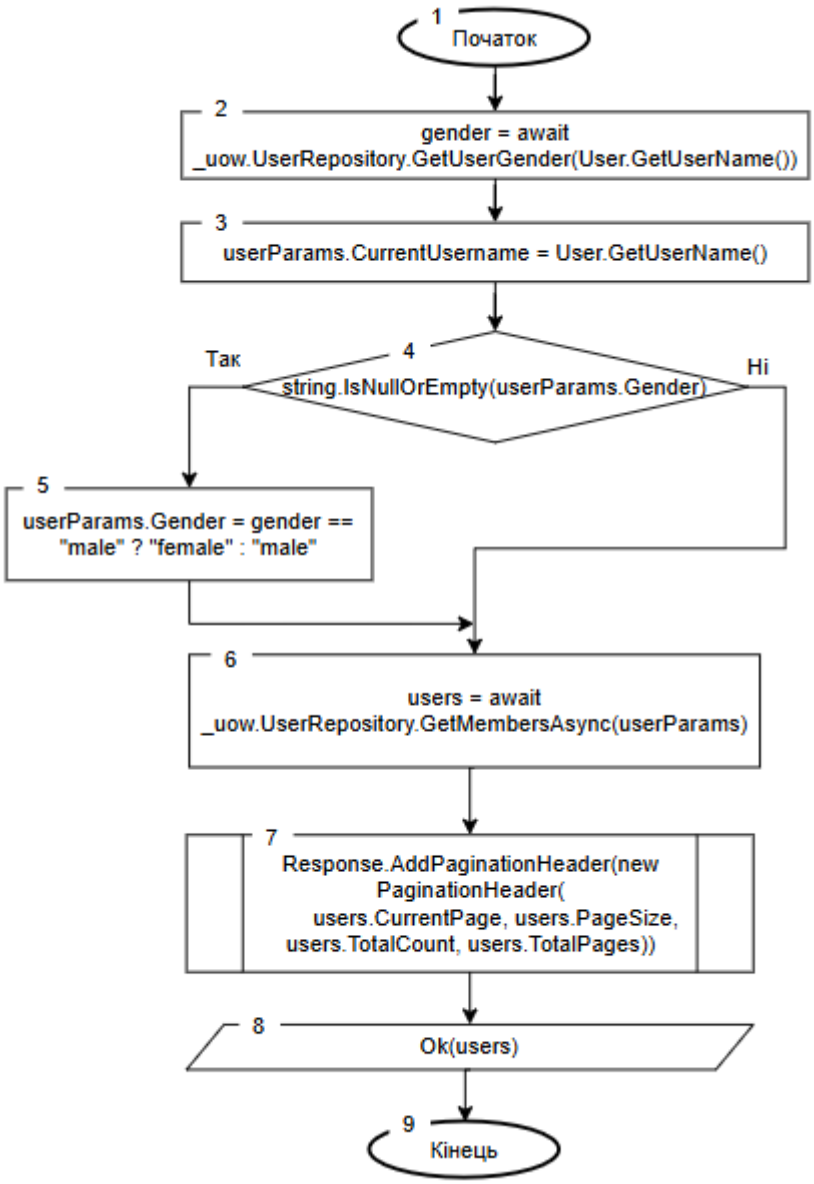

Рисунок 3.4 — Алгоритм функції GetUsers

- GetUser(string username): повертає деталі одного користувача.
- UpdateUser(MemberUpdateDto memberUpdateDto) (рисунок 3.5). Після

отримання MemberUpdateDto контролер завантажує з бази поточного користувача, відображає поля DTO на властивості AppUser за допомогою AutoMapper і викликає Complete() з unit of work. У разі успішного збереження повертається HTTP 204 (No Content), що сигналізує клієнту про успіх, або HTTP 400 із повідомленням про помилку в іншому випадку .

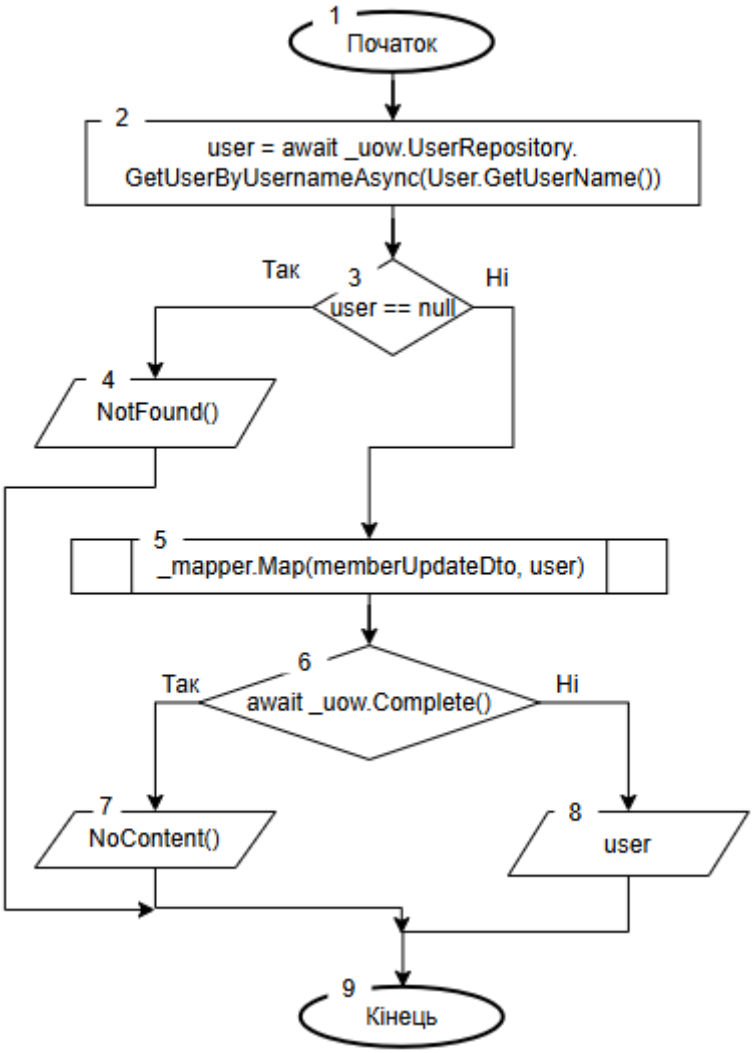

Рисунок 3.5 — Алгоритм функції UpdateUser

- AddPhoto(IFormFile file) (рисунок 3.6). Клієнт надсилає файл у FormData, контролер передає його в PhotoService.AddPhotoAsync, який завантажує зображення в Cloudinary і повертає uploadResult. Контролер перевіряє наявність помилок, створює сутність Photo зі URL та PublicId, додає її до колекції user.Photos і зберігає зміни. При успіху клієнт отримує HTTP 201 із DTO нової

фотографії, що дозволяє одразу відобразити її в UI .

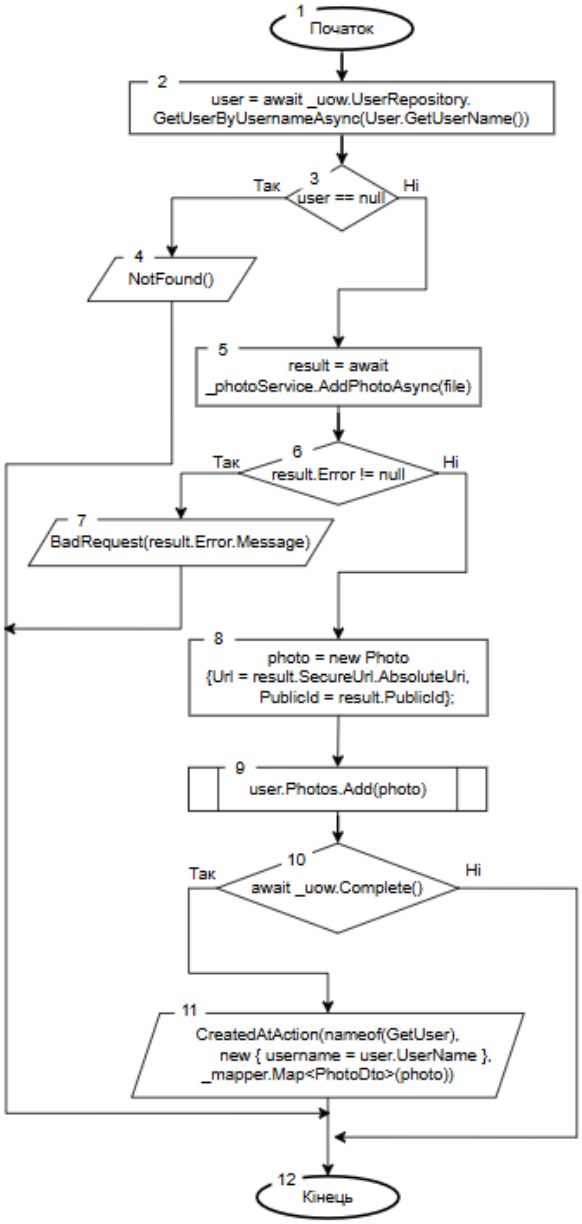

Рисунок 3.6 — Алгоритм функції AddPhoto

Модуль «лайки» дозволяє користувачам ставити лайки один одному та отримувати список користувачів, яких вони лайкнули чи хто лайкнув їх. Реалізовано у LikesController з використанням IUnitOfWork Основні функції:

- AddLike(string username) (рисунок 3.7). При лайку профілю виконується LikesController.AddLike(username). Алгоритм перевіряє, щоб користувач не

лайкнув себе, завантажує автора та цільового користувача, а також поточні лайки автора. Якщо лайк ще не існує, створюється сутність UserLike, додається до колекції sourceUser.LikedUsers і викликається Complete(). У разі взаємного лайку сервер поверне HTTP 201 і запустить сповіщення про «матч».

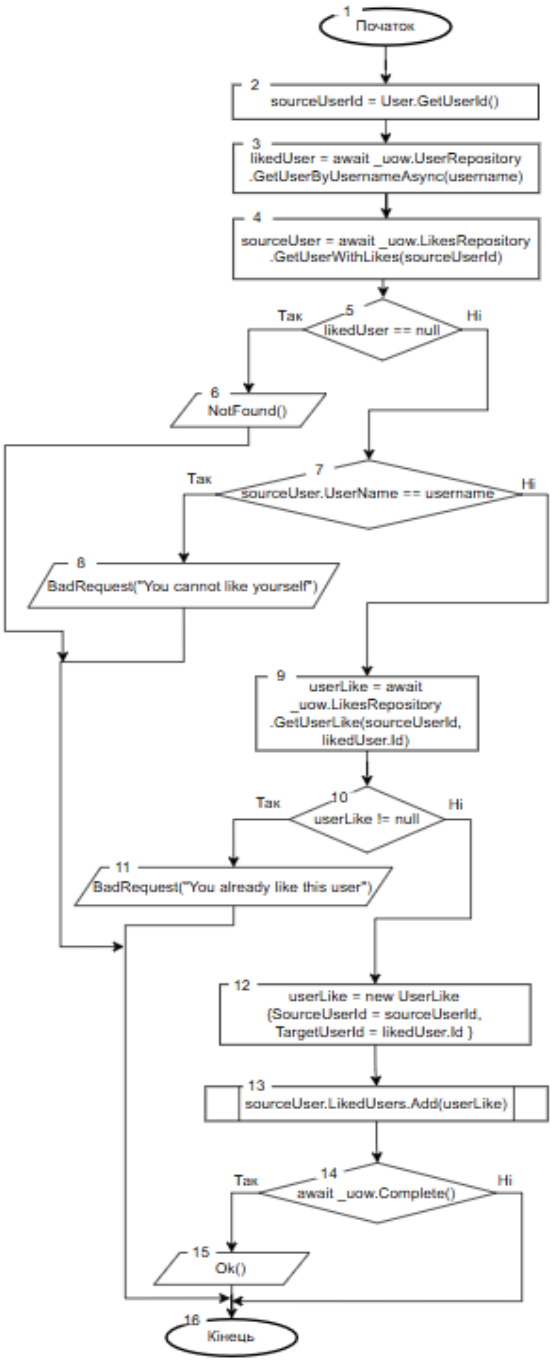

Рисунок 3.7 — Алгоритм функції AddLike

- GetUserLikes(LikesParams likesParams) (рисунок 3.8). Контролер отримує

параметри LikesParams (тип «liked» або «likedBy»), встановлює UserId і викликає LikesRepository.GetUserLikes, що повертає пагінований список LikeDto. Додаткові заголовки пагінації інформують клієнта про стан списку, що дозволяє реалізувати нескінченний скрол або числову навігацію.

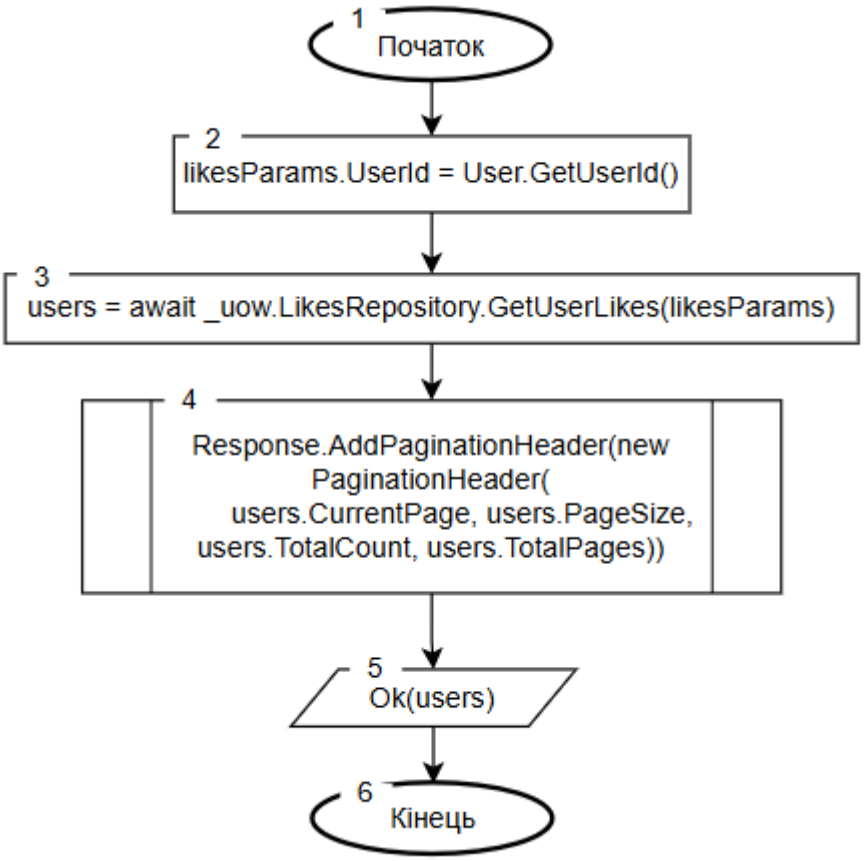

Рисунок 3.8 — Алгоритм функції GetUserLikes

У модулі обміну повідомленнями реалізовано CRUD-операції для текстових повідомлень між користувачами. Контролер MessagesController використовує шар IUnitOfWork та IMapper для зберігання, отримання та видалення повідомлень.

Основні функції:

- CreateMessage(CreateMessageDto createMessageDto) (рисунок 3.9) перевіряє, що відправник і отримувач різні, завантажує обидва об’єкти AppUser, формує новий Message з необхідними полями та додає його через MessageRepository.AddMessage(). Після Complete() контролер повертає клієнту

MessageDto, що дозволяє миттєво додати повідомлення в чат.

