

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Веб-сервіс для знайомств»

Виконала студентка групи ЗПЗк-20-1
ВЕНГІР Вікторія Олександрівна

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЛОКТІКОВА Тамара Миколаївна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ВЕНГІР Вікторія Олександрівна**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: «Веб-сервіс для знайомств»,

затверджена Наказом закладу вищої освіти від «14» лютого 2023 р., №71с

Вихідні дані для роботи: предметом і об'єктом досліджень веб-сервісу для знайомств є розробка та взаємодія користувачів з системою для покращення функціональності та користувацького досвіду

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
2	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
3	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів розробки	Примітка
1	Аналіз вимог та розробка концепції	16 лютого 2023 – 28 лютого 2023	Виконано
2	Проектування інтерфейсу та бази даних	1 березня 2023 – 15 березня 2023	Виконано
3	Реалізація функціональності та алгоритмів	16 березня 2023 – 3 квітня 2023	Виконано
4	Тестування та відладка	4 квітня 2023 – 12 квітня 2023	Виконано
5	Документування та підготовка звіту	11 травня 2023 – 2 червня 2023	Виконано
6	Попередній захист роботи	12 травня 2023	Виконано

РЕФЕРАТ

Випускна кваліфікаційна робота присвячена розробці веб-сервісу, який призначений для знайомств та пояснювальної записки. Пояснювальна записка до випускної роботи містить 59 сторінок, 27 ілюстрацій та 7 таблиць.

Метою моєї роботи є розробка веб-сервісу для знайомств, який надає можливість людям знаходити нових друзів, партнерів або романтичні відносини через онлайн-платформу. Основним завданням розроблення веб-додатку є створення зручного і безпечного середовища, в якому користувачі зможуть взаємодіяти, обмінюватись повідомленнями, переглядати профілі і зацікавленості, а також встановлювати зв'язки з іншими користувачами, відповідно до своїх вподобань та цілей.

У роботі я визначила основні завдання розробки веб-проекту, проаналізувала аналоги розробленої системи, визначила основний функціонал та вимоги до системи. Я спроектувала і реалізувала алгоритми роботи системи, а також провела тестування програмного продукту.

КЛЮЧОВІ СЛОВА: ВЕБ-СЕРВІС, РОЗРОБКА, ЗНАЙОМСТВА, ФУНКЦІОНАЛ, КОРИСТУВАЧІ

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Веб-сервіс для знайомств Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		В.О. Венгір					4	59
Керівник		Ю.М. Єфремов						
н. контр.		О.П. Тичина				Житомирська політехніка, група ЗІПЗк-20-1		
Зав. каф.		А.В.Морозов						

ABSTRACT

The final qualification work is devoted to the development of a web service designed for dating and an explanatory note. The explanatory note to the final thesis contains 59 pages, 27 illustrations and 7 tables.

The purpose of my work is to develop a dating web service that allows people to find new friends, partners, or romantic relationships through an online platform. The main task of developing a web application is to create a convenient and secure environment in which users can interact, exchange messages, view profiles and interests, and establish connections with other users, according to their preferences and goals.

In my work, I identified the main tasks of developing a web project, analyzed analogs of the developed system, defined the main functionality and requirements for the system. I designed and implemented the system's algorithms, and tested the software product.

KEYWORDS: WEB SERVICE, DEVELOPMENT, DATING, FUNCTIONALITY, USERS

					ИП3.КР.Б – 121 – 23 – П3	
						5

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ВЕБ-СЕРВІСУ ДЛЯ ЗНАЙОМСТВ	10
1.1 ПОСТАНОВКА ЗАДАЧІ	10
1.2 АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ	11
1.3 ВИБІР АРХІТЕКТУРИ ВЕБ-СЕРВІСУ	16
1.4 ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	20
Висновки до розділу 1.....	24
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ ЗНАЙОМСТВ.....	25
2.1 ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА СТРУКТУРИ СИСТЕМИ.....	25
2.2 РОЗРОБКА БАЗИ ДАНИХ СИСТЕМИ	33
2.3 ОПИС ВЗАЄМОДІЇ ЕЛЕМЕНТІВ СИСТЕМИ ПІД ЧАС РОБОТИ ПРОГРАМНОГО КОМПЛЕКСУ	36
2.4 РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ДЛЯ ЗНАЙОМСТВ	38
Висновки до розділу 2.....	41
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ	43
3.1 ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СИСТЕМИ.	43
3.2 СТРУКТУРА ІНТЕРФЕЙСУ ВЕБ-СЕРВІСУ ДЛЯ ЗНАЙОМСТВ	43
3.3 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ.....	50
Висновки до розділу 3.....	53
ВИСНОВКИ.....	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	57
ДОДАТКИ.....	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

VS – Visual Studio.

ОС – Операційна система.

JS – Javascript.

БД – База даних.

MVC – Model-View-Controller (Модель – Представлення – Контролер) – шаблон проектування.

GUI – Graphic User Interface – графічний інтерфейс користувача.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						7

ВСТУП

Актуальність теми "Веб-сервіс для знайомств" полягає в сучасних реаліях та потребах суспільства. Сьогоднішній світ, що зазнає стрімкого розвитку технологій, вимагає нових способів знаходження партнерів і встановлення взаємовідносин. Люди, які ведуть активний спосіб життя, мають обмежений час для пошуку партнерів, а традиційні способи знайомств часто виявляються неефективними.

Веб-сервіси для знайомств відкривають нові можливості та простір для людей, що шукають взаєморозуміння та створення стійких відносин. Вони надають зручну платформу для знаходження і спілкування з потенційними партнерами, забезпечуючи доступ до широкого спектру користувачів з різних географічних областей, вікових груп, інтересів та цінностей.

Розповсюдження смартфонів та доступ до Інтернету сприяють поширенню веб-сервісів для знайомств, роблячи їх доступними і популярними серед широкого кола користувачів. Це створює потребу в розробці та удосконаленні інноваційних додатків, які забезпечують зручну навігацію, безпеку та персоналізований підхід.

Розробка веб-сервісу для знайомств має велике практичне значення для суспільства. Вона допомагає людям знаходити партнерів, засновуючись на спільних інтересах, цінностях та характеристиках. Такі сервіси сприяють формуванню стійких і гармонійних відносин, а також підтримують соціальну інтеграцію та розвиток особистості.

Метою даної кваліфікаційної роботи є розробка веб-сервісу для знайомств, який надає ефективний та зручний інструмент для знаходження сумісних партнерів, заснований на аналізі користувачів та їхніх вподобань.

Головними завданнями роботи є:

- визначення функціональних вимог до додатку;
- розробка алгоритмів пошуку та рекомендацій;
- тестування та оптимізація роботи системи.

Об'єктом дослідження є процес знайомств і встановлення взаємовідносин в онлайн-середовищі.

Предметом дослідження є розробка веб-сервісу для знайомств, який забезпечує зручну та ефективну платформу для користувачів знаходити потенційних партнерів.

В процесі роботи використовувалися методи аналізу вимог користувачів, статистичний аналіз. Інформаційна база дослідження включала наукові статті, дослідження та релевантні джерела, пов'язані з психологією знайомств.

Результати даної роботи можуть бути використані для розробки та впровадження веб-сервісів знайомств, які надають більш точні та персоналізовані рекомендації користувачам, сприяючи більш успішним знайомствам та встановленню взаємовідносин.

Пояснювальна записка до кваліфікаційної роботи містить 59 сторінок, 27 ілюстрацій та 7 таблиць, список літературних джерел містить 30 найменувань.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ВЕБ-СЕРВІСУ ДЛЯ ЗНАЙОМСТВ

1.1 Постановка задачі

Метою роботи є розробка програмного комплексу - веб-сервісу для знайомств. У даному розділі буде розглянуто постановку задач, які потрібно вирішити під час опрацювання роботи та реалізації веб-сервісу.

Метою дослідження є розробка веб-сервісу для знайомств, який надасть користувачам можливість знайти і спілкуватися з новими людьми в Інтернеті. Даний сервіс повинен забезпечувати зручну та безпечну платформу для знайомств, де користувачі зможуть створювати профілі, переглядати інші профілі, взаємодіяти з іншими користувачами та встановлювати зв'язок з потенційними партнерами.

Для досягнення поставленої мети необхідно вирішити поетапно наступні задачі:

Етап 1: Аналіз та проектування

Аналіз та вивчення вимог:

- Визначити потреби та очікування користувачів щодо функціоналу веб-сервісу;
- Встановити основні функції та можливості, які повинен мати веб-сервіс.

Проектування архітектури системи:

- Визначити компоненти клієнтської та серверної частини системи;
- Розробити схему взаємодії між компонентами системи;
- Встановити структуру бази даних та засоби зберігання інформації.

Проектування інтерфейсу користувача:

- Розробити макети та прототипи інтерфейсу;
- Визначити структуру та зовнішній вигляд сторінок та елементів інтерфейсу.

Етап 2: Реалізація

Розробка серверної частини:

- Реалізувати механізми авторизації та аутентифікації користувачів;
- Забезпечити безпеку та конфіденційність даних.

Розробка функціоналу створення та редагування профілів:

- Реалізувати можливість створення нового профілю користувача;
- Розробити функціонал редагування та оновлення інформації у профілі.

Розробка пошукової системи:

- Реалізувати механізм пошуку користувачів за різними критеріями;
- Розробити алгоритми та фільтри для відповідності результатів пошуку запиту користувача.

Розробка клієнтської частини:

- Реалізувати інтерфейс користувача згідно з макетами та прототипами;
- Забезпечити зручну навігацію та взаємодію з системою.

Етап 3: Тестування та оптимізація

Тестування програмного комплексу:

- Виконати тести для виявлення та виправлення помилок;
- Перевірити функціональність та коректність роботи системи.

Оптимізація та удосконалення:

- Проаналізувати продуктивність та швидкість роботи системи;
- Впровадити оптимізації для підвищення ефективності роботи системи.

1.3 Аналіз аналогів програмного продукту

У даному розділі буде проведено аналіз існуючих рішень на ринку веб-сервісів для знайомств. Кожне з цих рішень буде описано з визначенням їх основних функціональних можливостей. На основі проведеного аналізу буде

зроблено висновок щодо функціональних можливостей систем даного типу.

Розглянемо 3 найпопулярніших додатки для пошуку пари:

1. Tinder



Рис. 1.1 - Tinder

Основні функціональні можливості:

- Реєстрація користувачів та створення профілів;
- Пошук та фільтрація користувачів за різними критеріями (вік, місцезнаходження, інтереси тощо);
- Функція "Swipe" для вибору потенційних партнерів;
- Відправлення повідомлень між співпрацюючими користувачами;
- Організація спільних подій та зустрічей.

Висновок:

Tinder [4] (рис. 1.1) надає можливість швидкого знайомства з людьми на основі спільних інтересів та взаємного інтересу. Він пропонує широкий функціонал для пошуку, спілкування та організації зустрічей. Інтерфейс доадка

2. Bumble

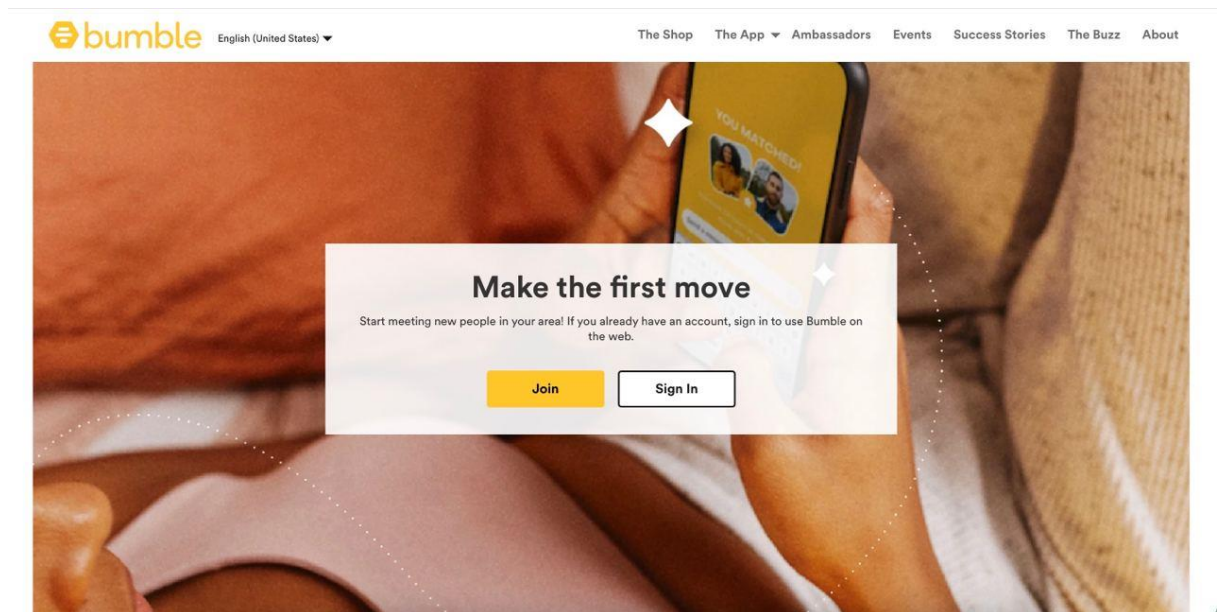


Рис. 1.2 - Bumble

Основні функціональні можливості:

- Реєстрація користувачів та створення профілів;
- Жінки мають перевагу над початком спілкування;
- Пошук та фільтрація користувачів за різними критеріями;
- Відправлення повідомлень між співпрацюючими користувачами;
- Можливість організації подій та активностей.

Висновок:

Bumble [2] (рис 1.2) надає жінкам більшу контроль над процесом спілкування та знайомства. Він пропонує різноманітні функціональні можливості для пошуку, спілкування та організації подій.

3. OkCupid

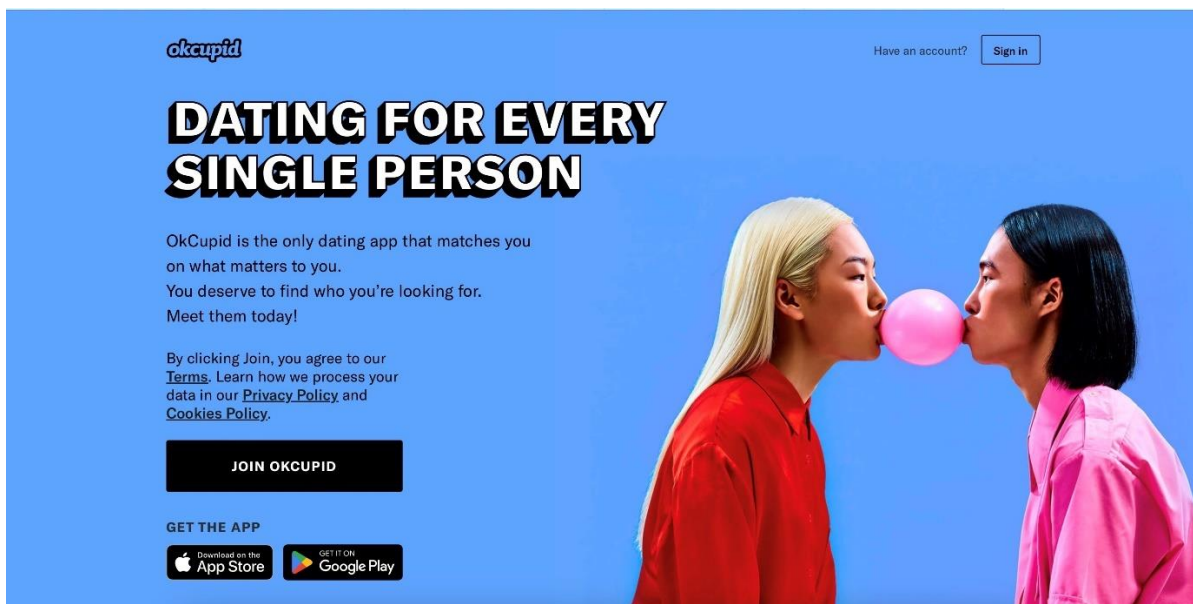


Рис. 1.3 - OkCupid

Основні функціональні можливості:

- Реєстрація користувачів та створення профілів;
- Пошук та фільтрація користувачів за різними параметрами;
- Відправлення повідомлень між співпрацюючими користувачами;
- Відображення відсотку відповідності між користувачами на основі запитань.

Висновок:

OkCupid [3] (рис. 1.3) пропонує систему відповідності на основі запитань, яка допомагає знайти партнера зі схожими інтересами. Він надає базові функції для пошуку та спілкування.

Тепер порівнюємо функціональні можливості (табл. 1.1), інноваційність (табл. 1.2), доступність (табл. 1.3), платіжні моделі (табл. 1.4) та захист особистих даних (табл. 1.5) кожного з наведених вище додатків.

Таблиця 1.1

Порівняння функціональних можливостей додатків для знайомств

Функціональні можливості	Tinder	Bumble	OkCupid
Реєстрація користувачів та створення профілів	+	+	+
Пошук та фільтрація користувачів	+	+	+

Продовження таблиці 1.1

Система "Swipe" для вибору партнерів	+	-	-
Відправлення повідомлень	+	+	+
Організація подій та зустрічей	+	+	-
Жінки мають перевагу в початку спілкування	-	+	-
Система відповідності на основі запитань	-	-	+

Таблиця 1.2

Порівняння інновацій та відмінностей додатків для знайомств

Інновації та відмінності	Tinder	Bumble	OkCupid
Система "Swipe"	Швидке вибирання потенційних партнерів	Жінки мають перевагу у спілкуванні	-
Перевага жінок	-	Жінки мають перевагу в спілкуванні	-
Система відповідності	-	-	Відображення відсотку відповідності

Таблиця 1.3

Порівняння доступності платформ для додатків для знайомств

Доступність платформ	Tinder	Bumble	OkCupid
Веб-версія	-	-	+
Android	+	+	+
iOS	+	+	+

Таблиця 1.4

Порівняння платіжних моделей додатків для знайомств

Платіжні моделі	Tinder	Bumble	OkCupid
Безкоштовний	Обмежений доступ	Обмежений доступ	Так
Підписка	Так	Так	Так
Додаткові послуги	Так	Так	Так

Порівняння захисту особистих даних у додатках для знайомств

Захист особистих даних	Tinder	Bumble	OkCupid
Анонімність	-	+	+
Керування приватністю	-	+	+
Захист від шахрайства	+	+	+

Висновок:

Існуючі рішення на ринку веб-сервісів для знайомств надають базовий функціонал, включаючи реєстрацію, пошук користувачів та спілкування. Кожне з них також пропонує унікальні функції, такі як система "Swipe" в Tinder, перевага жінок у Bumble та система відповідності на основі запитань у OkCupid. Наш веб-сервіс може врахувати ці особливості та додати нововведення для створення привабливої та інноваційної платформи для знайомств.

1.3 Вибір архітектури веб-сервісу

В процесі розгляду різних архітектурних патернів, було визначено основні характеристики та переваги кожного патерну. Нижче наведений стислий порівняльний огляд таких патернів:

1. Клієнт-серверна архітектура (рис. 1.4).

- Клієнтські пристрої взаємодіють з сервером для отримання та передачі даних;
- Забезпечує централізовану обробку та зберігання даних на сервері;
- Забезпечує розподілені обчислення та доступ до додатку з різних пристроїв [1].