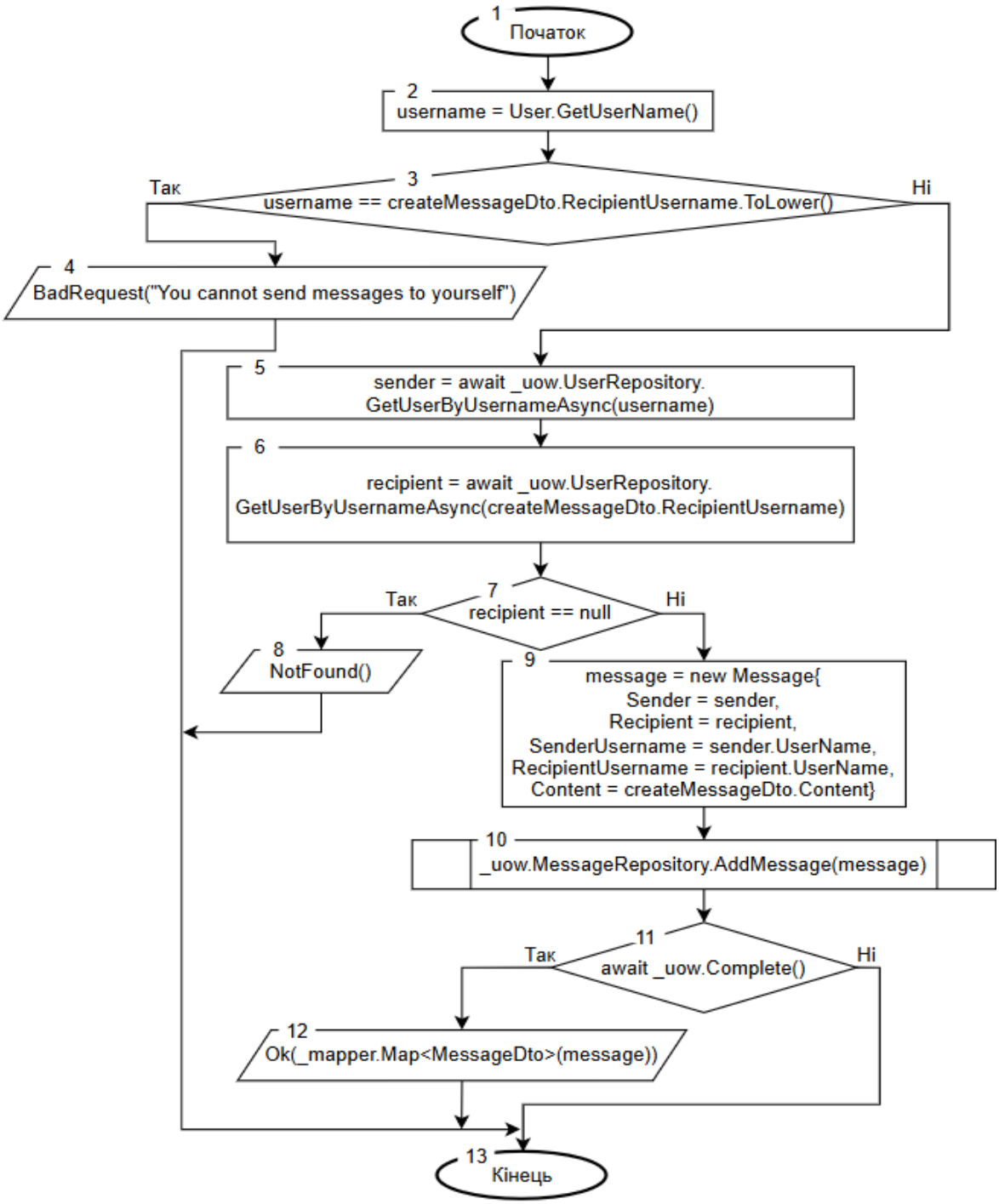

Рисунок 3.9 — Алгоритм функції GetMessage

- GetMessagesForUser(string container) (рисунок 3.10). Даний алгоритм приймає параметри MessageParams із назвою контейнера («inbox» або «outbox»), завантажує відповідний набір повідомлень через

MessageRepository.GetMessagesForUser і повертає PagedList<MessageDto>. Заголовки пагінації дозволяють клієнту коректно відобразити весь діалог.

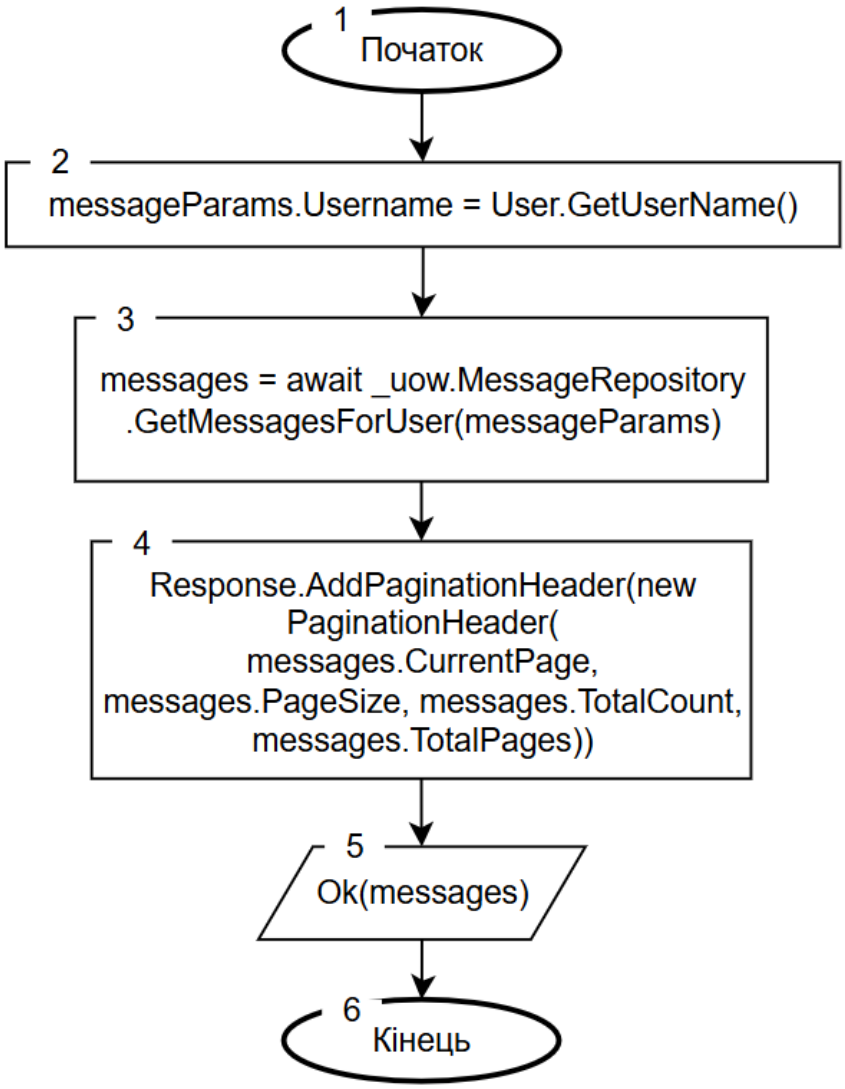

Рисунок 3.11 — Алгоритм функції GetMessageForUser

- DeleteMessage(int id)(рисунок 3.12). При видаленні повідомлення контролер спочатку завантажує його за id, позначає SenderDeleted або RecipientDeleted залежно від того, хто виконав дію, і лише коли обидва прапорці встановлено в true, видаляє запис фізично. Це гарантує, що жоден зі співрозмовників не втратить доступ до історії розмови раніше, ніж обидва підтвердять видалення.

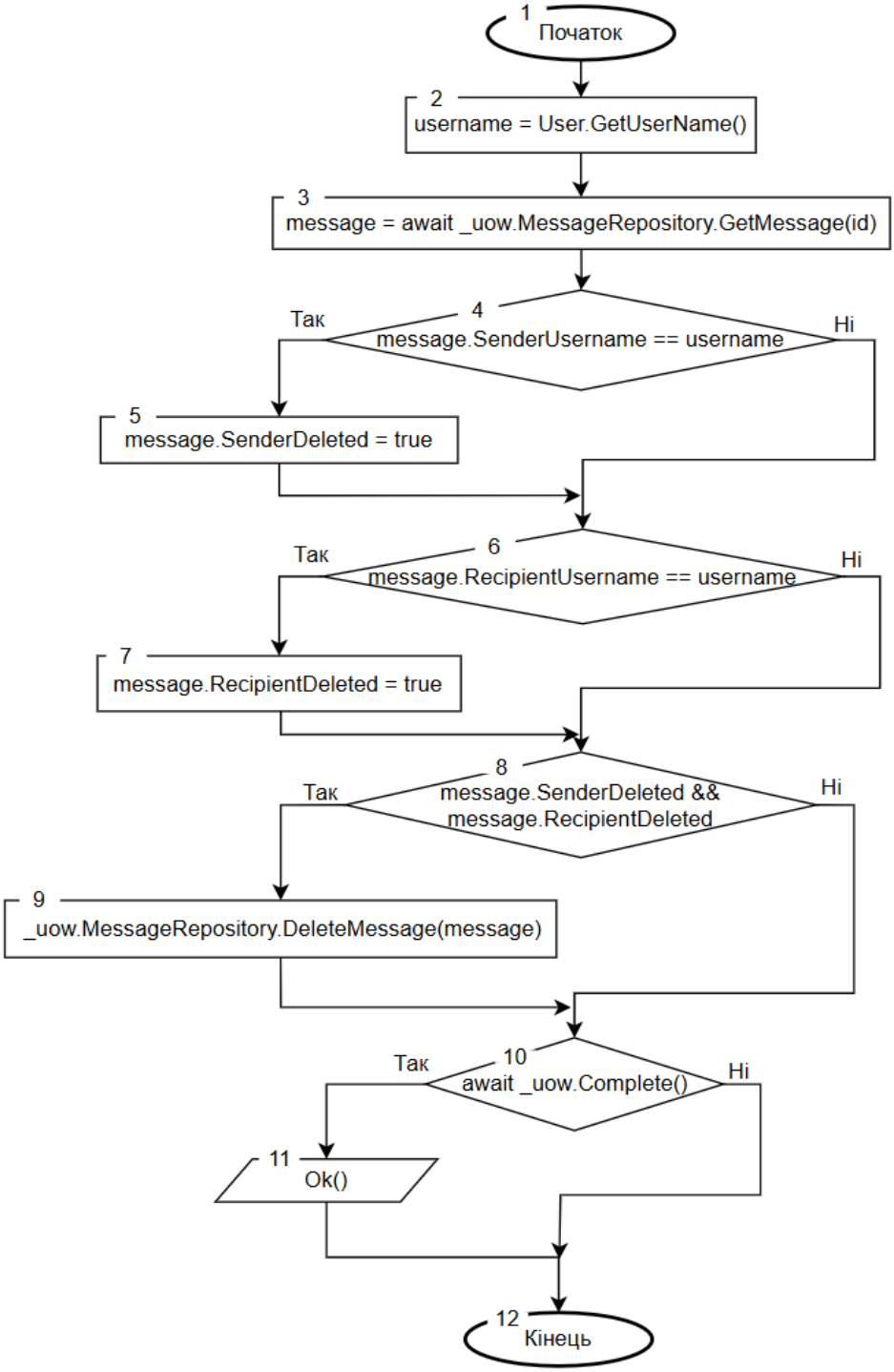

Рисунок 3.12 — Алгоритм функції DeleteMessage

Таким чином, були розроблені модулі вебсистеми для знайомства людей з використанням Angular. Розроблені модулі дозволяють реєструватися, авторизуватися користувачам, писати та видаляти повідомлення, ставити лайки, додавати фото, а адміністраторам системи керувати інформацією про користувачів.

3.4 Розробка інтерфейсу користувача

Інтерфейс користувача слугує посередником між людиною та функціональністю системи, перетворюючи складні внутрішні процеси у зрозумілі візуальні елементи та взаємодії [17]. У цьому застосунку він дозволяє реєструватися та входити в обліковий запис, налаштовувати власний профіль із завантаженням фотографій, переглядати й фільтрувати стрічку інших користувачів, ставити лайки та одразу бачити результат, а також спілкуватися за допомогою чату в реальному часі. Завдяки продуманому розташуванню елементів, адаптивному дизайну та миттєвим повідомленням про статус операцій інтерфейс робить взаємодію інтуїтивною, швидкою та приємною, що сприяє більшій залученості та задоволенню користувачів.

Вікно реєстрації дозволяє користувачам створювати аккаунт у системі, що є необхідним для використання її функціоналу. Воно складається з полів для вводу інформації про користувача, а також пароля для забезпечення санкціонованого доступу до аккаунту. Структурна схема вікна реєстрації наведена на рисунку 3.13.

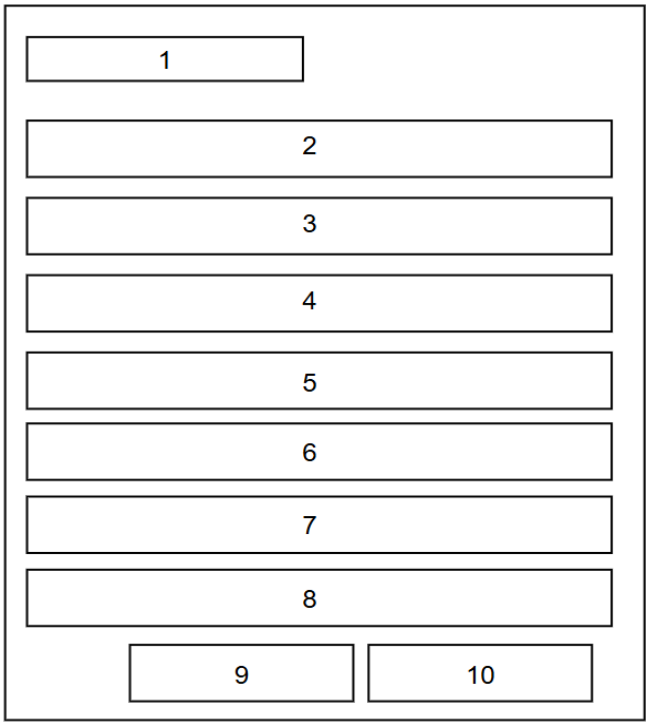

Рисунок 3.13 — Структурна схема інтерфейсу вікна реєстрації

Складові елементи вікна реєстрації:

1. Вибір статі.
2. Ім’я.
3. Прізвище.
4. Дата народження.
5. Місто.
6. Країна.
7. Пароль.
8. Повтор пароля.
9. Кнопка «Зареєструватися».
10. Кнопка «Скасувати».

На рисунку 3.14 наведено структурну схему інтерфейсу вікна пошуку анкет. Воно дозволяє налаштовувати фільтри пошуку та переглядати знайдені та цими фільтрами анкети.

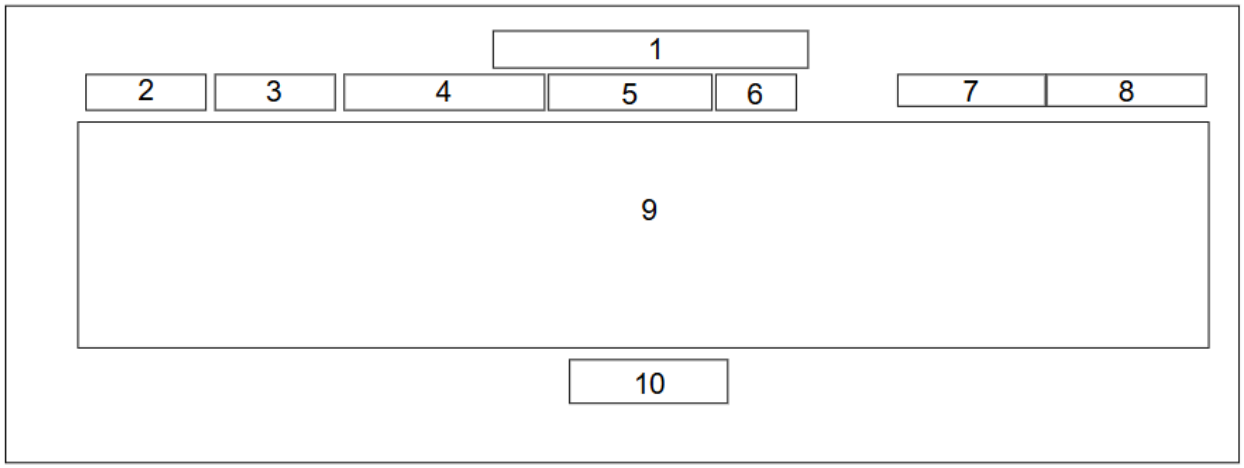

Рисунок 3.14 — Структурна схема інтерфейсу вікна пошуку анкет

Складові вікна пошуку анкет:

1. Кількість збігів за критеріями пошуку.
2. Поле для введення мінімального віку пошуку.
3. Поле для введення максимального віку пошуку.
4. Вибір статі користувачів для пошуку.

5. Кнопка «Застосувати фільтри».

6. Кнопка «Скинути фільтри».

7. Сортування за останньою активністю.

8. Сортування за найновішими учасниками.

9. Перелік знайдених анкет.

10. Меню для перемикання між сторінками анкет.

Вікно інтерфейсу користувача дозволяє переглядати інформацію про користувача, його фото, а також звідси перейти до процесу спілкування з користувачем. Структурна схема інтерфейсу цього вікна наведена на рисунку 3.15.