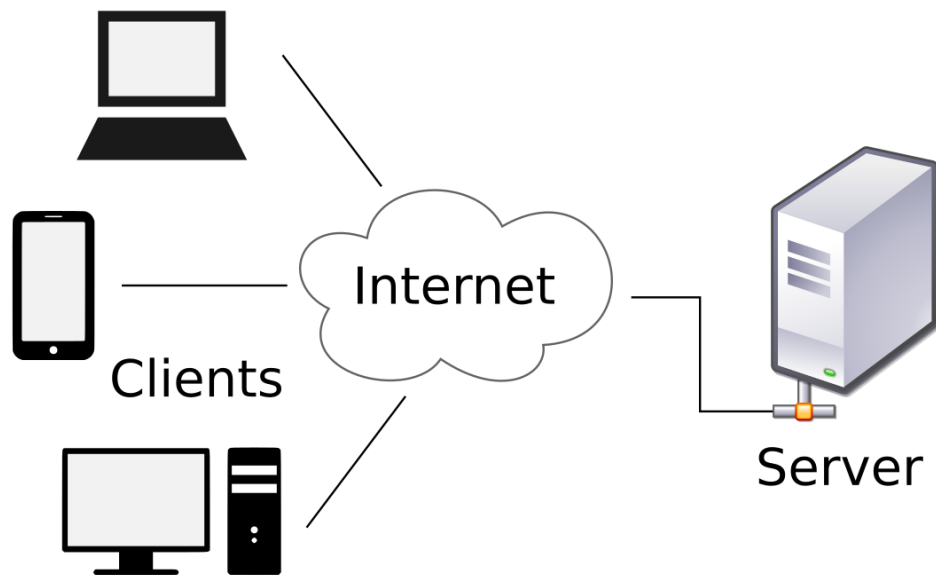


Рис. 1.4 - Схематичне зображення архітектури

2. MVC (Model-View-Controller) (рис. 1.5).

Опис архітектури MVC:

- Модель (Model): Модель представляє бізнес-логіку та дані додатку. Вона відповідає за збереження, обробку та маніпуляцію даними. Модель може включати класи, які описують структуру даних, логіку бізнес-процесів та доступ до бази даних. Модель не залежить від решти компонентів системи і може використовуватися в різних контекстах.
- Представлення (View): Представлення відповідає за відображення даних користувачу та забезпечує інтерфейс для взаємодії з користувачем. Воно може бути представлене у вигляді веб-сторінок, шаблонів або компонентів. Представлення отримує дані від моделі та відображає їх у відповідному форматі для користувача. Воно не містить логіки та не зберігає стан даних.
- Контролер (Controller): Контролер взаємодіє з користувачем та керує потоком даних між моделлю та представленням. Він обробляє вхідні дані від користувача, актуалізує модель та вибирає відповідне представлення для відображення даних. Контролер відповідає за координацію дій та управління взаємодією між моделлю та представленням [5-7].

Однією з переваг архітектури MVC є розділення відповідальностей між компонентами, що сприяє полегшенню розробки, тестуванню та підтримці додатку. Крім того, ця архітектура дозволяє легко замінювати або модифікувати окремі компоненти без впливу на інші.

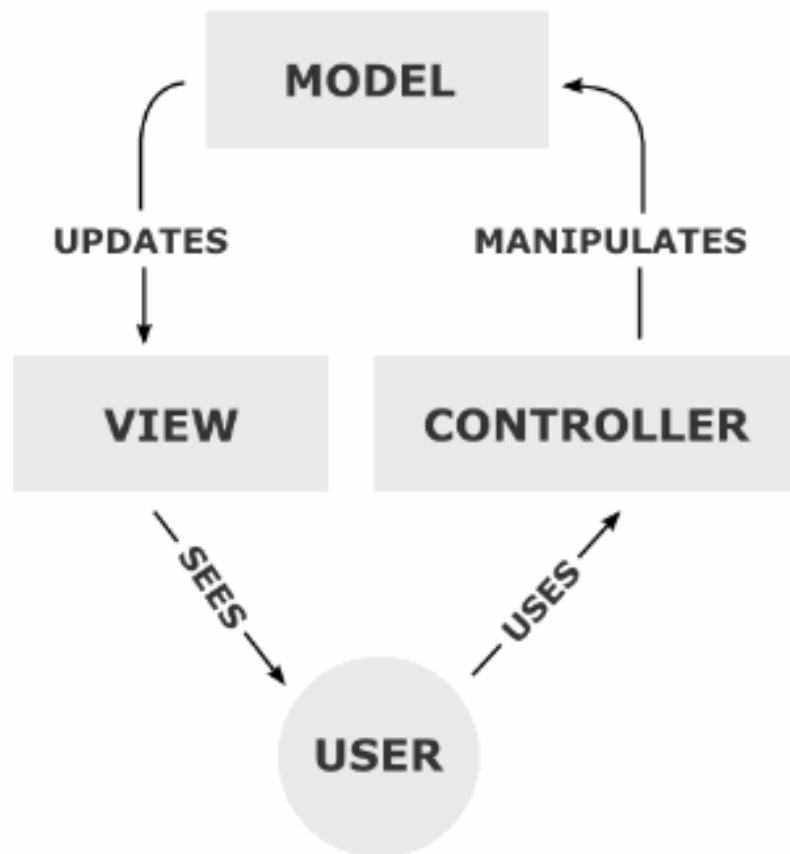


Рис. 1.5 - Схематичне зображення архітектури MVC

3. Microservices (Мікросервісна архітектура) (рис. 1.6).

Опис мікросервісної архітектури:

Розбиття на мікросервіси: додаток розбивається на набір мікросервісів, кожен з яких відповідає за конкретну функціональність. Наприклад, у веб-сервісі для знайомств можуть бути окремі мікросервіси для управління користувачами, обробки запитів на знайомства, повідомлень та інших функціональних частин.

Автономність мікросервісів: кожен мікросервіс може бути розроблений та випущений незалежно від інших мікросервісів. Це дозволяє розвивати та масштабувати окремі компоненти без впливу на решту системи. Кожен

мікросервіс може мати свою власну базу даних, логіку бізнес-процесів та інші ресурси.

Комунікація між мікросервісами: мікросервіси взаємодіють між собою за допомогою механізмів комунікації, таких як HTTP, REST або повідомлення. Кожен мікросервіс може мати своє API, яке дозволяє іншим мікросервісам виконувати запити та обмінюватися даними.

Масштабованість: мікросервісна архітектура дозволяє масштабувати окремі мікросервіси в залежності від навантаження. Наприклад, якщо окремий мікросервіс отримує велику кількість запитів, його можна масштабувати горизонтально, додавши додаткові інстанси цього мікросервісу.

Мікросервісна архітектура виявляється особливо корисною в ситуаціях, коли маємо справу зі складними та розширюваними додатками. Вона дозволяє розбити систему на більш менш незалежні компоненти, які можуть бути розвивані та масштабовані окремо. Крім того, мікросервісна архітектура сприяє зручності в управлінні додатком, підтримці, розгортанні та розширенні [8-11].

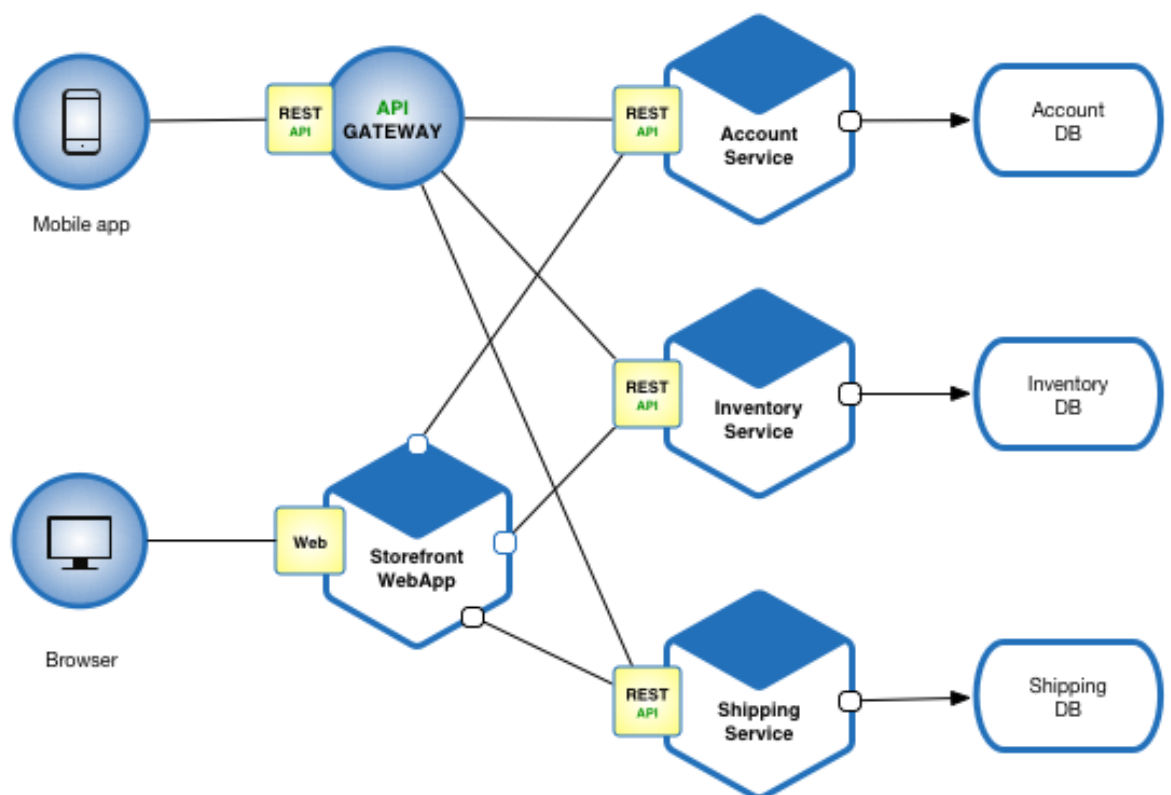


Рис. 1.6 - Схематичне зображення мікросервісної архітектури

Один з ключових аспектів вибору архітектури - це вибір технологій для реалізації клієнтської та серверної частин. У нашому випадку ми обрали наступну конфігурацію:

1. Клієнтська частина (Frontend):

Для реалізації клієнтської частини ми вибрали фреймворк Angular. Angular є потужним фреймворком для побудови веб-інтерфейсів, який надає широкі можливості для створення інтерактивних та динамічних додатків. За допомогою Angular ми зможемо побудувати модульний та масштабований фронтенд для нашого веб-сервісу для знайомств.

2. Серверна частина (Backend):

Для реалізації серверної частини ми обрали технологію .NET Web API. .NET Web API - це фреймворк для створення веб-сервісів з використанням платформи .NET. Він надає зручні інструменти для реалізації RESTful API, обробки запитів та доступу до бази даних. .NET Web API дозволить нам створити потужний та ефективний бекенд для нашого веб-сервісу.

Така клієнт-серверна архітектура з використанням Angular для клієнтської частини та .NET Web API для серверної частини надає нам широкі можливості для розробки та масштабування нашого веб-сервісу для знайомств. Комунікація між клієнтом та сервером буде здійснюватися за допомогою протоколу HTTP, дозволяючи обмінюватися даними та виконувати різні операції.

Таким чином, наша обрана клієнт-серверна архітектура забезпечить нам потужну та ефективну реалізацію веб-сервісу для знайомств, де клієнтська частина на базі Angular взаємодіє з серверною частиною на базі .NET Web API, що дозволить забезпечити високу швидкість роботи, масштабованість та надійність системи.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Оберемо інструментальні засоби для розробки веб-сервісу:

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						20

.NET Web API - це потужний фреймворк для створення веб-сервісів з використанням HTTP протоколу. Я обрала .NET Web API для реалізації серверної частини нашого веб-сервісу для знайомств з такими обґрунтуваннями:

- .NET Web API забезпечує легку розширюваність та розгортання веб-сервісу. Він інтегрується зі звичайними функціональними можливостями .NET Framework, що дозволяє ефективно створювати API та обробляти HTTP запити.
- .NET Web API має підтримку стандартних протоколів та форматів, таких як HTTP, JSON та XML. Це дозволяє нам легко обмінюватися даними з клієнтською частиною веб-сервісу.
- .NET Web API надає можливість використовувати архітектурний патерн Model-View-Controller (MVC), що сприяє розділенню логіки, даних та представлення. Це спрощує розробку та підтримку коду, забезпечуючи гнучкість та розширюваність [12-14].

Angular - це потужний фреймворк для розробки клієнтської частини веб-додатків. Я обрала Angular для фронтенд частини нашого веб-сервісу для знайомств з такими обґрунтуваннями:

- Angular надає широкі можливості для створення динамічного та інтерактивного інтерфейсу користувача. Він пропонує розширені функції шаблонізації, зв'язування даних та маніпуляції DOM елементами, що дозволяє створювати багатофункціональні та привабливі веб-сторінки.
- Angular використовує компонентну архітектуру, що сприяє розділенню логіки та покращує перевикористання коду. Компоненти Angular можуть бути незалежними та повторно використовуваними елементами, що спрощує розробку та підтримку коду.
- Angular надає потужні інструменти для роботи зі сторонніми бібліотеками та модулями, що дозволяє ефективно інтегрувати сторонні компоненти та функціональність в наш веб-сервіс [15-16].

Visual Studio та **Visual Studio Code** - це популярні середовища розробки для .NET та Angular відповідно. Я обрала ці середовища з такими обґрунтуваннями:

- Visual Studio є повнофункціональним інтегрованим середовищем розробки, яке надає широкі можливості для розробки, налагодження та тестування коду .NET Web API. Воно надає багато інструментів, таких як редактор коду, візуальні конструктори та засоби аналізу коду, що сприяють ефективному розроблянню.
- Visual Studio Code є легким та розширюваним середовищем розробки, яке підтримує роботу з Angular. Воно надає функціональний редактор коду, вбудовану підтримку Git, розширюваність через плагіни та інші корисні функції, що спрощують розробку клієнтської частини [17-18, 30].

SQLite - це легка та ефективна база даних, яку я обрала для нашого веб-сервісу для знайомств з такими обґрунтуваннями:

- SQLite пропонує вбудовану базу даних, яка знаходиться в одному файлі, що спрощує розгортання та управління базою даних. Використання SQLite не потребує налаштування окремого сервера баз даних, що зручно для невеликих проектів.
- SQLite підтримує стандартні SQL запити та має багато корисних функцій, що дозволяє нам ефективно працювати з даними в нашому веб-сервісі [19-21].

SignalR - це бібліотека, яку я обрала для реалізації функціональності чату в нашому веб-сервісі для знайомств з такими обґрунтуваннями:

- SignalR надає можливість реалізувати двосторонню комунікацію в реальному часі між клієнтом та сервером. Використання SignalR дозволить нам створити чатову функціональність, де користувачі зможуть обмінюватися повідомленнями в реальному часі.

- SignalR має вбудовані механізми виявлення та автоматичного підтримування з'єднання між клієнтом та сервером. Це забезпечує стабільну та надійну комунікацію між користувачами в чаті [22].

Bootstrap - це потужний CSS фреймворк, який я обрала для фронтенд частини нашого веб-сервісу для знайомств з такими обґрунтуваннями:

- Bootstrap надає готові компоненти та стилізацію, що дозволяє швидко та ефективно розробляти користувацький інтерфейс. Використання Bootstrap допоможе нам створити привабливий та адаптивний дизайн для нашого веб-сервісу.
- Bootstrap пропонує респонсивний дизайн, що дозволяє нашому веб-сервісу відповідати на різні розміри екранів та пристроїв. Це забезпечить зручний та приємний досвід користувачів, незалежно від того, який пристрій вони використовують [23-24].

Microsoft Identity - це система реєстрації та авторизації, яку я обрала для нашого веб-сервісу для знайомств з такими обґрунтуваннями:

- Microsoft Identity надає готові компоненти та функціональність для реєстрації, аутентифікації та авторизації користувачів. Використання Microsoft Identity дозволить нам безпечно та ефективно управляти доступом до ресурсів нашого веб-сервісу.
- Microsoft Identity підтримує різні методи аутентифікації, включаючи вхід через обліковий запис Microsoft, соціальні мережі та інші методи. Це забезпечить зручну та гнучку систему аутентифікації для наших користувачів [25].

MailKit - це бібліотека для відправки електронних листів, яку я обрала для розсилки емейлів в нашому веб-сервісі для знайомств з такими обґрунтуваннями:

- MailKit надає потужні та розширені функції для взаємодії з SMTP серверами та відправки електронних листів. Використання MailKit дозволить нам надійно та ефективно відправляти електронні повідомлення нашим користувачам.

- MailKit підтримує різні формати електронних листів, включаючи HTML та текстовий формат, а також можливість додавання вкладень до листів. Це дозволить нам створювати професійні та інформативні електронні повідомлення для наших користувачів [26].

Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено детальне дослідження та аналіз функціональних вимог, архітектурних патернів та інструментальних засобів для розробки веб-сервісу для знайомств.

В рамках аналізу предметної області було визначено основні функціональні можливості, які має мати веб-сервіс для знайомств, такі як створення профілів користувачів, пошук та фільтрація, спілкування та інші. Були оглянуті існуючі на ринку аналоги, з'ясовано їхні переваги та недоліки, а також виявлено прогалини у функціональних можливостях, які можуть бути враховані при розробці нового веб-сервісу.

З метою вибору оптимальної архітектури були розглянуті різні патерни, зокрема клієнт-серверна архітектура та Model-View-Controller (MVC). Після аналізу було обрано клієнт-серверну архітектуру, оскільки вона забезпечує розділення логіки на клієнтську та серверну частини, забезпечує масштабованість та підтримку зручної взаємодії.

Обґрунтовано вибір інструментальних засобів, таких як .NET Web API та Angular для розробки веб-сервісу, Visual Studio та Visual Studio Code для розробки в середовищі, SQLite як база даних, SignalR для реалізації чату, Bootstrap компоненти для фронтенду, Microsoft Identity для реєстрації та авторизації, і MailKit для розсилки електронних листів.

Загальний аналіз і обґрунтування вибору показали, що запропоновані технології та архітектурні рішення мають потенціал задовольнити функціональні вимоги та очікування користувачів веб-сервісу для знайомств.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ ЗНАЙОМСТВ

2.1 Визначення варіантів використання та структури системи.

Веб-сервіс створюється для надання платформи, що дозволяє людям знайти нових партнерів або друзів шляхом онлайн-знайомств.

Вимоги користувачів.

Користувачам надається можливість скористатися наступними функціями:

1. Пошук людей;
2. Авторизація;
3. Редагування інформації;
4. Перегляд профілів користувачів;
5. Перегляд списків вподобань.

Адміністраторам надається можливість:

1. Редагування ролі користувача;
2. Авторизація адміністратора;
3. Зміна ролей користувачів.

Варіанти використання веб-сервісу зображені на рис. 2.1.

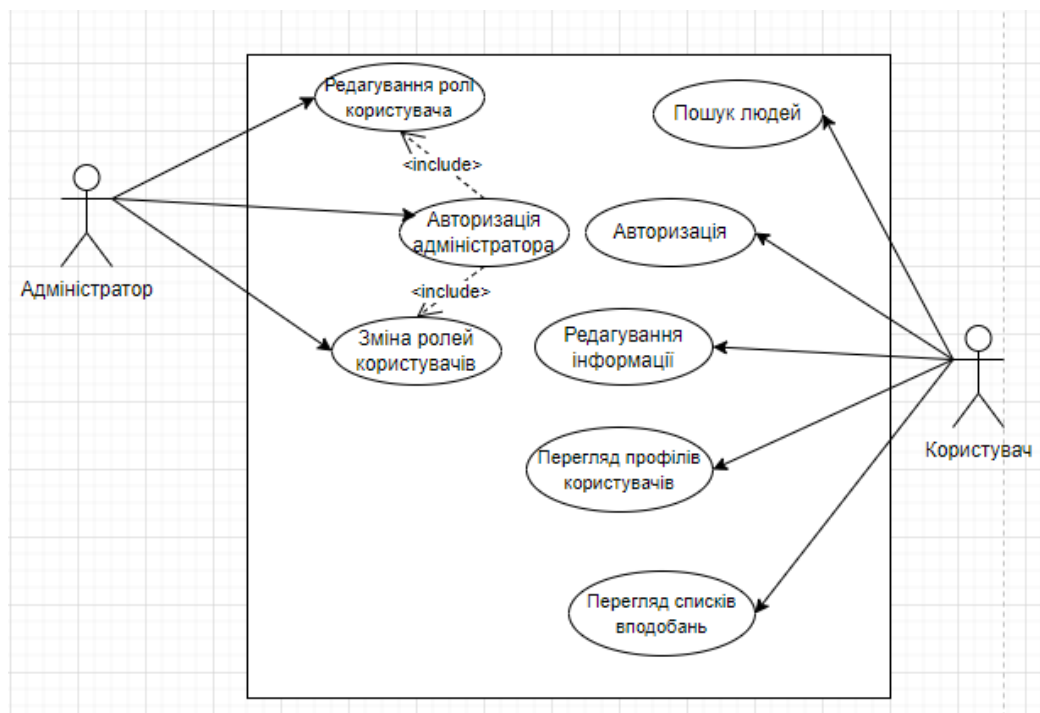


Рис. 2.1 - Варіанти використання

Функціональні вимоги:

Реєстрація та авторизація:

1. Реєстрація нового користувача - користувач має мати можливість створити обліковий запис, вказавши необхідну інформацію, таку як ім'я, електронну адресу та пароль; система повинна перевіряти унікальність електронної адреси та надавати повідомлення про успішну реєстрацію.
2. Авторизація користувача - користувач має мати можливість ввести свої облікові дані (електронна адреса та пароль) для отримання доступу до свого облікового запису; система повинна перевіряти коректність введених даних та надавати доступ до функціональності системи в разі успішної авторизації.