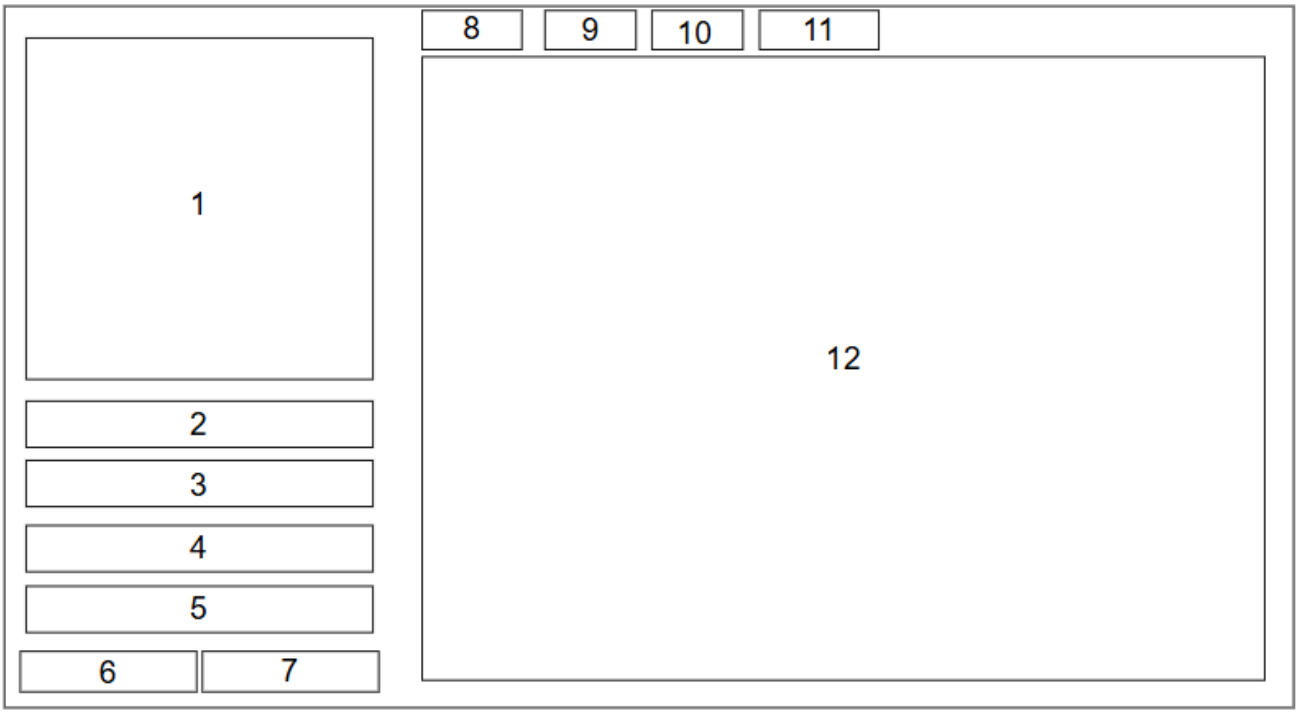

Рисунок 3.15 — Структурна схема інтерфейсу сторінки користувача

Складові сторінки користувача:

1. Фото.

2. Локація.

3. Вік.

4. Остання активність у мережі.

5. Дата реєстрації.

6. Кнопка «Подобається».

7. Кнопка «Написати повідомлення».

8. Кнопка для перегляду опису користувача.

9. Кнопка для перегляду інтересів.

10. Кнопка для перегляду фотографій.

11. Кнопка «Написати повідомлення».

12. Опис користувача.

Таким чином, було розроблено інтерфейс вебсистеми для знайомства людей. Розроблений інтерфейс дозволяє створювати акаунт користувача, шукати анкети інших користувачів, переглядати детальну інформацію, та вести чат з ними.

3.5 Висновки

У третьому розділі було проведено варіантний аналіз та обґрунтування вибору мови програмування розробки, обрано мову програмування JavaScript для реалізації вебсистеми для знайомства людей. Було розроблено базу даних, що відповідає за роботу з даними користувачів, та їх збереження. Розроблено модулі вебсистеми, наведено алгоритми їх роботи. Розроблено інтерфейс користувача.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз методів тестування

Юніт-тестування перевіряє окремі класи та методи без залучення зовнішніх ресурсів [18]. Для контролерів (AccountController, UsersController, LikesController, MessagesController) пишуть тести з використанням таких інструментів, як xUnit або NUnit, де залежності (наприклад, IUserRepository, ITokenService, IPhotoService, IUnitOfWork) підміняються моками (Moq). Юніт-тести гарантують коректну валідацію вхідних параметрів, обробку помилок та формування відповідей згідно з очікуваннями (HTTP-статуси, DTO). Також юніт-тести для сервісів (PhotoService, TokenService) перевіряють бізнес-логіку: генерування JWT, формування запитів у Cloudinary, обробку помилок завантаження тощо.

Інтеграційні тести залучають реальні компоненти стеку: базу даних (наприклад, EF Core InMemory або тестова SQL-СУБД), ASP.NET Core TestServer або WebApplicationFactory для підняття вбудованого сервера [19]. Вони допомагають перевірити роботу репозиторіїв (UserRepository, LikesRepository, MessageRepository), міграції БД, налаштування DI-контейнера, а також реальні HTTP-ендпоінти: реєстрація, логін, CRUD для фото і повідомлень тощо. Інтеграційні тести можуть покривати сценарії перегляду стрічки користувачів із пагінацією та фільтрами, підтверджуючи, що заголовки пагінації додаються коректно.

End-to-End тестування імітує поведінку реального користувача через UI (браузер або мобільний клієнт). Для вебверсії можна використовувати Cypress або Playwright: перевірити авторизацію, завантаження фото, лайки й відкриття чату, надсилання й отримання повідомлень у реальному часі.

Оскільки додаток підтримує real-time-зв’язок через SignalR і має інтенсивні запити до API (стрічка профілів, пагінація), слід провести навантажувальні тести (JMeter, k6). Моделюють сотні–тисячі паралельних клієнтів: відкриття чатів, масові лайки, завантаження фотографій, щоб виявити

вузькі місця в масштабуванні, час відповіді та стабільність з’єднання WebSocket [20].

Після автоматичних перевірок виконуються сценарії, які складно або автоматизувати: зручність інтерфейсу, адаптивність на різних розмірах екранів, поведінка при поганому з’єднанні, обробка великих фото та нетипових даних. Дослідницьке тестування допомагає знайти нетривіальні помилки ідеї використання, наприклад, при спробі лайнути себе або надіслати занадто велике повідомлення.

4.2 Тестування вебсистеми

Сторінка реєстрації реалізована у вигляді форми, яка містить поля для введення імені користувача, пароля та підтвердження пароля, а також кнопку «Register». Над полями виводяться повідомлення валідації (наприклад, «Username is required» або «Passwords do not match»). У шапці сторінки є назва застосунку, а внизу—посилання на форму входу для вже зареєстрованих користувачів.

Сторінка реєстрації наведена на рисунку 4.1.

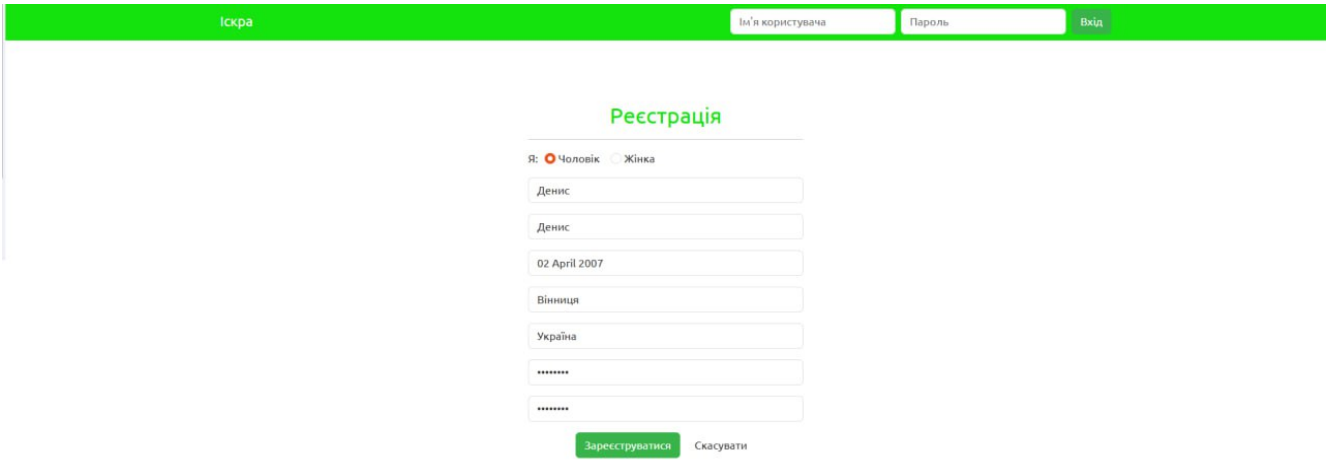

Рисунок 4.1 — Сторінка реєстрації

На рисунку 4.2 наведено результат пошуку анкет користувачів за заданими фільтрами, а саме: вік від 18 до 99, стать Females (тобто жінки).

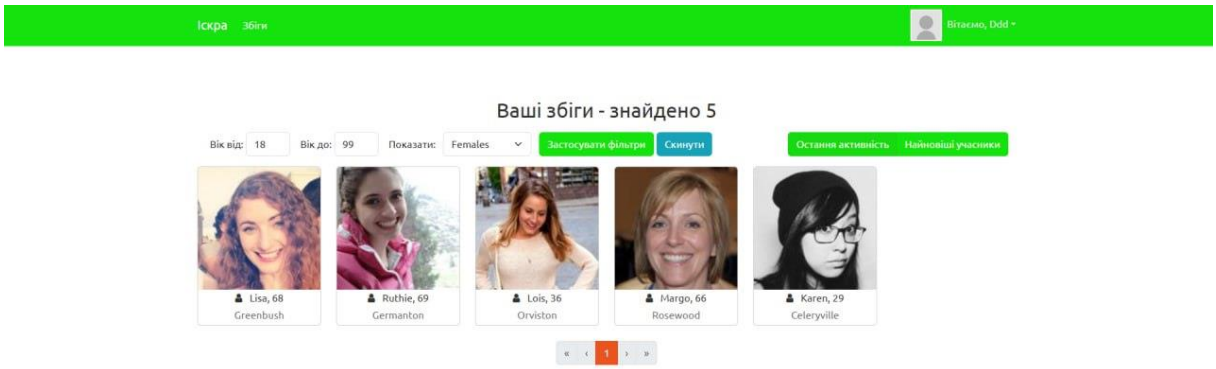

Рисунок 4.2 — Результати пошуку анкет за заданими фільтрами

Змінимо фільтри пошуку, а саме: вік від 20 до 50, стать Males (тобто чоловіки). Оновлені результати пошуку наведені на рисунку 4.3.

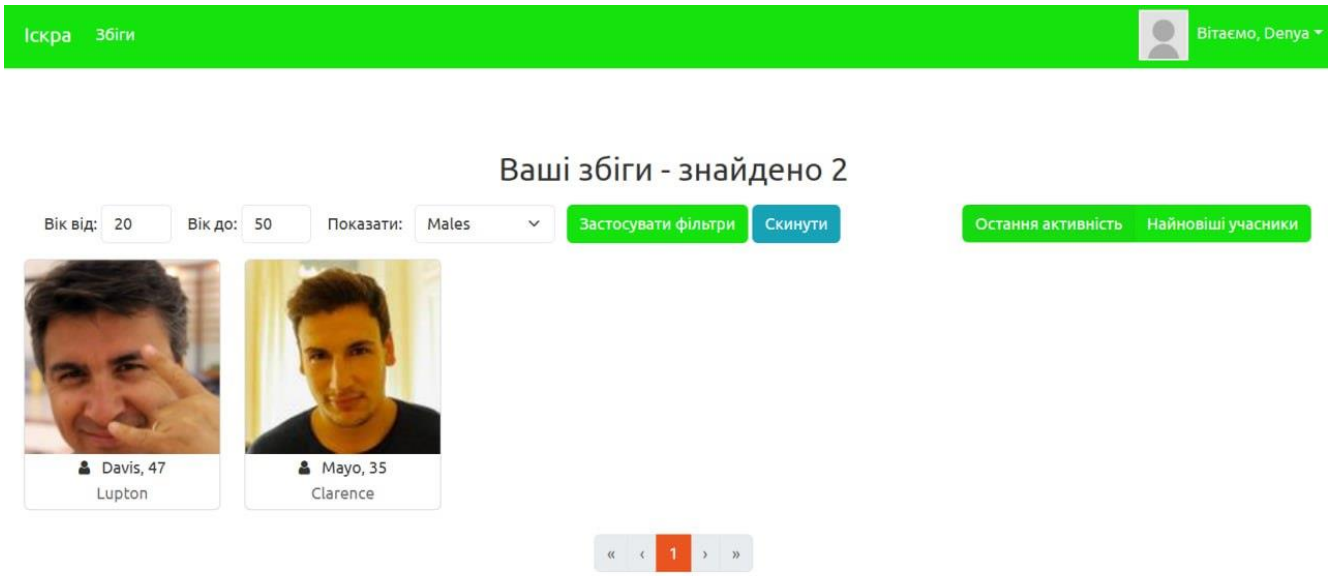

Рисунок 4.3 — Результати пошуку при змінених фільтрах

На рисунку 4.4 можна переглянути профіль користувача. Він містить

детальну інформацію, а саме фото профілю, вік, місто проживання, останню активність у мережі, опис користувача, інтереси, фото.

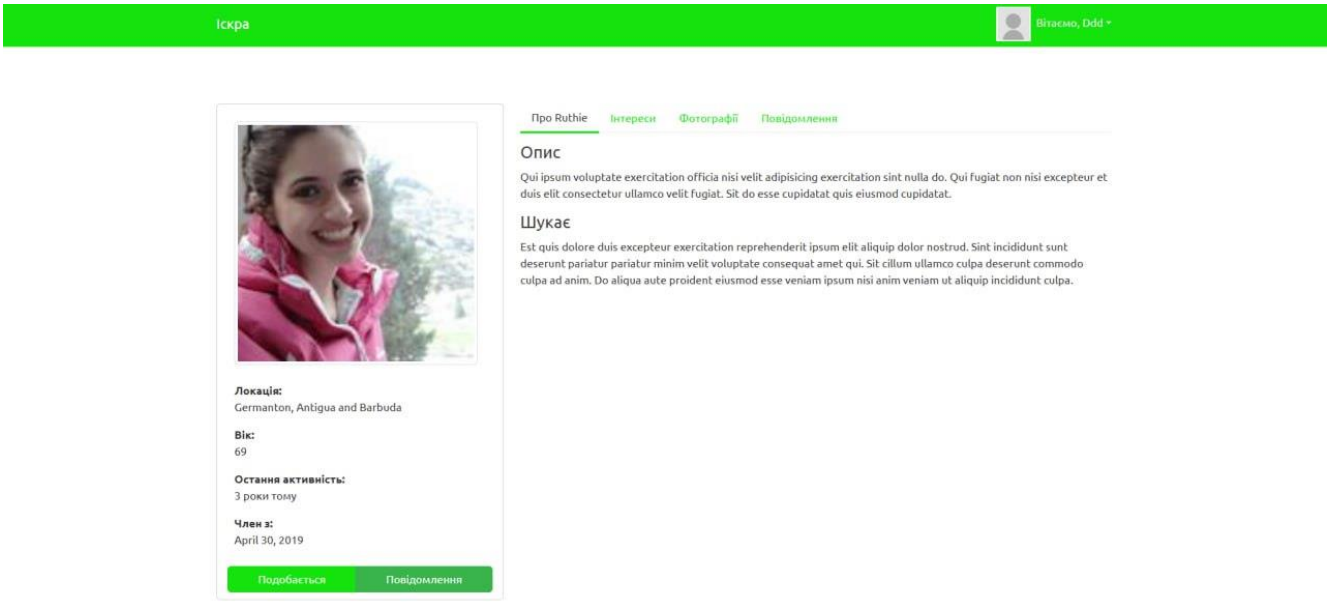

Рисунок 4.4 — Сторінка профілю користувача

На рисунку 4.5 продемонстровано відкрите меню для ведення чату для спілкування з користувачем.