Створення профілю:

1. Заповнення основних даних - користувач має мати можливість вказати свої основні дані, такі як вік, стать, місце проживання, інтереси та інші відомості, які допоможуть знайти відповідних партнерів для знайомств; система повинна забезпечувати коректне збереження та відображення цих даних в профілі користувача.
2. Додавання фотографій - користувач має мати можливість завантажувати фотографії та додавати їх до свого профілю; система повинна забезпечувати збереження та відображення фотографій в профілі користувача.

Пошук профілів:

1. Пошук за параметрами - користувач має мати можливість здійснювати пошук інших профілів за різними параметрами.

Відправлення повідомлень:

1. Надсилання повідомлень - користувач має мати можливість надсилати повідомлення іншим користувачам; система повинна забезпечувати надійну доставку повідомлень та їх відображення відповідним користувачам.

2. Отримання повідомлень - користувач має мати можливість отримувати повідомлення від інших користувачів; система повинна забезпечувати швидке та точне отримання повідомлень та їх відображення користувачеві.

Онлайн-чат:

1. Запуск онлайн-чату - користувач має мати можливість запустити онлайн-чат з іншими користувачами, які перебувають в мережі; система повинна забезпечувати швидке та стабільне підключення до чату та взаємодію з іншими користувачами.
2. Відправлення повідомлень в чаті - користувач має мати можливість надсилати повідомлення в онлайн-чаті; система повинна забезпечувати швидку та точну доставку повідомлень у чаті.

Налаштування облікового запису:

1. Зміна основних даних - користувач має мати можливість змінювати свої основні дані, такі як вік, стать, місце проживання та інші відомості; система повинна забезпечувати коректне збереження та відображення оновлених даних у профілі користувача.
2. Зміна паролю - користувач має мати можливість змінювати свій пароль для забезпечення безпеки облікового запису; система повинна забезпечувати надійну зміну паролю та збереження його в зашифрованому вигляді.

Нефункціональні вимоги веб-сервісу для знайомств включають такі аспекти:

1. Ефективність: Система повинна працювати швидко і забезпечувати мінімальний час відповіді на запити користувачів.
2. Надійність: Сервіс повинен бути стабільним і надійним, здатним опрацьовувати велику кількість одночасних запитів та забезпечувати захист від витоку даних або несанкціонованого доступу.
3. Безпека: Веб-сервіс повинен мати механізми аутентифікації та авторизації, щоб забезпечити захист особистих даних користувачів.

Також потрібно забезпечити захист від хакерських атак, використовуючи заходи безпеки, такі як шифрування даних та захист від вторгнень.

4. Скалювання: Система повинна бути готова масштабуватися залежно від зростаючої кількості користувачів і навантаження на сервер.
5. Сумісність: Веб-сервіс повинен бути сумісний з різними веб-браузерами і пристроями, щоб користувачі могли отримати доступ до нього з будь-якого пристрою.
6. Інтерфейс: Користувальницький інтерфейс повинен бути зручним, привабливим та інтуїтивно зрозумілим для користувачів.

Аналіз функціональних вимог виявив наступні сутності, які гарантують реалізацію програмного комплексу системи. Діаграма класів, зображена на рис. 2.2, використовується для модуля «Користувачі».

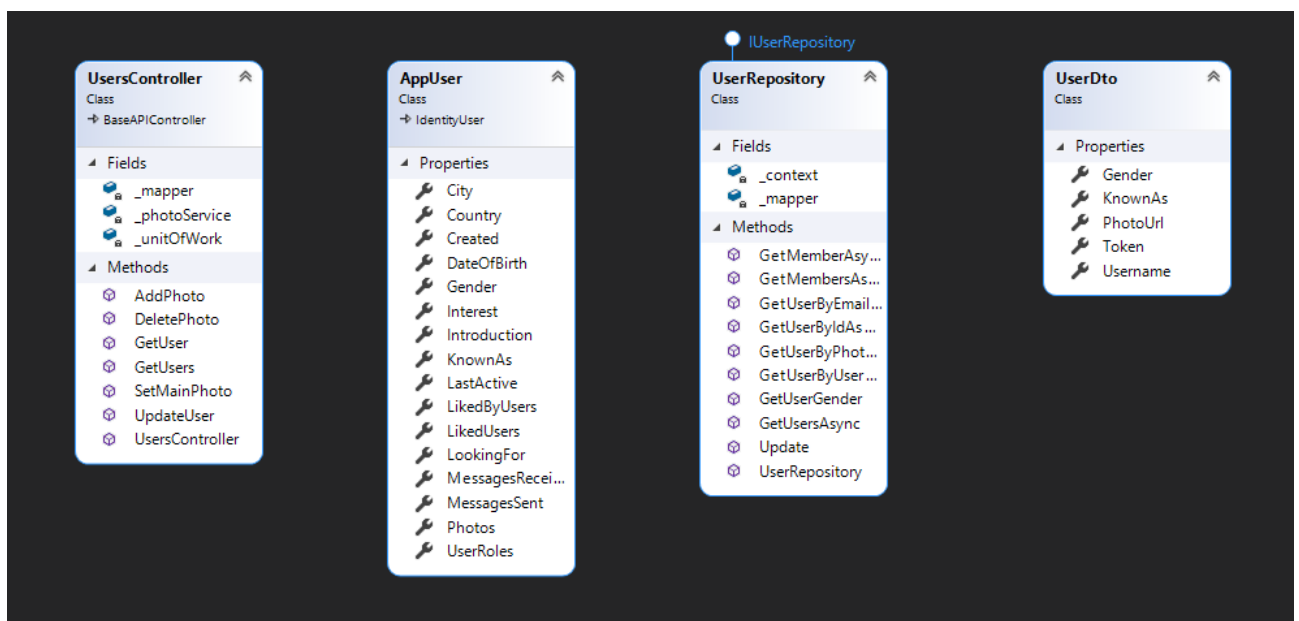


Рис. 2.2 - Діаграма класів модуля «Користувачі»

UsersController – клас, через який відбувається взаємодія клієнта і сервера щодо користувачів:

- **GetUsers**: GET-запит, який повертає список користувачів у форматі MemberDto з можливістю фільтрації та пагінації результатів.

- GetUser: GET-запит, який повертає конкретного користувача за його ім'ям користувача (username) у форматі MemberDto.
- UpdateUser: PUT-запит, який оновлює дані користувача на основі переданого об'єкта MemberUpdateDTO.
- AddPhoto: POST-запит, який додає фотографію користувача. Використовується IFormFile для передачі фотографії.
- SetMainPhoto: PUT-запит, який встановлює фотографію користувача як основну на основі її ідентифікатора (photoId).
- DeletePhoto: DELETE-запит, який видаляє фотографію користувача на основі її ідентифікатора (photoId).

AppUser – клас-модель для сутності користувача:

- DateOfBirth: Дата народження користувача.
- KnownAs: Псевдонім або ім'я, яким користувач відомий.
- Created: Дата створення облікового запису користувача. Значення за замовчуванням - поточна дата та час UTC.
- LastActive: Дата та час останньої активності користувача. Значення за замовчуванням - поточна дата та час UTC.
- Gender: Стать користувача.
- Introduction: Вступний опис користувача.
- LookingFor: Інформація про те, кого користувач шукає.
- Interest: Інтереси користувача.
- City: Місто, в якому проживає користувач.
- Country: Країна, в якій проживає користувач.
- Photos: Колекція фотографій користувача.
- LikedByUsers: Колекція об'єктів UserLike, що представляють користувачів, які виразили вподобання до даного користувача.
- LikedUsers: Колекція об'єктів UserLike, що представляють користувачів, які були виражені вподобанням цим користувачем.
- MessagesSent: Колекція повідомлень, відправлених цим користувачем.

– MessagesReceived: Колекція повідомлень, отриманих цим користувачем.

– UserRoles: Колекція ролей користувача. Використовується для управління ролями та доступом користувача.

Цей клас успадковує клас IdentityUser<int>, який є частиною ASP.NET Identity і надає базову функціональність для управління користувачами, включаючи аутентифікацію, авторизацію та управління ролями.

UserRepository – клас-репозиторій, що виконує операції з користувачами, використовуючи контекст бази даних:

– GetMemberAsync: Асинхронний метод, який отримує об'єкт MemberDto для конкретного користувача за його ім'ям користувача (username). Параметр isCurrentUser вказує, чи це поточний користувач.

– GetMembersAsync: Асинхронний метод, який повертає список користувачів у форматі PageList<MemberDto> з урахуванням пагінації та фільтрації за параметрами UserParams.

– GetUserByEmailAsync: Асинхронний метод, який отримує об'єкт AppUser за його електронною поштою (email).

– GetUserByIdAsync: Асинхронний метод, який отримує об'єкт AppUser за його ідентифікатором (id).

– GetUserByPhotoId: Асинхронний метод, який отримує об'єкт AppUser за ідентифікатором фотографії (photoId).

– GetUserByUsernameAsync: Асинхронний метод, який отримує об'єкт AppUser за його ім'ям користувача (username).

– GetUserGender: Асинхронний метод, який отримує стать користувача за його ім'ям користувача (username).

– GetUsersAsync: Асинхронний метод, який повертає список всіх користувачів у форматі IEnumerable<AppUser>.

– Update: Метод, який оновлює дані користувача у базі даних.

Діаграма класів для модуля «Лайки (вподобання)» зображена на рис. 2.3.

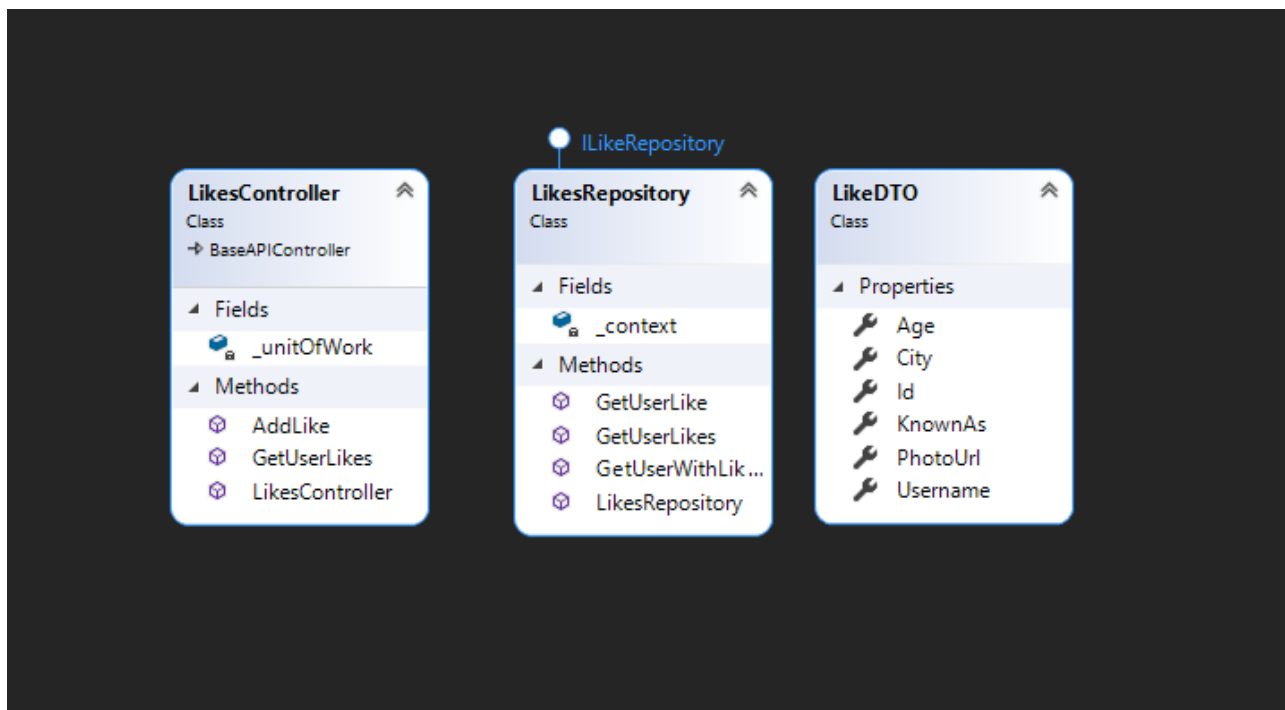


Рис. 2.3 - Діаграма класів модуля «Лайки (вподобання)»

LikesController – клас-контролер, що призначений для роботи з вподобаннями користувачів:

- **AddLike**: Асинхронний метод, який додає лайк користувачу з вказаним ім'ям користувача (username). Перевіряє, чи користувач не лайкає сам себе або вже не лайкав цього користувача раніше.
- **GetUserLikes**: Асинхронний метод, який отримує список лайків користувача, що викликав запит. Застосовує пагінацію та фільтрацію за параметрами LikesParams.

LikesRepository – клас-репозиторій, призначений для роботи з базою даних за допомогою контексту з вподобаннями користувачів:

- **GetUserLike**: Асинхронний метод, який отримує лайк користувача за заданими ідентифікаторами sourceUserId та likedUserId.
- **GetUserLikes**: Асинхронний метод, який отримує список лайків користувача за заданими параметрами LikesParams. Виконує фільтрацію за типом лайку (liked або likedBy) та створює список LikeDTO з інформацією про користувачів, які були лайкнуті або лайкнули.

- GetUserWithLikes: Асинхронний метод, який отримує користувача за заданим ідентифікатором userId разом з його лайками.

LikeDTO – клас-модель, для відображення структури вподобань користувачів:

- Id: Ідентифікатор лайку.
- Username: Ім'я користувача, якому належить лайк.
- Age: Вік користувача, якому належить лайк.
- KnownAs: Псевдонім або відоме як.
- PhotoUrl: URL-адреса фотографії користувача, якому належить лайк.

На рис. 2.4 зображено діаграму класів модуля «Адмін».

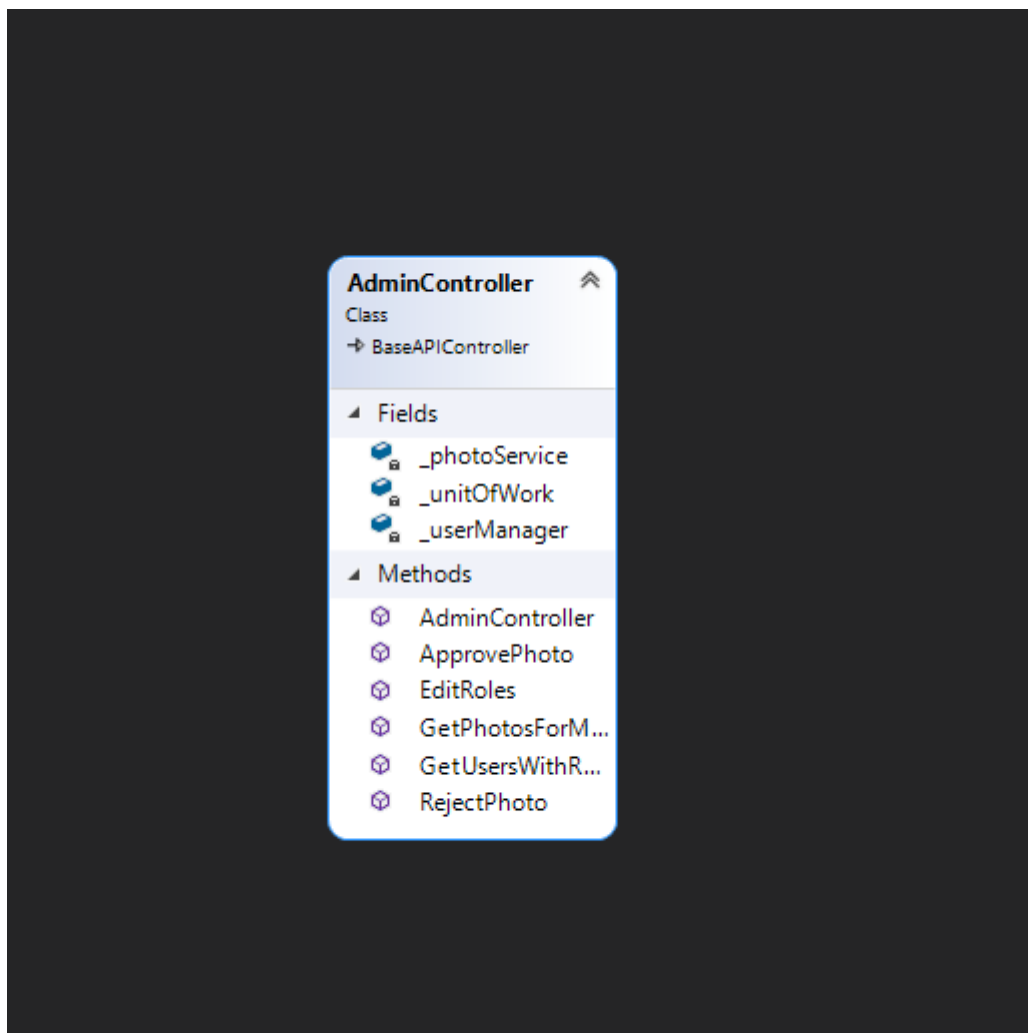


Рис. 2.4 - Діаграма класів модуля «Адмін»

Модуль «Адмін» містить клас-контрол **AdminController**, що включає в себе такі методи:

- **GetUsersWithRoles**: GET-запит для отримання списку користувачів з їхніми ролями. Виконує запит до бази даних і повертає список користувачів з їхніми ідентифікаторами, іменами користувачів та ролями.
- **EditRoles**: POST-запит для редагування ролей користувача. Приймає параметри `username` (ім'я користувача) та `roles` (ролі, розділені комами). Здійснює пошук користувача за іменем, додає або видаляє ролі користувача відповідно до наданих ролей і повертає оновлені ролі користувача.
- **GetPhotosForModeration**: GET-запит для отримання фотографій, які потребують модерації. Виконує запит до бази даних і повертає список фотографій, які ще не схвалені.
- **ApprovePhoto**: POST-запит для схвалення фотографії. Приймає параметр `photoId` (ідентифікатор фотографії). Знаходить фотографію за ідентифікатором, встановлює прапор `IsApproved` в значення `true` і встановлює фотографію як головну для користувача, якщо він не має інших головних фотографій. Зберігає зміни в базі даних.
- **RejectPhoto**: POST-запит для відхилення фотографії. Приймає параметр `photoId` (ідентифікатор фотографії). Знаходить фотографію за ідентифікатором, видаляє її з бази даних та відповідного сховища фотографій, якщо вона існує. Зберігає зміни в базі даних.

2.2 Розробка бази даних системи

Далі розробимо базу даних. Для старту додатку чудово підійде `Sqlite` – легка база, що зберігається в одному файлі, але в подальшому при розвитку додатка, можна буде замінити її дуже легко за допомогою простої підміни бази в конфігурації веб-сервіса та зміни `ConnectionString` для бази. Конфігурація бази даних зображена на рис. 2.5.

```

0 references
public static class ApplicationServiceExtensions
{
    1 reference
    public static IServiceCollection AddApplicationServices(this IServiceCollection services, IConfiguration config)
    {
        services.AddSingleton<PresenceTracker>();
        services.Configure<CloudinarySettings>(config.GetSection("CloudinarySettings"));
        services.AddScoped<ITokenService, TokenService>();
        services.AddScoped<IPhotoService, PhotoService>();
        services.AddScoped<IUnitOfWork, UnitOfWork>();
        services.AddScoped<LogUserActivity>();
        services.AddAutoMapper(typeof(AutomapperProfiles).Assembly);
        services.AddDbContext<DataContext>(options =>
        {
            options.UseSqlite(config.GetConnectionString("DefaultConnection"));
        });

        return services;
    }
}

```

Рис. 2.5 - Конфігурація бази даних

Не втратити дані допоможуть файли міграції, які можуть бути завчасно збережені. Приклад файлів міграції зображений на рис. 2.6.

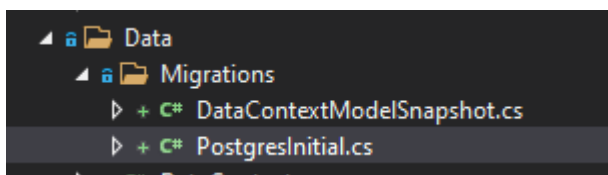


Рис. 2.6 - Файли міграції

Міграції в Entity Framework (EF) - це механізм, який дозволяє автоматизувати процес зміни схеми бази даних, коли ви працюєте з кодом C# або .NET. Вони надають зручний спосіб зберігання та збереження змін у структурі бази даних, а також спрощують розгортання оновлень вашої програми [28-29].

Коли ви розробляєте додаток, який використовує Entity Framework, схема бази даних відображається у вигляді моделі об'єктів C# - набору класів, які представляють таблиці та зв'язки між ними. Однак з часом ви можете змінювати цю модель, додаючи нові таблиці, змінюючи властивості і структуру існуючих таблиць тощо.