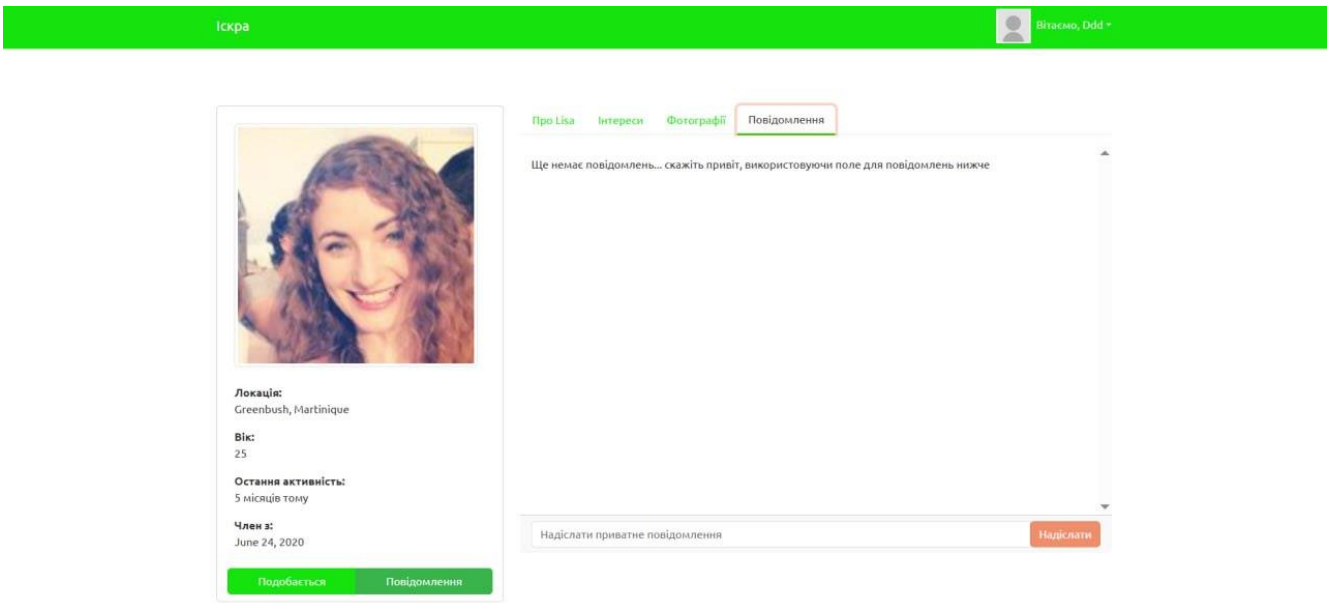

Рисунок 4.5 — Вікно чату з користувачем

Додамо фото до власного профілю, як це наведено на рисунку 4.6. Вони

додаються до профілю не одразу, адже спершу вони повинні пройти перевірку на відсутність порушень та підтвердження автентичності фото та їх приналежності власнику сторінки.

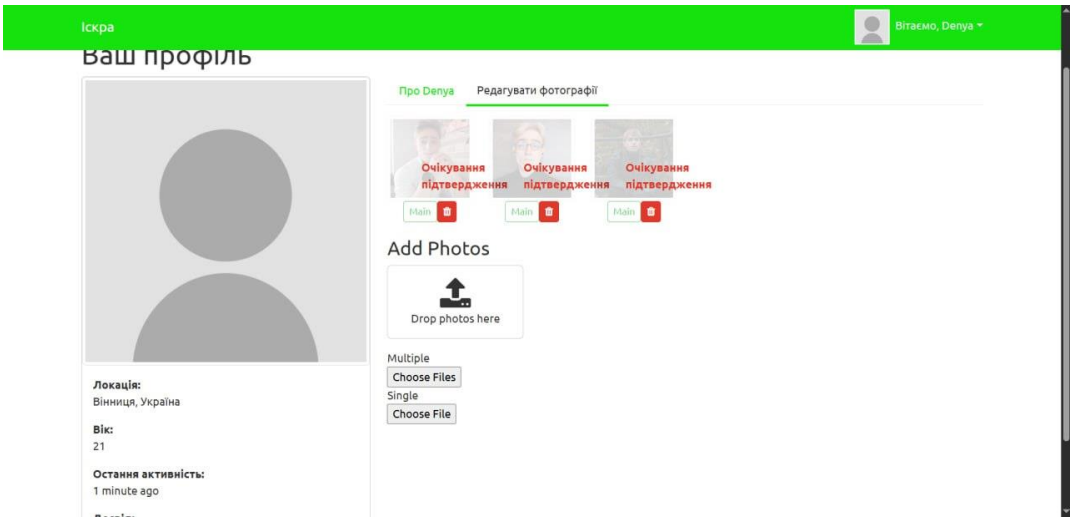

Рисунок 4.6 — Додавання фото до профілю до перевірки

На рисунку 4.7 наведено панель адміністратора вона містить список користувачів, дозволяє переглядати профілі користувачів та керувати ними.

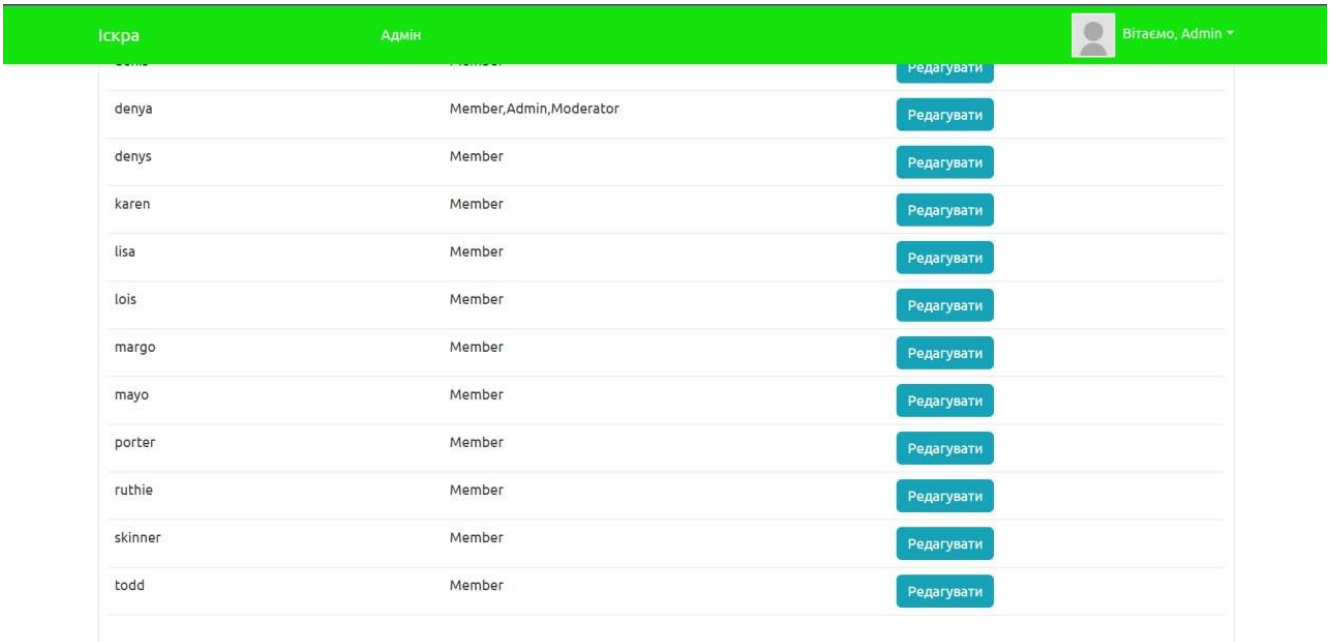

Рисунок 4.7 — Список користувачів

На рисунку 4.8 наведено панель управління аккаунтом користувача для перевірки фото. Адміністратор може підтвердити або відхилити додавання

завантажених користувачем фото.

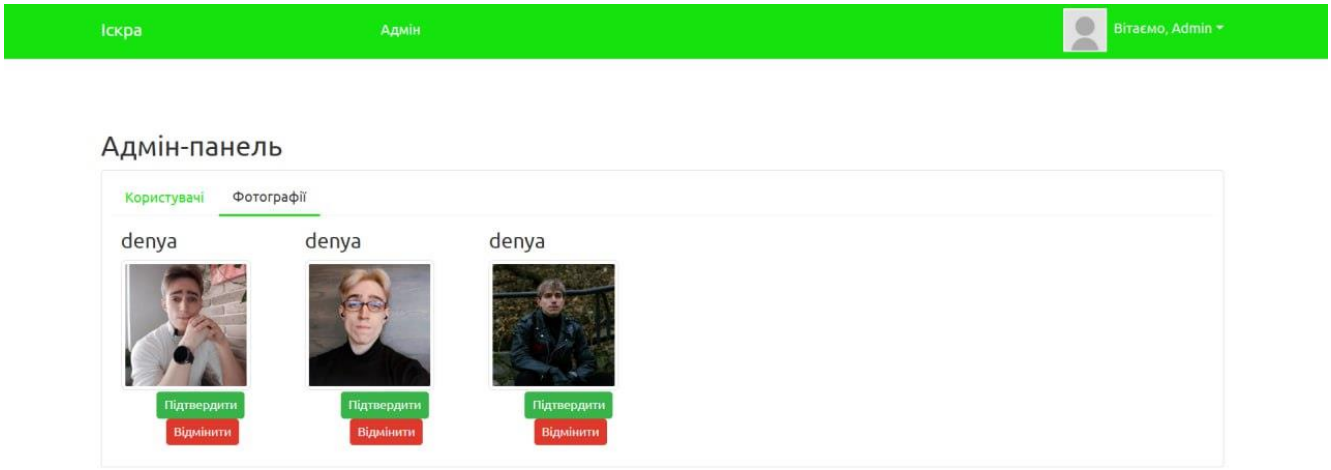

Рисунок 4.8 — Адмін-панель управління фото

Після підтвердження фото, вони повноцінно відображаються у системі (рисунок 4.9).

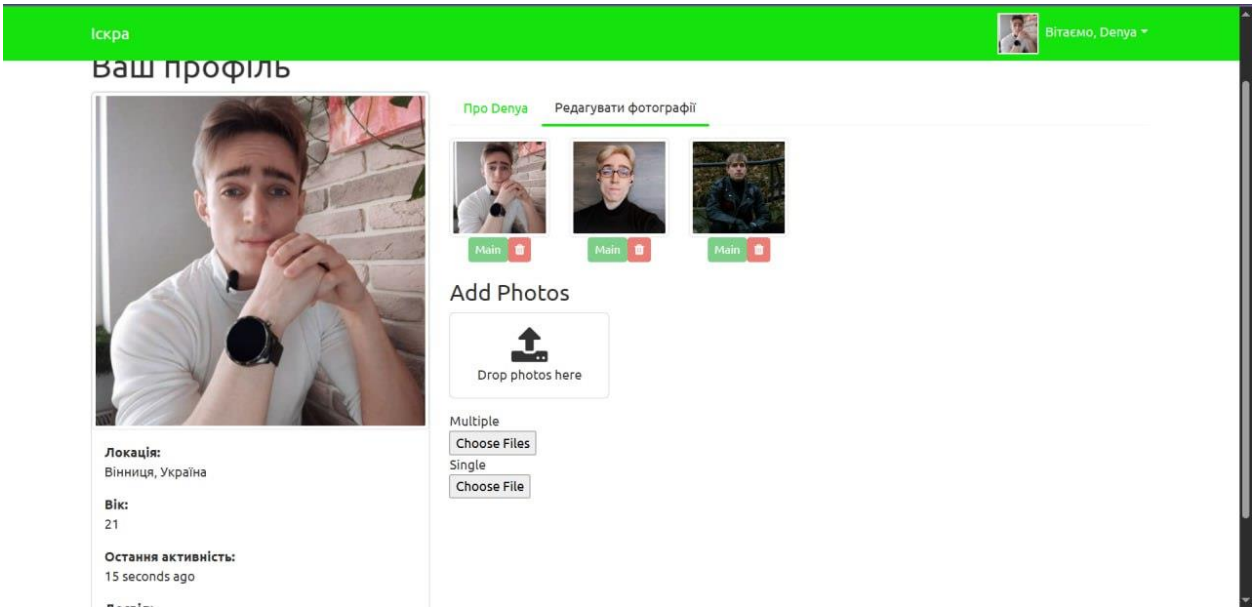

Рисунок 4.9 — Фото користувача після перевірки

На рисунку 4.10 наведено список користувачів, які поставили «лайк» обраному користувачеві.

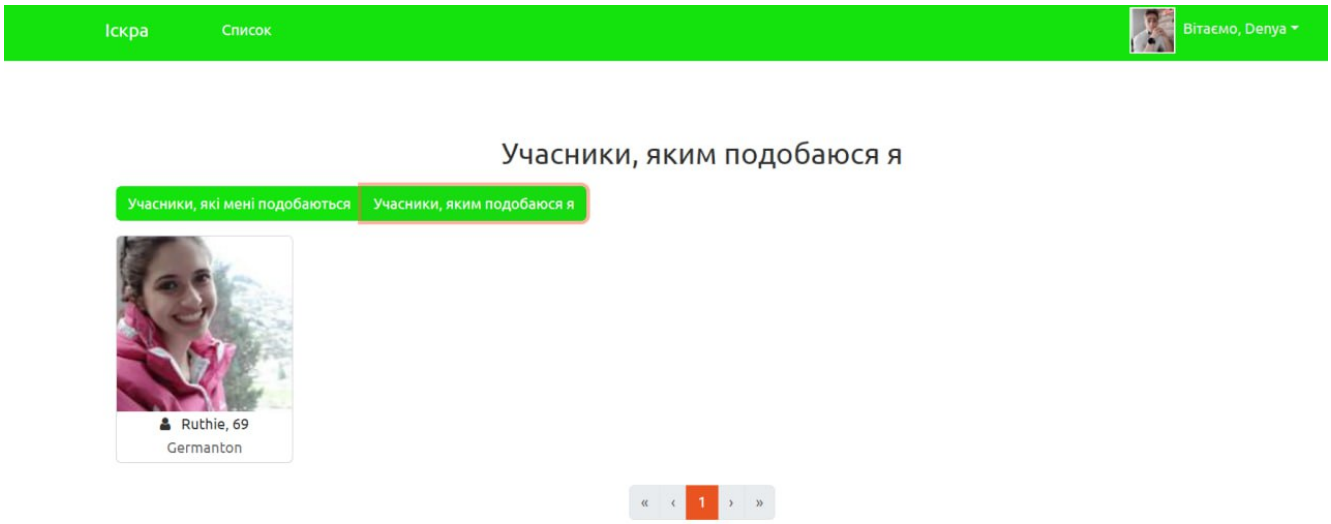

Рисунок 4.10 — Лайк від іншої людини

Таким чином, було проведено тестування системи для знайомства людей з використанням Angular. Продемонстровано можливості розробленої системи. Система виконує поставлені на початку розробки завдання.

4.3 Висновки

У четвертому розділі, було проведено аналіз методів тестування вебсистеми для знайомства людей з використанням Angular. Було проведено тестування розробленої системи, наведено приклади використання, продемонстровано можливості розробленого застосунку. Розроблена вебсистема виконує поставлені на початку розробки завдання.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено вебсистему для знайомства людей з використанням Angular. Розроблена вебсистема дозволяє користувачам безпечно реєструватися та входити в обліковий запис, створювати й редагувати власні профілі з можливістю завантаження фотографій у хмарне сховище, швидко знаходити сумісних співрозмовників за допомогою механізму «лайків» і миттєвого «матчу», а також обмінюватися повідомленнями в реальному часі. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [21].

Для розробки було використано мову програмування JavaScript, середовище розробки Visual Studio Code, бібліотеки SignalR, технологію JSON Web Tokens, AutoMapper.

У першому розділі було проведено аналіз стану питання розробки вебсистем для знайомства людей, розглянуто існуючі аналоги та проведено порівняльний аналіз, продемонстровано актуальність власної розробки. Проведено аналіз методів розв’язання задачі, проведено постановку задач дослідження.

У другому розділі, було проведено аналіз інформаційного забезпечення системи, описано, які дані використовуються та генеруються при роботі системи. Розроблено архітектуру системи. Розроблено метод формування репутації користувача, метод вказання точок зустрічі. Побудовано загальний алгоритм роботи системи, продемонстровано його у вигляді відповідної блок-схеми, на якій відображено увесь процес роботи системи.

У третьому розділі було проведено варіантний аналіз та обґрунтування вибору мови програмування розробки, обрано мову програмування JavaScript для реалізації вебсистеми для знайомства людей. Було розроблено базу даних, що відповідає за роботу з даними користувачів, та їх збереження. Розроблено модулі вебсистеми, наведено алгоритми їх роботи. Розроблено інтерфейс користувача.

У четвертому розділі, було проведено аналіз методів тестування вебсистеми для знайомства людей з використанням Angular. Було проведено тестування розробленої системи, наведено приклади використання, продемонстровано можливості розробленого застосунку. Розроблена вебсистема виконує поставлені на початку розробки завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The virtues and downsides of online dating [Електронний ресурс] – Режим доступу до ресурсу: https://www.pewresearch.org/internet/2020/02/06/the-virtues-and-downsides-of-online-dating

2. Online dating (Worldwide) [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/outlook/emo/dating-services/online-dating/worldwide

3. Online Dating Market Size, Share & Trends Analysis Report By Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.grandviewresearch.com/industry-analysis/online-dating-market-report#

4. Куксін Д.О. Переваги та недоліки додатків для знайомств / Д.О. Куксін, Г.О. Черноволик // Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025) : збірник доповідей [Електронний ресурс]. — Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24497/20192

5. An Engineer’s Guide to Dating App Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.cometchat.com/blog/building-your-own-dating-app

6. Tinder [Електронний ресурс] – Режим доступу до ресурсу: https://tinder.com/uk

7. Badoo [Електронний ресурс] – Режим доступу до ресурсу: https://badoo.com/uk/

8. Bumble [Електронний ресурс] – Режим доступу до ресурсу: https://bumble.com/

9. Most popular technologies for webframe [Електронний ресурс] – Режим доступу до ресурсу: https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe

10. Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://www.valuecoders.com/blog/technology-and-apps/vue-js-angularjs-reactjs-updates/

11. GraphQL [Електронний ресурс] – Режим доступу до ресурсу: https://graphql.org/

12. Angular securiry XSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.okta.com/blog/2022/07/21/angular-security-xss

13. D.Flanagan. JavaScript. The Definitive Guide. Master the World's Most-Used Programming Language 7th Edition. Farnham: O'Reilly Media, 2020. 706с.

14. Майк МакГрат. C# Programming in easy steps: Modern coding with C# 10 and .NET 6. Holly Walk: In Easy Steps, 2022. 192с.

15. A.Sedunov. Kotlin In-Depth: A Guide to a Multipurpose Programming Language. Нойда: BPB Publications, 2022. 687с.

16. What is a database? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/think/topics/database

17. What is User Interface Design? [Електронний ресурс] – Режим доступу до ресурсу: https://www.interaction-design.org/literature/topics/ui-design?srsltid=AfmBOoqRdpb-Nz2IguRSIie4Wyxl47Rfj6c8726oHntmoz8IuRgKSgxl

18. Lucas da Costa. Testing JavaScript Applications. Нью-Йорк: Manning, 2022. 512с.

19. Аттіла Ковач. Modern Software Testing Techniques: A Practical Guide for Developers and Testers. Нью-Йорк: Apress, 2024. 266с.

20. Анжеліна Самару. Software Testing: An ISTQB-BCS Certified Tester Foundation Level guide (CTFL v4.0) - Fifth edition. Лондон: BCS, 2024. 283с.

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О.Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

68

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
« 21 » березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка вебсистеми для знайомства людей з використанням фреймворку Angular»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Черноволик Г.О.
« 21 » березня 2025 р.

Виконав:
студент гр. 4ПІ-21б Куксін Д.О.
« 21 » березня 2025 р.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка вебсистеми для знайомства людей з використанням фреймворку Angular».

Галузь застосування – вебтехнології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є покращення процесу пошуку та знаймомства людей за допомогою спеціалізованої вебсистеми, що дозволяє пошук користувачів та комунікацію з ними.

4. Вихідні дані для проведення НДР

1. Online Dating Market Size, Share & Trends Analysis Report By Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.grandviewresearch.com/industry-analysis/online-dating-market-report#
2. An Engineer’s Guide to Dating App Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.cometchat.com/blog/building-your-own-dating-app
3. Angular securiry XSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.okta.com/blog/2022/07/21/angular-security-xss
4. D.Flanagan. JavaScript. The Definitive Guide. Master the World's Most-Used Programming Language 7th Edition. Farnham: O’Reilly Media, 2020. 706с.

5. Технічні вимоги

Вхідні дані — дані користувача, фото, дії користувача.

Вихідні дані — списки користувачів, повідомлення.

6. Конструктивні вимоги.

Інтерфейс програми поовинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку вебсистем для знайомства людей	25.03.25 - 05.04.25
2	Розробка структури та алгоритмів програмного застосунку	06.04.25 - 18.04.25
3	Варіантний аналіз та вибір мови програмування розробки	19.04.25 - 21.04.25
4	Розробка вебсистеми	22.04.25 - 17.05.25
5	Тестування програмного забезпечення	18.05.25 - 25.05.25
6	Оформлення матеріалів до захисту БКР	26.05.25 - 30.05.25

10. Порядок контролю та прийняття.

Усі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. К'афедрою згідно з графіком захисту.

71

Додаток Б – Протокол перевірки на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка вебсистеми для знайомства людей з використанням фреймворку Angular»

Тип роботи: БКР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8.2 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

□У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

□У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений

Керівник _____ Черноволик Г.О. к.т.н., доцент каф.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Куксін Д.О.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

```
import { Component, OnInit } from '@angular/core';
import { Message } from '../models/message';
import { Pagination } from '../models/pagination';
import { MessageService } from '../services/message.service';

@Component({
  selector: 'app-messages',
  templateUrl: './messages.component.html',
  styleUrls: ['./messages.component.css']
})
export class MessagesComponent implements OnInit {
  messages?: Message[];
  pagination?: Pagination;
  container = 'Unread';
  pageNumber = 1;
  pageSize = 5;
  loading = false;

  constructor(private messageService: MessageService) { }

  ngOnInit(): void {
    this.loadMessages();
  }

  loadMessages() {
    this.loading = true;
    this.messageService.getMessages(this.pageNumber, this.pageSize, this.container).subscribe({
      next: response => {
        this.messages = response.result;
        this.pagination = response.pagination;
        this.loading = false;
      }
    });
  }

  deleteMessage(id: number) {
    this.messageService.deleteMessage(id).subscribe({
      next: () => this.messages?.splice(this.messages.findIndex(m => m.id === id), 1)
    });
```

```
  }

  pageChanged(event: any) {
    if (this.pageNumber !== event.page) {
      this.pageNumber = event.page;
      this.loadMessages();
    }
  }

}

import { Component, OnInit } from '@angular/core';
import { Message } from '../models/message';
import { Pagination } from '../models/pagination';
import { MessageService } from '../services/message.service';

@Component({
  selector: 'app-messages',
  templateUrl: './messages.component.html',
  styleUrls: ['./messages.component.css']
})
export class MessagesComponent implements OnInit {
  messages?: Message[];
  pagination?: Pagination;
  container = 'Unread';
  pageNumber = 1;
  pageSize = 5;
  loading = false;

  constructor(private messageService: MessageService) { }

  ngOnInit(): void {
    this.loadMessages();
  }

  loadMessages() {
    this.loading = true;
    this.messageService.getMessages(this.pageNumber, this.pageSize, this.container).subscribe({
      next: response => {
        this.messages = response.result;
        this.pagination = response.pagination;
```

```
        this.loading = false;
      }
    });
  }

  deleteMessage(id: number) {
    this.messageService.deleteMessage(id).subscribe({
      next: () => this.messages?.splice(this.messages.findIndex(m => m.id === id), 1)
    });
  }

  pageChanged(event: any) {
    if (this.pageNumber !== event.page) {
      this.pageNumber = event.page;
      this.loadMessages();
    }
  }

}

import { Component, OnInit } from '@angular/core';
import { BsModalRef, BsModalService, ModalOptions } from 'ngx-bootstrap/modal';
import { RolesModalComponent } from '../../modals/roles-modal/roles-modal.component';
import { User } from '../../models/user';
import { AdminService } from '../../services/admin.service';

@Component({
  selector: 'app-user-management',
  templateUrl: './user-management.component.html',
  styleUrls: ['./user-management.component.css']
})
export class UserManagementComponent implements OnInit {
  users: User[] = [];
  bsModalRef: BsModalRef<RolesModalComponent> = new BsModalRef<RolesModalComponent>();
  availableRoles = [
    'Admin',
    'Moderator',
    'Member'
  ];

  constructor(private adminService: AdminService,
```

```
    private modalService: BsModalService) { }

  ngOnInit(): void {
    this.getUsersWithRoles();
  }

  getUsersWithRoles() {
    this.adminService.getUsersWithRoles().subscribe({
      next: users => this.users = users
    });
  }

  openRolesModal(user: User) {
    const config = {
      class: 'modal-dialog-centered',
      initialState: {
        username: user.username,
        availableRoles: this.availableRoles,
        selectedRoles: [...user.roles]
      }
    };
    this.bsModalRef = this.modalService.show(RolesModalComponent, config);
    this.bsModalRef.onHide?.subscribe({
      next: () => {
        const selectedRoles = this.bsModalRef.content?.selectedRoles;
        if (!this.arrayEqual(selectedRoles!, user.roles)) {
          this.adminService.updateUserRoles(user.username, selectedRoles!).subscribe({
            next: roles => user.roles = roles
          });
        }
      }
    });
  }

  private arrayEqual(arr1: any[], arr2: any[]) {
    return JSON.stringify(arr1.sort()) === JSON.stringify(arr2.sort());
  }

}

import { Component, OnInit } from '@angular/core';
```

```
import { Photo } from '../../models/photo';
import { AdminService } from '../../services/admin.service';

@Component({
  selector: 'app-photo-management',
  templateUrl: './photo-management.component.html',
  styleUrls: ['./photo-management.component.css']
})
export class PhotoManagementComponent implements OnInit {

  photos: Photo[] = [];

  constructor(private adminService: AdminService) { }

  ngOnInit(): void {
    this.getPhotosForApproval();
  }

  getPhotosForApproval() {
    this.adminService.getPhotosForApproval().subscribe({
      next: photos => this.photos = photos
    });
  }

  approve(id: number) {
    this.adminService.approvePhoto(id).subscribe({
      next: () => {
        const index = this.photos.findIndex(p => p.id === id);
        this.photos.splice(index, 1);
      }
    });
  }

  reject(id: number) {
    this.adminService.rejectPhoto(id).subscribe({
      next: () => {
        const index = this.photos.findIndex(p => p.id === id);
        this.photos.splice(index, 1);
      }
    });
  }
```

```
}

import { Component, EventEmitter, Input, OnInit, Output } from '@angular/core';
import { AbstractControl, FormBuilder, FormControl, FormGroup, ValidatorFn, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { ToastrService } from 'ngx-toastr';
import { AccountService } from '../services/account.service';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  @Output() cancelRegister = new EventEmitter();
  registerForm: FormGroup = new FormGroup({});
  maxDate: Date = new Date();
  validationErrors: string[] | undefined;

  constructor(private accountService: AccountService,
    private toastr: ToastrService,
    private fb: FormBuilder,
    private router: Router
    ) { }

  ngOnInit(): void {
    this.initForm();
    this.maxDate.setFullYear(this.maxDate.getFullYear() - 18);
  }

  initForm() {
    this.registerForm = this.fb.group({
      gender: ['male'],
      username: ['', Validators.required],
      knownAs: ['', Validators.required],
      dateOfBirth: ['', Validators.required],
      city: ['', Validators.required],
      country: ['', Validators.required],
      password: ['', [Validators.required, Validators.minLength(4), Validators.maxLength(8)]],
```

```
    confirmPassword: ['', [Validators.required,
      Validators.minLength(4),
      Validators.maxLength(8),
      this.matchValues('password')]]
  });

  this.registerForm.controls['password'].valueChanges.subscribe({
    next: () => this.registerForm.controls['confirmPassword'].updateValueAndValidity()
  });
}

matchValues(matchTo: string): ValidatorFn {
  return (control: AbstractControl) => {
    return control.value === control.parent?.get(matchTo)?.value ? null : { notMatching: true }
  }
}

register() {
  const dob = this.getDateOnly(this.registerForm.controls['dateOfBirth'].value);
  const values = { ...this.registerForm.value, dateOfBirth: dob };

  this.accountService.register(values).subscribe({
    next: () => {
      this.router.navigateByUrl('/members');
    },
    error: error => {
      this.validationErrors = error
    }
  });
}

cancel() {
  this.cancelRegister.emit(false);
}

private getDateOnly(dob: string | undefined) {
  if (!dob) return;

  let theDob = new Date(dob);
  return new Date(theDob.setMinutes(theDob.getMinutes() - theDob.getTimezoneOffset()))
```

```
      .toISOString()
      .slice(0, 10);
  }
}

import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { HubConnection, HubConnectionBuilder } from '@microsoft/signalr';
import { ToastrService } from 'ngx-toastr';
import { BehaviorSubject, take } from 'rxjs';
import { environment } from '../../environments/environment';
import { User } from '../models/user';

@Injectable({
  providedIn: 'root'
})
export class PresenseService {
  hubUrl = environment.hubUrl;
  private hubConnection?: HubConnection;
  private onlineUsersSource = new BehaviorSubject<string[]>([]);
  onlineUsers$ = this.onlineUsersSource.asObservable();

  constructor(private toastr: ToastrService, private router: Router) { }

  createHubConnection(user: User) {
    this.hubConnection = new HubConnectionBuilder()
      .withUrl(this.hubUrl + 'presense', {
        accessTokenFactory: () => user.token
      })
      .withAutomaticReconnect()
      .build();

    this.hubConnection.start().catch(error => console.log(error));

    this.hubConnection.on('UserIsOnline', username => {
      this.onlineUsers$.pipe(take(1)).subscribe({
        next: usernames => this.onlineUsersSource.next([...usernames, username])
      });
    });

    this.hubConnection.on('UserIsOffline', username => {
```

```
      this.onlineUsers$.pipe(take(1)).subscribe({
        next: usernames => this.onlineUsersSource.next(usernames.filter(x => x !== username))
      });
    });

    this.hubConnection.on('GetOnlineUsers', usernames => {
      this.onlineUsersSource.next(usernames);
    });

    this.hubConnection.on('NewMessageReceived', ({ username, knownAs }) => {
      this.toastr.info(knownAs + ' has sent you a new message! Click me to see it')
        .onTap.pipe(take(1)).subscribe({
          next: () => this.router.navigateByUrl('/members/' + username + '?tab=Messages')
        });
    });
  }

  stopHubConnection() {
    this.hubConnection?.stop().catch(error => console.log(error));
  }
}

import { HttpClient, HttpParams } from "@angular/common/http";
import { map } from "rxjs";
import { PaginatedResult } from "../models/pagination";

export function getPaginatedResult<T>(url: string, params: HttpParams, http: HttpClient) {
  const paginatedResult: PaginatedResult<T> = new PaginatedResult<T>;
  return http.get<T>(url, { observe: 'response', params: params }).pipe(
    map(response => {
      if (response.body) {
        paginatedResult.result = response.body;
      }
      const pagination = response.headers.get('Pagination');
      if (pagination) {
        paginatedResult.pagination = JSON.parse(pagination);
      }

      return paginatedResult;
    })
  );
```

```
}

export function getPaginationHeaders(pageNumber: number, pageSize: number) {
  let params = new HttpParams();

  params = params.append('pageNumber', pageNumber);
  params = params.append('pageSize', pageSize);

  return params;
}

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { HubConnection, HubConnectionBuilder } from '@microsoft/signalr';
import { BehaviorSubject, take } from 'rxjs';
import { environment } from '../../environments/environment';
import { Group } from '../models/group';
import { Message } from '../models/message';
import { User } from '../models/user';
import { BusyService } from './busy.service';
import { getPaginatedResult, getPaginationHeaders } from './paginationHelper';

@Injectable({
  providedIn: 'root'
})
export class MessageService {
  baseUrl = environment.apiUrl;
  hubUrl = environment.hubUrl;
  private hubConnection?: HubConnection;
  private messageThreadSource = new BehaviorSubject<Message[]>([]);
  messageThread$ = this.messageThreadSource.asObservable();

  constructor(private http: HttpClient, private busyService: BusyService) { }

  createHubConnection(user: User, otherUsername: string) {
    this.busyService.busy();
    this.hubConnection = new HubConnectionBuilder()
      .withUrl(this.hubUrl + 'messages?user=' + otherUsername, {
        accessTokenFactory: () => user.token
      })
      .withAutomaticReconnect()
```

```
    .build();

  this.hubConnection.start()
    .catch(error => console.log(error))
    .finally(() => this.busyService.idle());

  this.hubConnection.on('ReceiveMessageThread', messages => {
    this.messageThreadSource.next(messages);
    console.log(messages);
  });

  this.hubConnection.on('UpdatedGroup', (group: Group) => {
    if (group.connections.some(x => x.username == otherUsername)) {
      this.messageThread$.pipe(take(1)).subscribe({
        next: messages => {
          messages.forEach(message => {
            if (!message.dateRead)
              message.dateRead = new Date(Date.now())

          });

          this.messageThreadSource.next([...messages]);
        }
      });
    }
  });

  this.hubConnection.on('NewMessage', message => {
    this.messageThread$.pipe(take(1)).subscribe({
      next: messages => {
        this.messageThreadSource.next([...messages, message]);
      }
    });
  });
}

stopHubConnection() {
  if (this.hubConnection) {
    this.messageThreadSource.next([]);
    this.hubConnection.stop();
  }
```

```
  }

  getMessages(pageNumber: number, pageSize: number, container: string) {
    let params = getPaginationHeaders(pageNumber, pageSize);
    params = params.append('Container', container);
    return getPaginatedResult<Message[]>(this.baseUrl + 'messages', params, this.http);
  }

  getMessageThread(username: string) {
    return this.http.get<Message[]>(this.baseUrl + 'messages/thread/' + username);
  }

  async sendMessage(username: string, content: string) {
    return this.hubConnection?.invoke('SendMessage', { recipientUsername: username, content })
      .catch(error => console.log(error));
  }

  deleteMessage(id: number) {
    return this.http.delete<Message>(this.baseUrl + 'messages/' + id);
  }
}

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { BehaviorSubject, map } from 'rxjs';
import { environment } from '../../environments/environment';
import { User } from '../models/user';
import { PresenseService } from './presense.service';

@Injectable({
  providedIn: 'root'
})
export class AccountService {
  baseUrl: string = environment.apiUrl;

  private currentUserSource = new BehaviorSubject<User | null>(null);
  currentUser$ = this.currentUserSource.asObservable();

  constructor(private http: HttpClient, private presenseService: PresenseService) { }

  login(model: any) {
```

```
    return this.http.post<User>(this.baseUrl + 'account/login', model).pipe(
      map((response: User) => {
        const user = response;
        if (user) {
          this.setCurrentUser(user);
        }
      })
    );
  }

  register(model: any) {
    return this.http.post<User>(this.baseUrl + 'account/register', model).pipe(
      map(user => {
        if (user) {
          this.setCurrentUser(user);
        }
      })
    );
  }

  setCurrentUser(user: User) {
    user.roles = [];
    const roles = this.getDecodedToken(user.token).role;
    Array.isArray(roles) ? user.roles = roles : user.roles.push(roles);
    localStorage.setItem('user', JSON.stringify(user));
    this.currentUserSource.next(user);
    this.presenseService.createHubConnection(user);
  }

  logout() {
    localStorage.removeItem('user');
    this.currentUserSource.next(null);
    this.presenseService.stopHubConnection();
  }

  getDecodedToken(token: string) {
    return JSON.parse(atob(token.split('.')[1]));
  }
}

import { NgModule } from '@angular/core';
```

```
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule, HTTP_INTERCEPTORS } from "@angular/common/http";
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { AppRoutingModule } from './app-routing.module';

import { SharedModule } from './_modules/shared.module';

import { AppComponent } from './app.component';
import { NavComponent } from './nav/nav.component';
import { HomeComponent } from './home/home.component';
import { RegisterComponent } from './register/register.component';
import { MemberListComponent } from './members/member-list/member-list.component';
import { ListsComponent } from './lists/lists.component';
import { MessagesComponent } from './messages/messages.component';
import { TestErrorComponent } from './errors/test-error/test-error.component';
import { ErrorInterceptor } from './_interceptors/error.interceptor';
import { NotFoundComponent } from './errors/not-found/not-found.component';
import { ServerErrorComponent } from './errors/server-error/server-error.component';
import { MemberCardComponent } from './members/member-card/member-card.component';
import { JwtInterceptor } from './_interceptors/jwt.interceptor';
import { MemberEditComponent } from './members/member-edit/member-edit.component';
import { LoadingInterceptor } from './_interceptors/loading.interceptor';
import { PhotoEditorComponent } from './members/photo-editor/photo-editor.component';
import { TextInputComponent } from './_forms/text-input/text-input.component';
import { DatePickerComponent } from './_forms/date-picker/date-picker.component';
import { AdminPanelComponent } from './admin/admin-panel/admin-panel.component';
import { HasRoleDirective } from './_directives/has-role.directive';
import { UserManagementComponent } from './admin/user-management/user-management.component';
import { PhotoManagementComponent } from './admin/photo-management/photo-management.component';
import { RolesModalComponent } from './modals/roles-modal/roles-modal.component';
import { ConfirmDialogComponent } from './modals/confirm-dialog/confirm-dialog.component';
import { RouteReuseStrategy } from '@angular/router';
import { CustomRouteReuseStrategy } from './services/customRouteReuseStrategy';




@NgModule({
  declarations: [
    AppComponent,
```

```
    NavComponent,
    HomeComponent,
    RegisterComponent,
    MemberListComponent,
    ListsComponent,
    MessagesComponent,
    TestErrorComponent,
    NotFoundComponent,
    ServerErrorComponent,
    MemberCardComponent,
    MemberEditComponent,
    PhotoEditorComponent,
    TextInputComponent,
    DatePickerComponent,
    AdminPanelComponent,
    HasRoleDirective,
    UserManagementComponent,
    PhotoManagementComponent,
    RolesModalComponent,
    ConfirmDialogComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule,
    BrowserAnimationsModule,
    SharedModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: LoadingInterceptor, multi: true },
    { provide: RouteReuseStrategy, useClass: CustomRouteReuseStrategy }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

import { HttpClient } from "@angular/common/http";
```

```
import { Component, OnInit } from "@angular/core";
import { environment } from "../../../environments/environment";

@Component({
  selector: "app-test-error",
  templateUrl: "./test-error.component.html",
  styleUrls: ["./test-error.component.css"],
})
export class TestErrorComponent implements OnInit {
  baseUrl = environment.apiUrl;
  validationErrors: string[] = [];

  constructor(private http: HttpClient) {}

  ngOnInit(): void {}

  get404Error() {
    this.http.get(this.baseUrl + "buggy/not-found").subscribe({
      next: (response) => console.log(response),
      error: (error) => console.log(error),
    });
  }

  get400Error() {
    this.http.get(this.baseUrl + "buggy/bad-request").subscribe({
      next: (response) => console.log(response),
      error: (error) => console.log(error),
    });
  }

  get500Error() {
    this.http.get(this.baseUrl + "buggy/server-error").subscribe({
      next: (response) => console.log(response),
      error: (error) => console.log(error),
    });
  }

  get401Error() {
    this.http.get(this.baseUrl + "buggy/auth").subscribe({
      next: (response) => console.log(response),
      error: (error) => console.log(error),
```

```
    });
  }

  get400ValidationError() {
    this.http.post(this.baseUrl + "account/register", {}).subscribe({
      next: (response) => console.log(response),
      error: (error) => {
        console.log(error);
        this.validationErrors = error;
      },
    });
  }
}

import { Component, OnInit } from '@angular/core';
import { Member } from '../models/member';
import { Pagination } from '../models/pagination';
import { MembersService } from '../services/members.service';

@Component({
  selector: 'app-lists',
  templateUrl: './lists.component.html',
  styleUrls: ['./lists.component.css']
})
export class ListsComponent implements OnInit {
  members: Member[] | undefined;
  predicate = 'liked';
  pageNumber = 1;
  pageSize = 5;
  pagination: Pagination | undefined;

  constructor(private memberService: MembersService) { }

  ngOnInit(): void {
    this.loadLikes();
  }

  loadLikes() {
    this.memberService.getLikes(this.predicate, this.pageNumber, this.pageSize).subscribe({
      next: response => {
        this.members = response.result;
```

```
        this.pagination = response.pagination;
      }
    });
  }

  pageChanged(event: any) {
    if (this.pageNumber !== event.page) {
      this.pageNumber = event.page;
      this.loadLikes();
    }
  }

}
```