Міграції EF дозволяють зберігати зміни у вигляді коду, а потім виконувати цей код для оновлення бази даних до нового стану. Коли ви створюєте нову міграцію, EF аналізує вашу модель та визначає різницю між поточним станом бази даних і новим станом, який ви описали в коді. За допомогою цієї інформації EF створює набір операцій, які потрібно виконати, щоб перевести базу даних в новий стан. Ці операції можуть включати створення нових таблиць, зміну типів стовпців, додавання або видалення стовпців тощо.

Коли ви виконуєте міграцію, EF автоматично створює необхідні SQL-запити для виконання цих операцій на базі даних. Ви можете виконати ці SQL-запити безпосередньо на базі даних або використати механізм Code First міграцій EF, який автоматично виконує ці запити за вас.

Міграції EF зберігаються в спеціальній папці вашого проекту і можуть бути контрольовані версійною системою. Це дозволяє зберігати історію змін схеми бази даних та легко переключатись між різними версіями.

Узагальнюючи, міграції в Entity Framework є потужним інструментом для автоматизації змін схеми бази даних у вашому додатку. Вони дозволяють зберігати зміни в коді та автоматично оновлювати базу даних до нового стану, забезпечуючи зручний та контрольований процес розгортання змін.

Структура бази даних зображена на рис. 2.7 та на рис. 2.8.

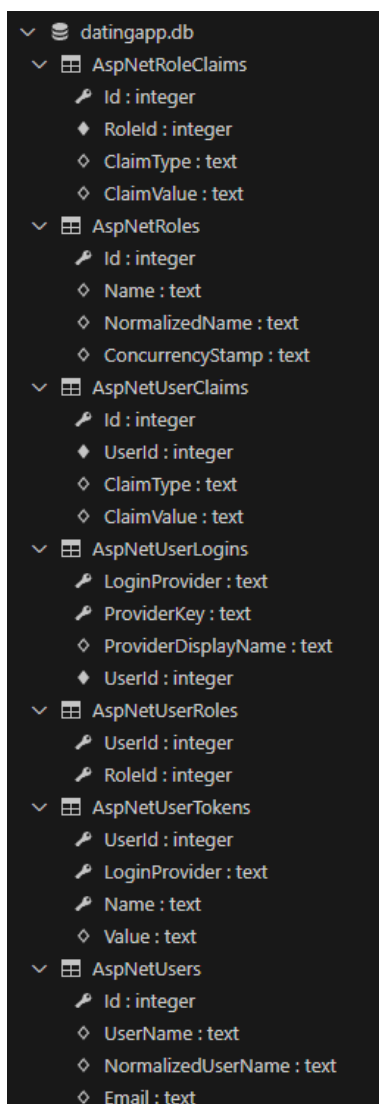


Рис. 2.7 – Автоматично згенеровані таблиці Microsoft Identity

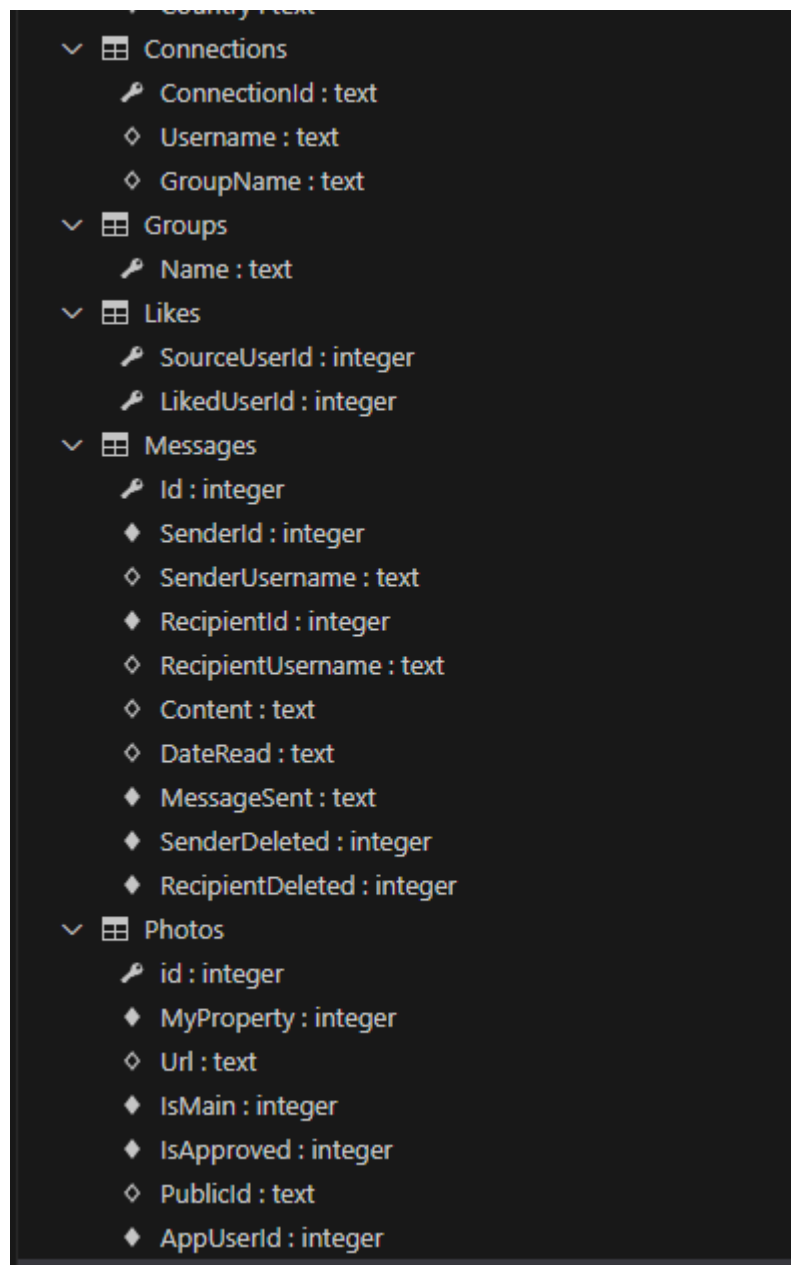


Рис. 2.8 – Створені таблиці, що зберігають користувацьку інформацію

2.3 Опис взаємодії елементів системи під час роботи програмного комплексу

Взаємодія елементів системи під час роботи програмного комплексу веб-сервісу для знайомств може бути організована наступним чином:

Користувачі взаємодіють з системою шляхом реєстрації, авторизації та входу в систему. Після успішного входу в систему, користувачі мають доступ до основних функцій веб-сервісу, таких як створення профілю, перегляд профілів

інших користувачів, пошук потенційних партнерів, надсилання повідомлень, оцінювання профілів та участь в групах і подіях.

Взаємодія з базою даних відбувається для збереження та отримання даних, пов'язаних з користувачами, їх профілями, повідомленнями та іншою важливою інформацією. Користувачі можуть створювати, оновлювати та видаляти свої профілі, змінювати налаштування приватності, зчитувати та відправляти повідомлення, а також отримувати рекомендації щодо партнерів на основі алгоритмів пошуку.

Веб-сервер, як основний компонент системи, обробляє запити від користувачів та надсилає їм відповіді. Він взаємодіє з базою даних для отримання та збереження інформації, виконує бізнес-логіку, пов'язану з функціональними можливостями веб-сервісу, та забезпечує взаємодію з фронтенд-клієнтом через API.

Фронтенд-клієнт представляє інтерфейс користувачам, де вони можуть взаємодіяти з системою. Він передбачає веб-додаток, розроблений з використанням фреймворка Angular. Фронтенд-клієнт взаємодіє з веб-сервером, відправляючи запити на отримання даних, оновлення профілю або надсилання повідомлень. Він також отримує відповіді від сервера та відображає їх користувачу у зрозумілому і зручному форматі.

Сигналізація використовується для забезпечення функціональності чату веб-сервісу. Вона дозволяє користувачам надсилати та отримувати повідомлення в реальному часі без необхідності оновлення сторінки. Коли користувач відправляє повідомлення, сигналізація передає його до веб-сервера, який потім передає його отримувачу, щоб відобразити повідомлення у його чаті.

Таким чином, елементи системи - користувачі, база даних, веб-сервер, фронтенд-клієнт та сигналізація - взаємодіють між собою під час роботи програмного комплексу веб-сервісу для знайомств, забезпечуючи користувачам можливість реєстрації, авторизації, створення та перегляду профілів, пошуку партнерів, надсилання повідомлень. Нижче на рис. 2.9 приведена схема взаємодії користувача з системою та адміністратором.