Додаток Г – Ілюстративна частина

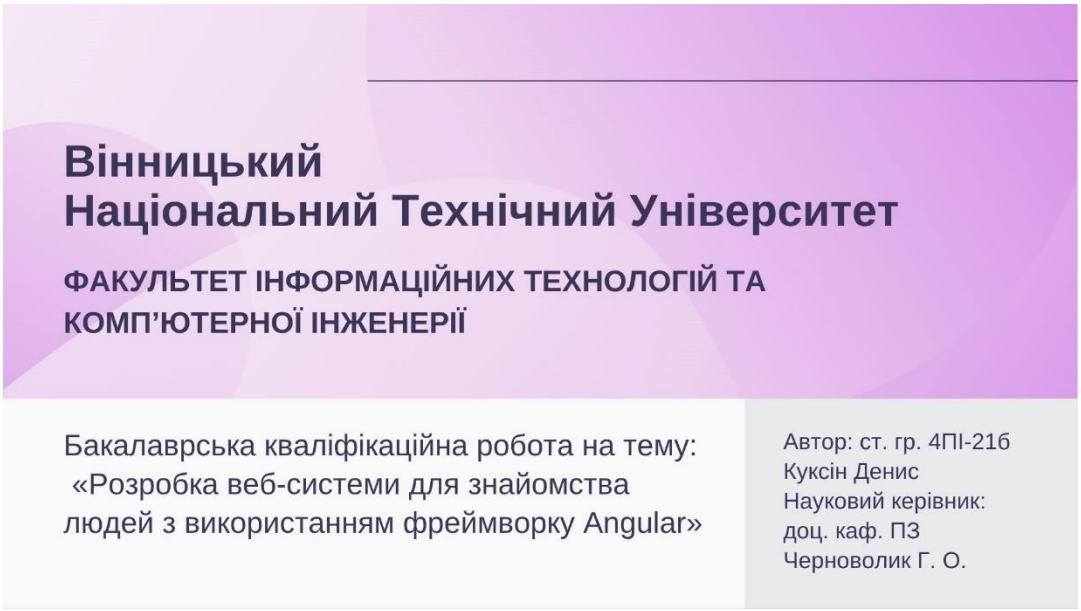

Рисунок Г.1 - слайд 1

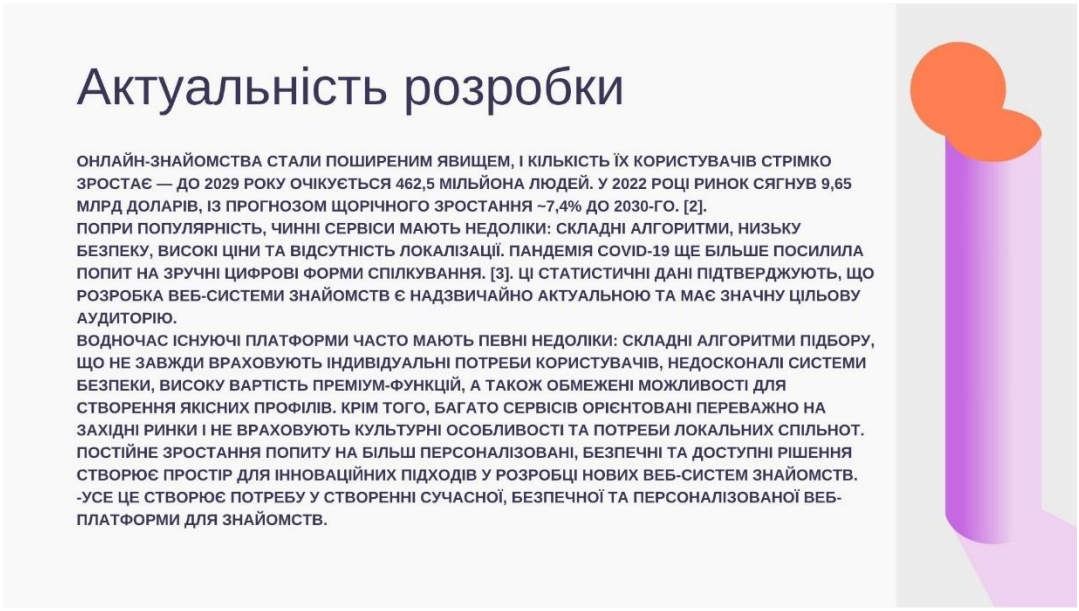

Рисунок Г.2 - слайд 2

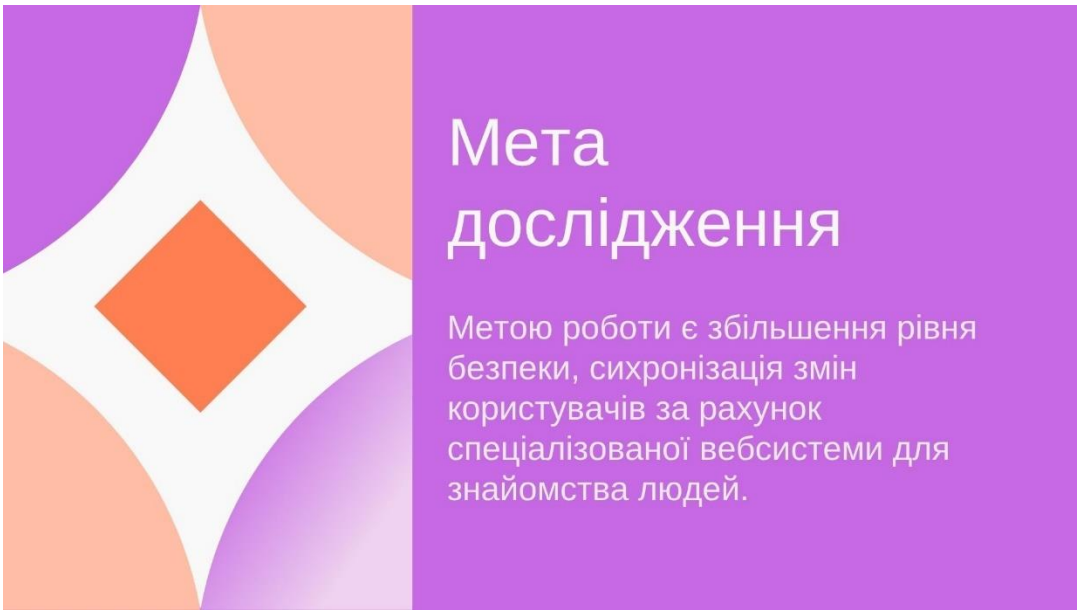

Рисунок Г.3 - слайд 3

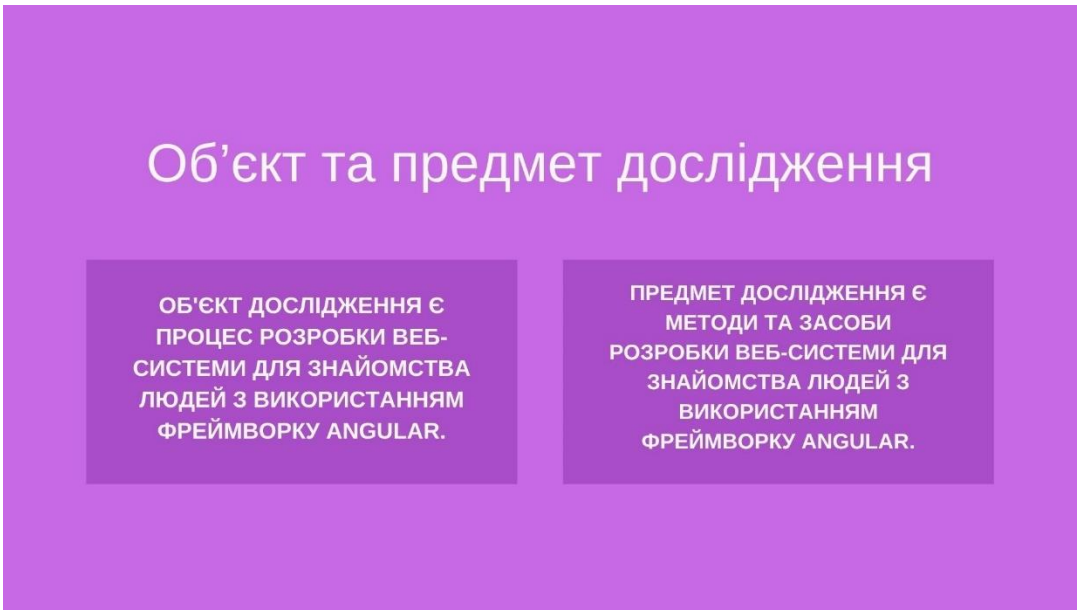

Рисунок Г.4 - слайд 4

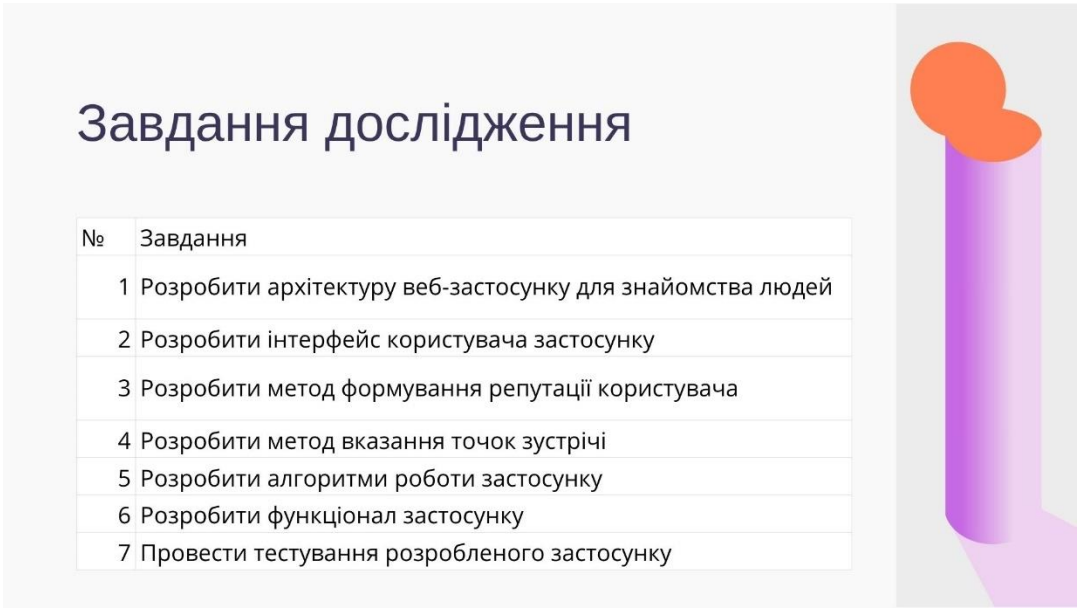

Рисунок Г.5 - слайд 5

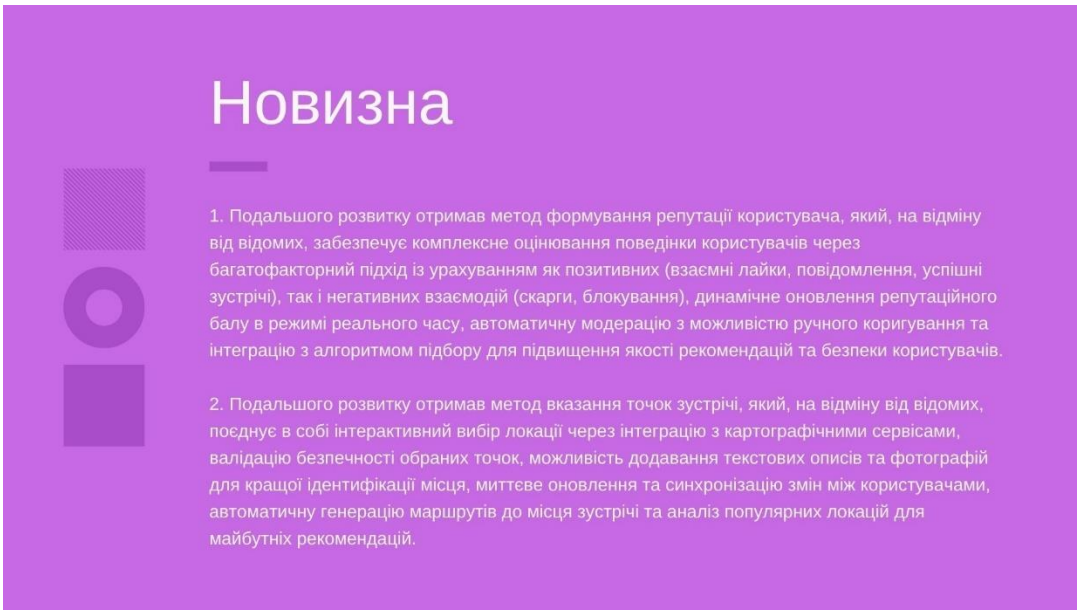

Рисунок Г.6 - слайд 6

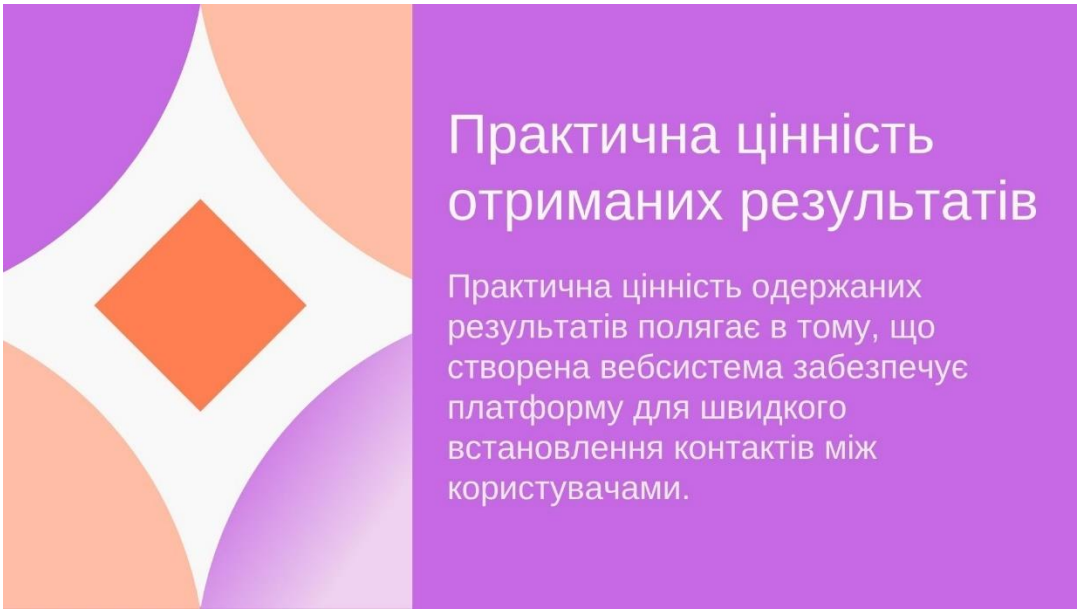

Рисунок Г.7 - слайд 7

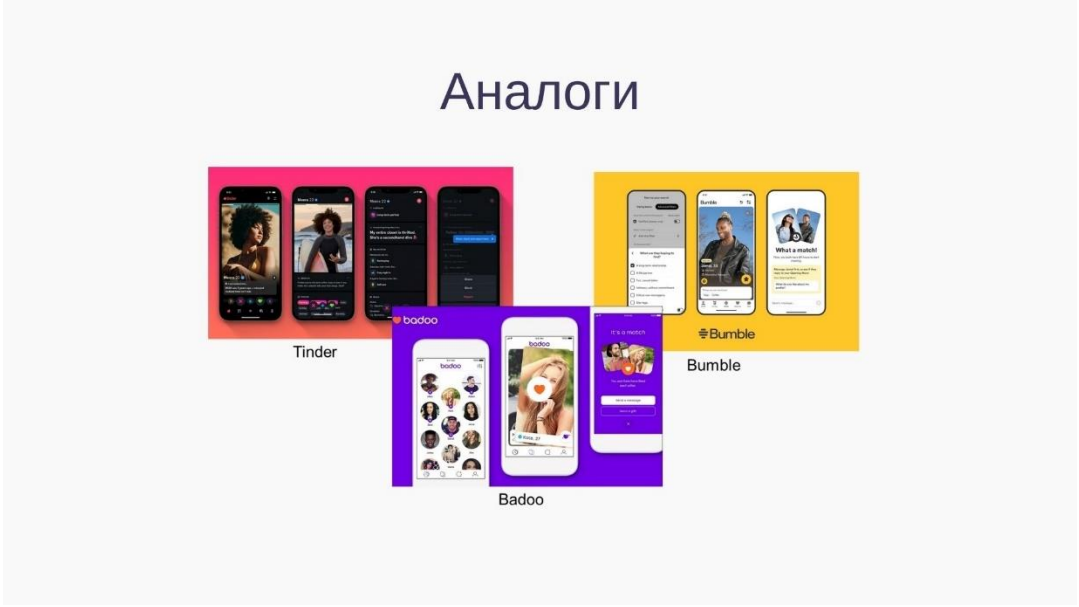

Рисунок Г.8 - слайд 8

Порівняльний аналіз аналогів

Характеристика	Tinder	Badoo	Bumble	Власна розробка
Свайпи	+	+	+	+
Показ активності користувача	+	+	-	+
Система взаємних лайків	+	+	+	+
Можливість відеодзвінків	+	+	+	+
Верифікація	+	+	+	+
Репутація користувача	-	-	-	+
Вказання точок для зустрічі	-	-	-	+
Загальний бал	5	5	4	7

Рисунок Г.9 - слайд 9

Розробка методу формування репутації користувача

1. Збір даних – під час реєстрації фіксуються базові дані (ім'я, вік, фото, місцезнаходження, верифікація), а також подальші дії: повідомлення, лайки, блокування, скарги тощо.
2. KPI – враховуються позитивні (взаємні дії, успішні зустрічі) та негативні взаємодії (скарги, блокування, фейки).
3. Ваги – позитивні дії додають бали (лайк – 2, повідомлення – 3, зустріч – 5), негативні віднімають (скарга – 4, блокування – 5). Верифіковані користувачі отримують бонуси.
4. Розрахунок балу – сума балів за період нормалізується до шкали (0–100), враховується динаміка поведінки.
5. Оновлення в реальному часі – бал оновлюється при кожній взаємодії, періодично аналізується система.
6. Модерація – алгоритми фіксують підозрілу активність і обмежують функції або передають справу модераторам.
7. Використання балу – алгоритм сортує профілі за репутацією, надає бонуси активним користувачам.
8. Зворотний зв'язок – користувач може оскаржити оцінку або подати запит на перегляд.

Рисунок Г.10 - слайд 10

Розробка методу формування репутації користувача

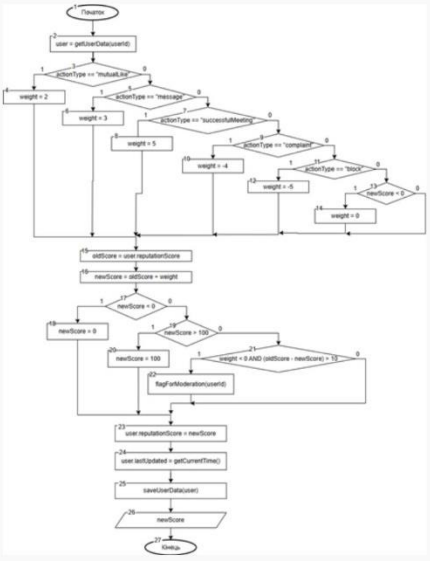

Рисунок Г.11 - слайд 11

Розробка методу вказання точок зустрічі

1. Збір даних – система запитує доступ до геолокації або дозволяє ввести бажане місце зустрічі (місто, район, адреса).
2. Інтеграція з картою – за допомогою Google Maps користувач бачить карту з дорогами, закладами та іншими об'єктами.
3. Вибір точки зустрічі – користувач може натиснути на карту або перетягнути маркер, а також скористатися пошуком по назві місця.
4. Додаткова інформація – можна додати опис локації, фото чи відгуки для кращої ідентифікації.
5. Перевірка та збереження – координати валідуються через картографічний сервіс, а точка зберігається у базі даних.
6. Сповіщення – іншому користувачу надсилається повідомлення з локацією для підтвердження або редагування.

Рисунок Г.12 - слайд 12

Рисунок Г.13 - слайд 13

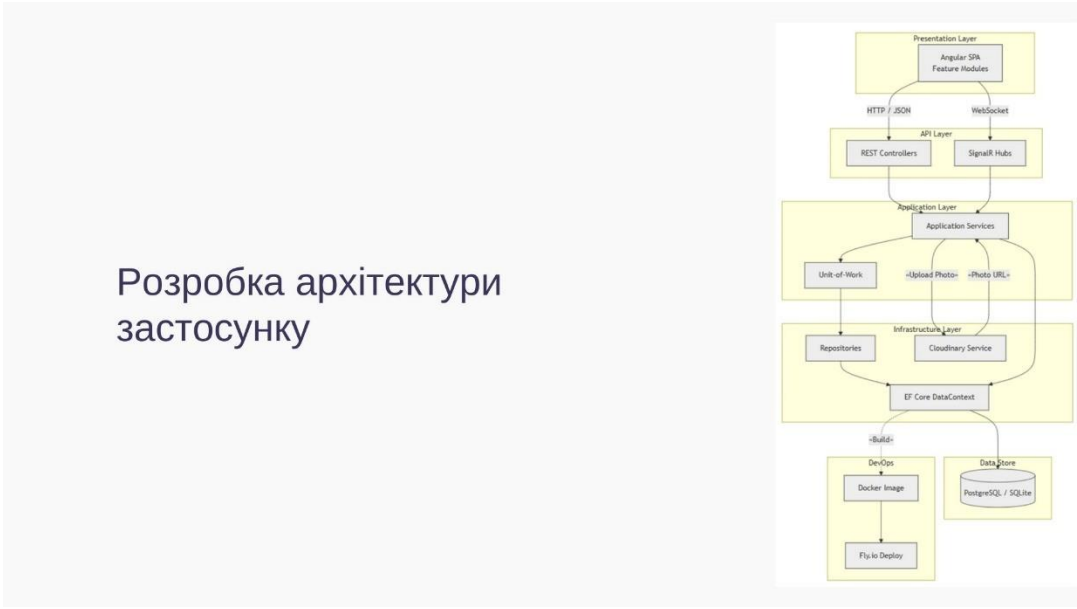

Рисунок Г.14 - слайд 14

Рисунок Г.15 - слайд 15

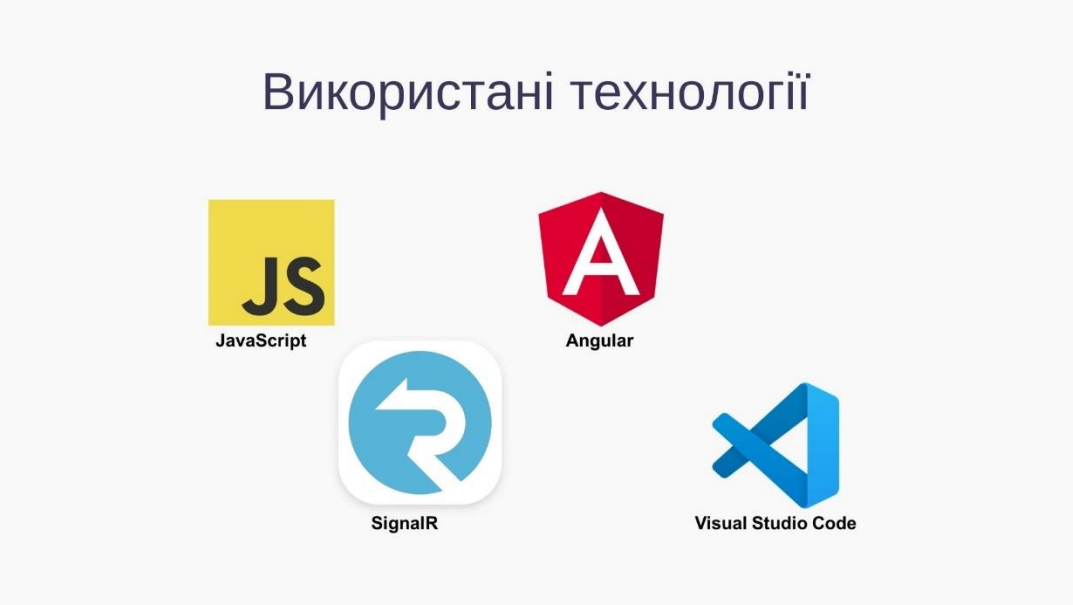

Рисунок Г.16 - слайд 16

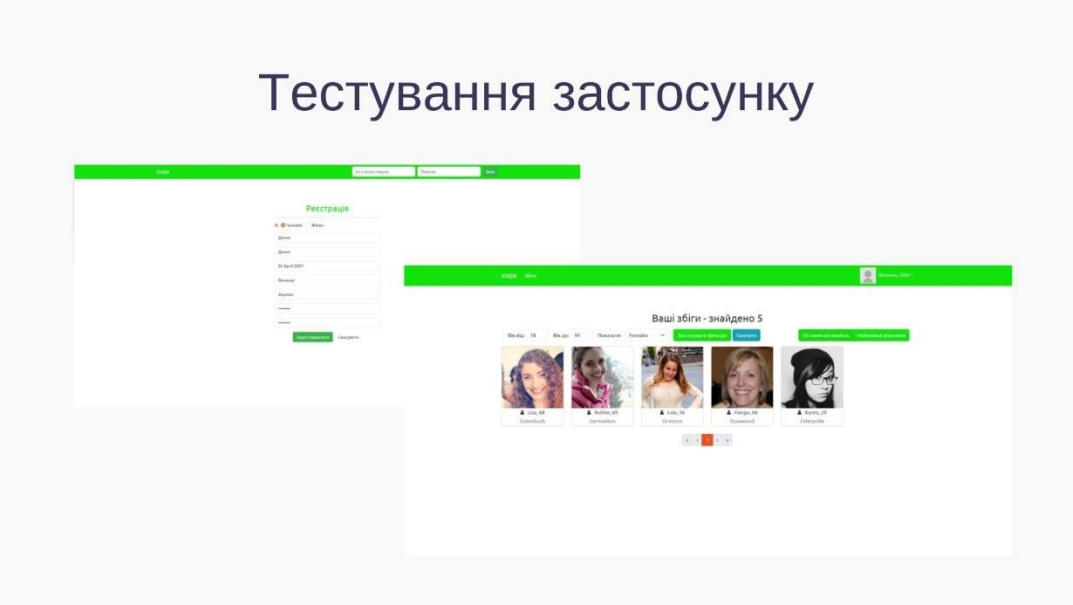

Рисунок Г.17 - слайд 17

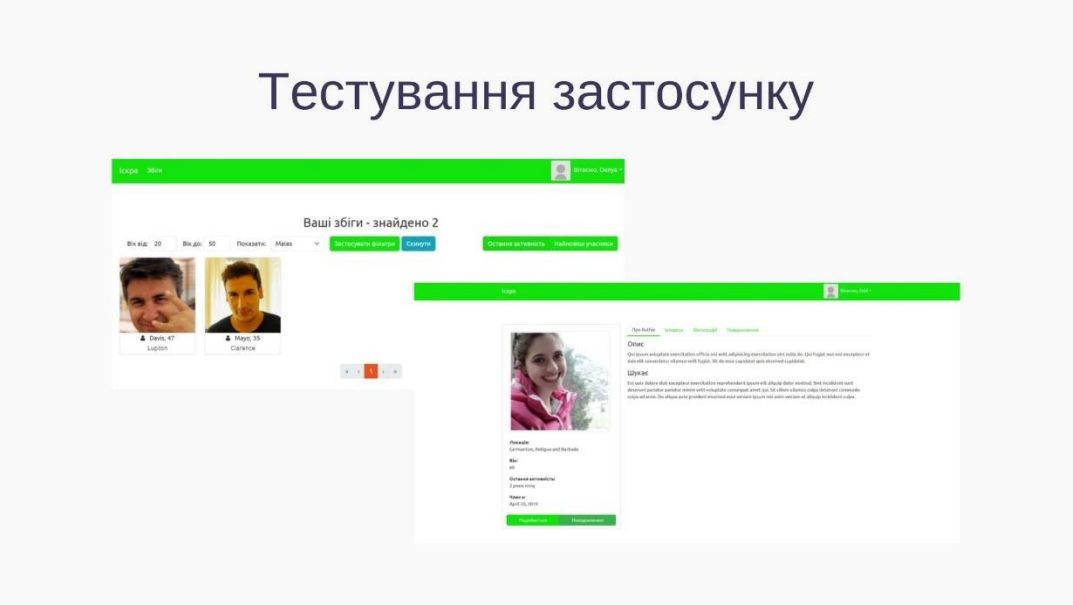

Рисунок Г.18 - слайд 18

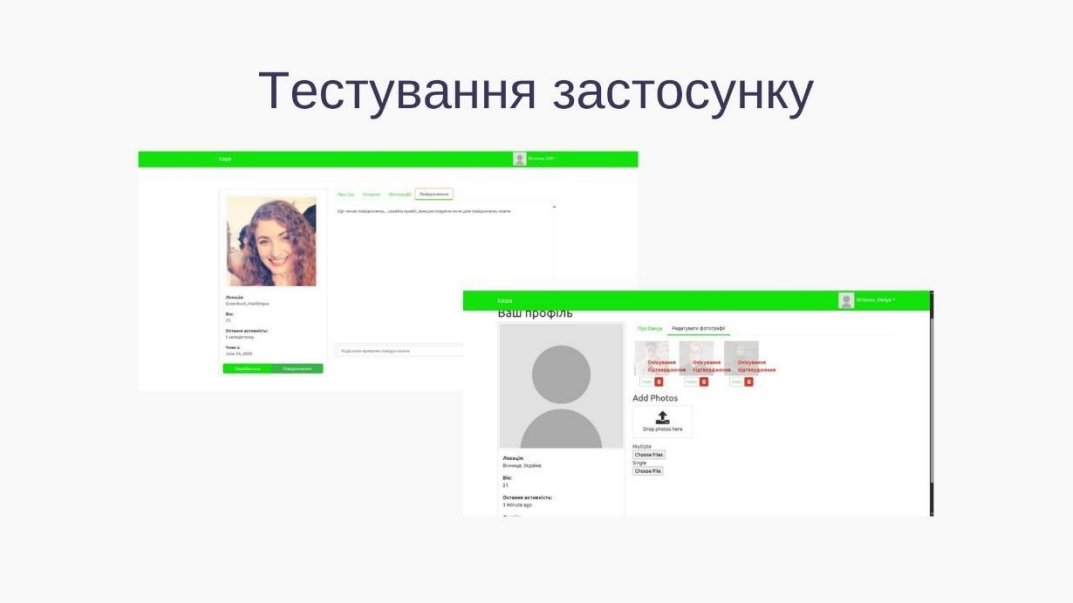

Рисунок Г.19 - слайд 19

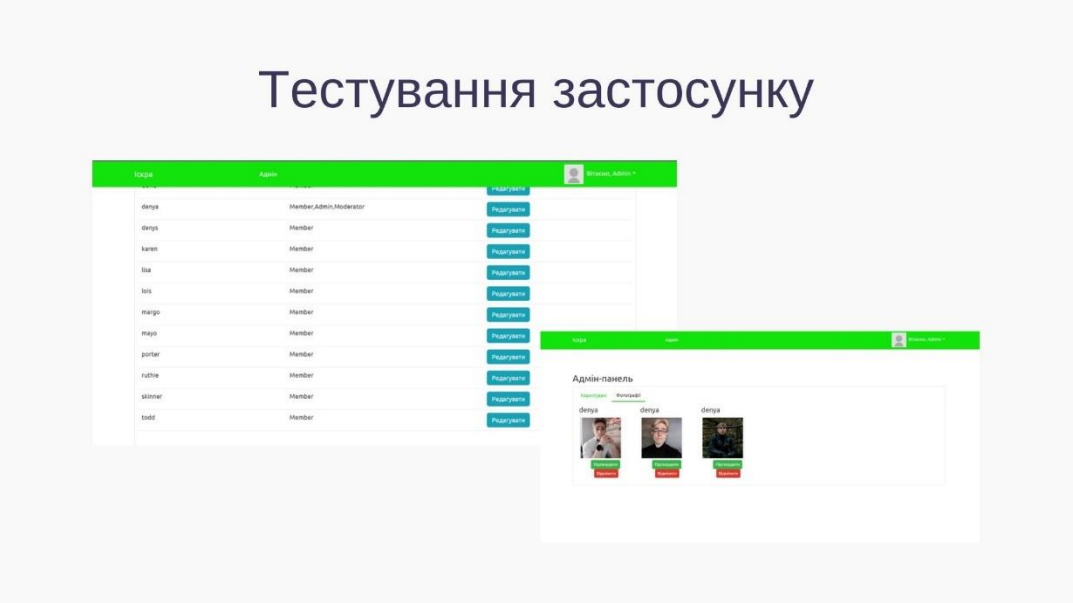

Рисунок Г.20 - слайд 20

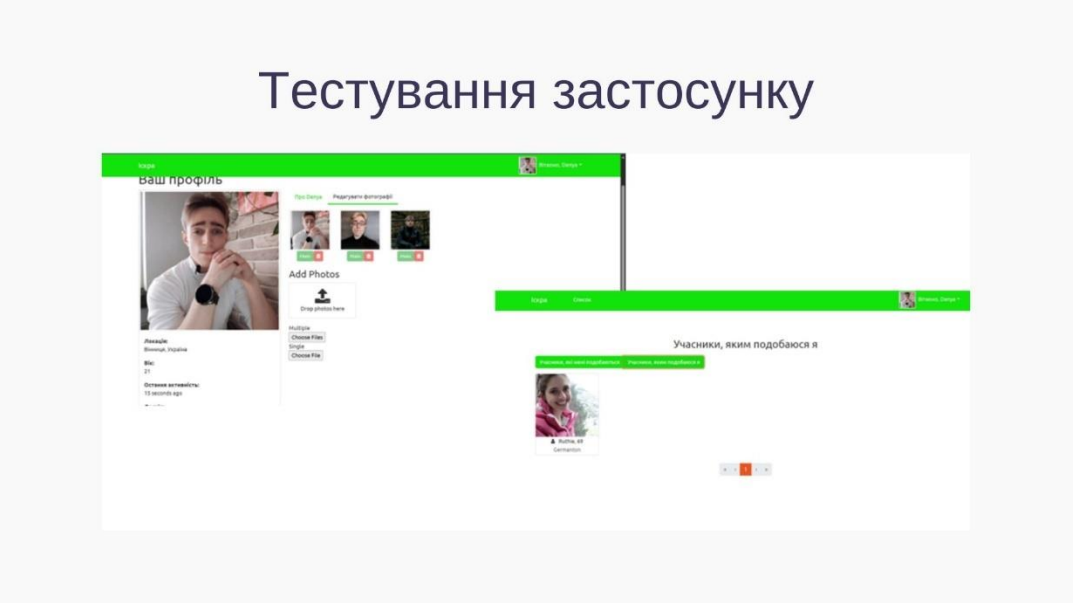

Рисунок Г.21 - слайд 21

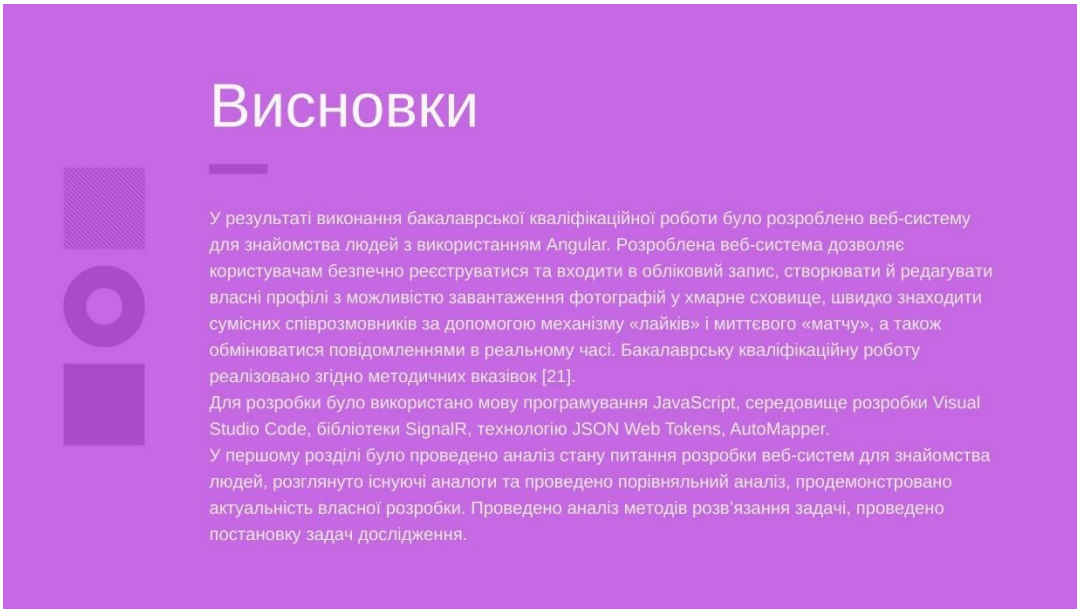

Рисунок Г.22 - слайд 22

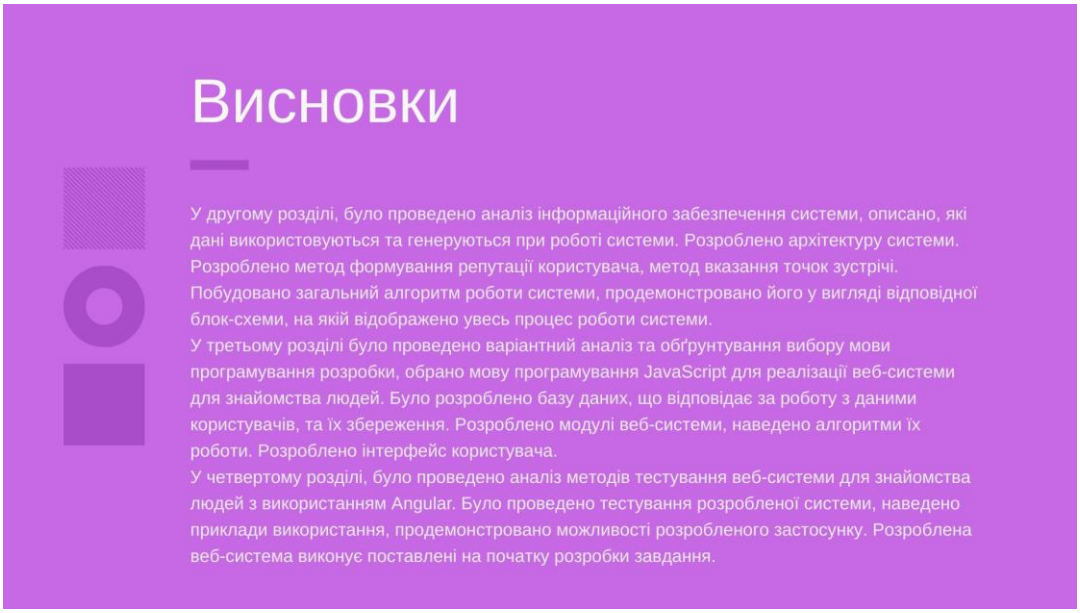

Рисунок Г.23 - слайд 23

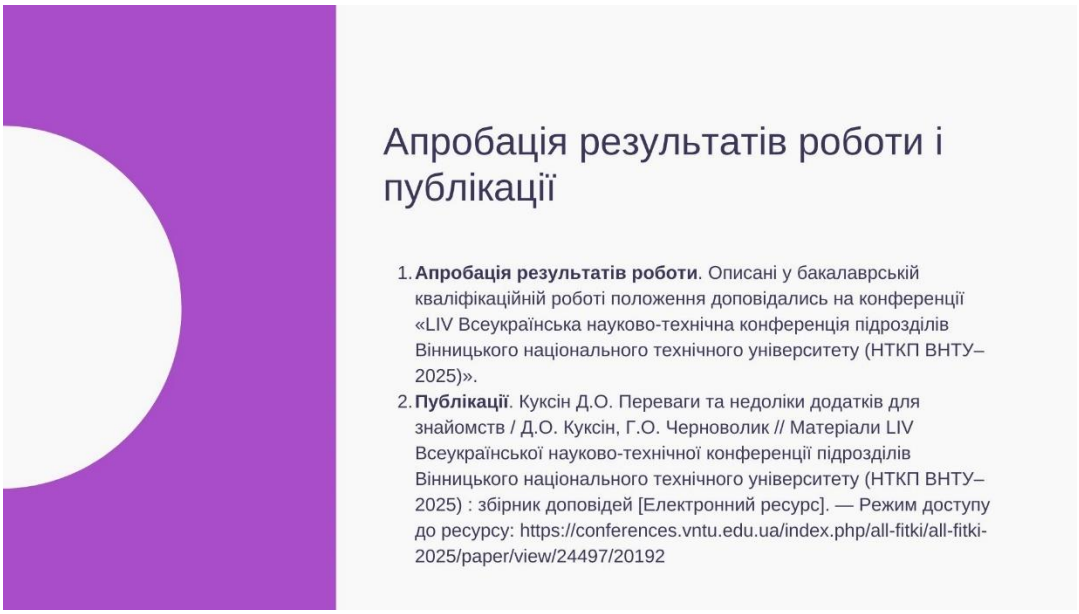

Рисунок Г.24 - слайд 24

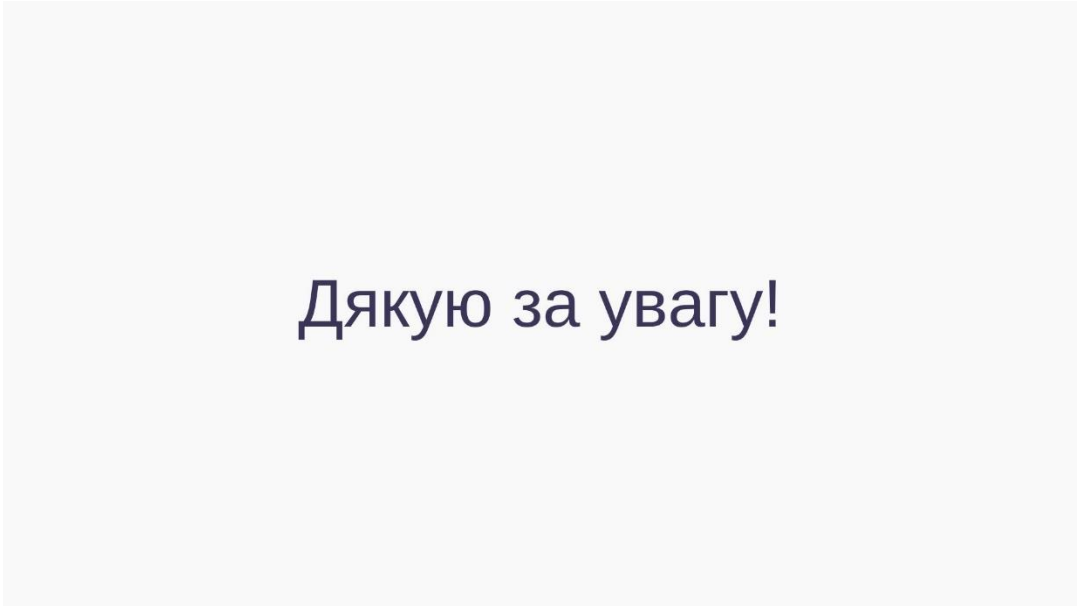

Рисунок Г.25 - слайд 25