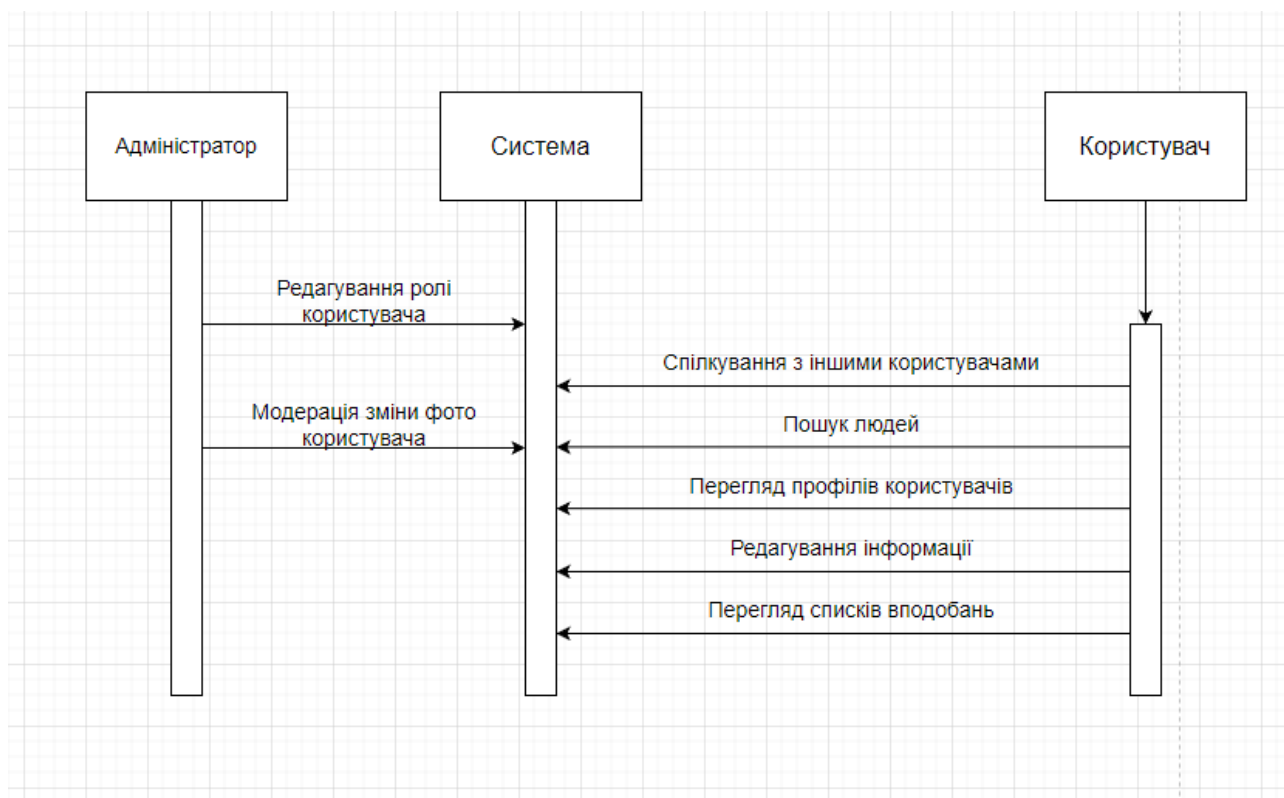


Рис. 2.9 – Схема взаємодії користувача з системою та адміністратором

2.4 Реалізація веб-сервісу для знайомств

Першим і основним модулем будь-якого додатка для знайомств є модуль реєстрації-авторизації. Такий модуль представлено класом `AccountController`, який включає в себе класи для роботи з користувачами (`AccountManager`) та `EmailSender` (клас для відправки емейлів для підтвердження реєстрації та скидування пароля).

В табл. 2.1 наведено назву методів та покроковий опис того, що робить кожен метод у класі `AccountController`:

Таблиця 2.1

Методи контролера `AccountController`

Назва методу	Опис
Register	Реєструє нового користувача. Перевіряє, чи існує вже користувач з таким ім'ям. Створює новий об'єкт AppUser на основі registerDto . Зберігає користувача в базі даних. Надсилає підтвердження електронної пошти. Повертає об'єкт UserDto .
Login	Виконує процес авторизації користувача. Перевіряє правильність введеного логіну та пароля. Повертає об'єкт UserDto з інформацією про користувача.

UserExist	Перевіряє, чи існує користувач з вказаним ім'ям. Повертає true , якщо користувач існує, і false в іншому випадку.
ConfirmEmail	Підтверджує електронну пошту користувача за допомогою отриманого токена та ідентифікатора користувача.
ForgotPassword	Відправляє на електронну пошту користувача лист з посиланням на зміну пароля.
ResetPassword	Скидає пароль користувача з використанням отриманого токена та ідентифікатора користувача.

Розберемо детально метод реєстрації, що використовує підтвердження по електронній пошті:

```
public async Task<ActionResult<UserDto>> Register(RegisterDto registerDto)
{
    if (await UserExist(registerDto.Username)) return BadRequest("UserName
already taken.");

    var user = _mapper.Map<AppUser>(registerDto);

    user.UserName = registerDto.Username.ToLower();
    user.Email = registerDto.Email;
    user.EmailConfirmed = true;

    var result = await _userManager.CreateAsync(user, registerDto.Password);

    if (!result.Succeeded) return BadRequest(result.Errors);
    var token = await _userManager.GenerateEmailConfirmationTokenAsync(user);

    string urlPath = "";
    var env = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");
    if (env.ToLower() == "development")
    {
        urlPath = _config["returnPaths:ConfirmEmail"];
    }
    else
    {
        urlPath =
Environment.GetEnvironmentVariable("ReturnPaths:ConfirmEmail");
    }

    var confirmationLink = URLBuilder.BuildUrl(urlPath, token,
user.Id.ToString());

    var message = new Message(new string[] { user.Email }, "Confirmation Email
Link",
        URLBuilder.BuildContext(URLBuilder.MessageType.ConfirmEmail,
confirmationLink, user.UserName), null);
    await _emailSender.SendEmailAsync(message);

    var roleResult = await _userManager.AddToRoleAsync(user, "Member");
    if (!roleResult.Succeeded) return BadRequest(roleResult.Errors);

    return new UserDto
    {
        Username = user.UserName,
        Token = await _tokenService.CreateToken(user),
        KnownAs = user.KnownAs,
        Gender = user.Gender,
```

```
};
}
```

Перше, що робить метод – це перевіряє чи не створено такого користувача раніше. Якщо ні, намагаємось створити користувача. Якщо з якихось причин створення не відбулось, то віддаємо на клієнт помилку. Якщо користувача створено, далі формуємо листа для підтвердження електронної пошти, так як користувача зареєстровано з прапорцем EmailConfirmed: false. Це означає, що без підтвердження пошти, користувач не зможе авторизуватись. Для відправлення емейлу, за допомогою Identity сервісу будуємо URL, за допомогою якого користувач зможе підтведити свою пошту і далі за допомогою EmailSender надсилаємо листа за допомогою метода SendEmailAsync:

```
public async Task SendEmailAsync(Message message)
{
    var mailMessage = CreateEmailMessage(message);

    await SendAsync(mailMessage);
}
```

Цей метод спочатку створює вміст листа за допомогою метода CreateEmailMessage:

```
private MimeMessage CreateEmailMessage(Message message)
{
    var emailMessage = new MimeMessage();
    emailMessage.From.Add(new MailboxAddress(_emailConfig.From));
    emailMessage.To.AddRange(message.To);
    emailMessage.Subject = message.Subject;

    var bodyBuilder = new BodyBuilder { HtmlBody = string.Format("<h2
style='color:red;'>{0}</h2>", message.Content) };

    if (message.Attachments != null && message.Attachments.Any())
    {
        byte[] fileBytes;
        foreach (var attachment in message.Attachments)
        {
            using (var ms = new MemoryStream())
            {
                attachment.CopyTo(ms);
                fileBytes = ms.ToArray();
            }

            bodyBuilder.Attachments.Add(attachment.FileName, fileBytes,
ContentType.Parse(attachment.ContentType));
        }
    }

    emailMessage.Body = bodyBuilder.ToMessageBody();
    return emailMessage;
}
```

та метода SendAsync:

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	
						40

```

private async Task SendAsync(MimeMessage mailMessage)
{
    using (var client = new SmtpClient())
    {
        try
        {
            await client.ConnectAsync(_emailConfig.SmtServer,
            _emailConfig.Port, false);
            client.AuthenticationMechanisms.Remove("XOAUTH2");
            await client.AuthenticateAsync(_emailConfig.UserName,
            _emailConfig.Password);

            await client.SendAsync(mailMessage);
        }
        catch (Exception e)
        {
            var message = e.Message;
            //log an error message or throw an exception, or both.
            throw;
        }
        finally
        {
            await client.DisconnectAsync(true);
            client.Dispose();
        }
    }
}

```

Цей метод вичитує конфігурацію SMTP сервера та за допомогою SmtpClient надсилає листа.

Висновки до розділу 2

У результаті аналізу варіантів використання веб-сервісу для знайомств, було визначено, що система повинна надати користувачам можливість реєстрації, авторизації, створення та перегляду профілів, пошуку партнерів, надсилання повідомлень та участі в групах та подіях. Об'єктно-орієнтована структура системи була розроблена, враховуючи ці вимоги та розділяючи їх на модулі та компоненти, що спрощує розробку та підтримку системи.

Для забезпечення ефективного зберігання та отримання даних була розроблена база даних системи. Використання бази даних SQLite було обрано з метою простоти встановлення та використання, а також для забезпечення масштабованості системи.

Проектування та реалізація алгоритмів роботи системи були проведені з урахуванням функціональних вимог, таких як пошук потенційних партнерів та

надсилання повідомлень. Були розроблені алгоритми, які забезпечують ефективну та зручну роботу системи.

В результаті реалізації додатку на основі .NET Web API та Angular в середовищах Visual Studio та Visual Studio Code було створено функціональності, яка задовольняє вимоги користувачів. Використання Bootstrap компонентів забезпечує зручний та привабливий інтерфейс користувача, а використання Microsoft Identity дозволяє забезпечити безпеку та авторизацію в системі. Для розсилки електронних листів використовується MailKit, що дозволяє ефективно взаємодіяти з електронною поштою.

Таким чином, розділ 2 відображає всі етапи проектування та реалізації веб-сервісу для знайомств, забезпечуючи функціональні та нефункціональні вимоги, а також створюючи ефективну взаємодію між елементами системи. Результатом розділу є розроблений додаток, який має потужні можливості для знайомств та спілкування між користувачами.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1 Порядок встановлення та налаштування системи.

Система не потребує додаткового встановлення, так як буде розгорнута на окремому веб-сервері. Для доступу до додатка потрібно мати браузер та доступ до мережі Інтернет.

Мінімальні системні вимоги для запуску веб-сервісу наведені в табл. 3.1.

Таблиця 3.1

Компонент	Мінімальні вимоги
Операційна система	Windows 7 або новіше, macOS 10.13 або новіше, Linux
Процесор	Intel Core 2 Duo або еквівалентний
Оперативна пам'ять	Мінімум 2 ГБ RAM
Відеокарта	Сумісна з DirectX 10 або OpenGL 3.3
Роздільна здатність екрану	Мінімум 1280x800 пікселів
Браузер	Остання версія Google Chrome, Mozilla Firefox, Safari, Microsoft Edge

3.2 Структура інтерфейсу веб-сервісу для знайомств

При першому візиті сайту користувач бачить сторінку авторизації (рис. 3.1).

Рис. 3.1 - Сторінка авторизації

Для незареєстрованого користувача потрібно перейти на сторінку реєстрації (рис. 3.2).

Blinder

Зареєструватись

Я: ☒ Чоловік ☐ Жінка

Рис. 3.2 - Сторінка реєстрації.

Форма реєстрації має валідацію, поки форма не заповнена вірно, кнопка «Зареєструватись» неактивна (рис. 3.3).

Blinder

Зареєструватись

Я: ☒ Чоловік ☐ Жінка

Будь ласка, заповніть поле Ім'я

Будь ласка, заповніть поле Місто

Будь ласка, заповніть поле Email

Будь ласка, заповніть поле Підтвердіть Email

Будь ласка, заповніть поле Підтвердіть пароль

Рис. 3.3 - Валідація форми реєстрації

Після реєстрації отримуємо перенаправляємось на сторінку, зображену на рис. 3.4.

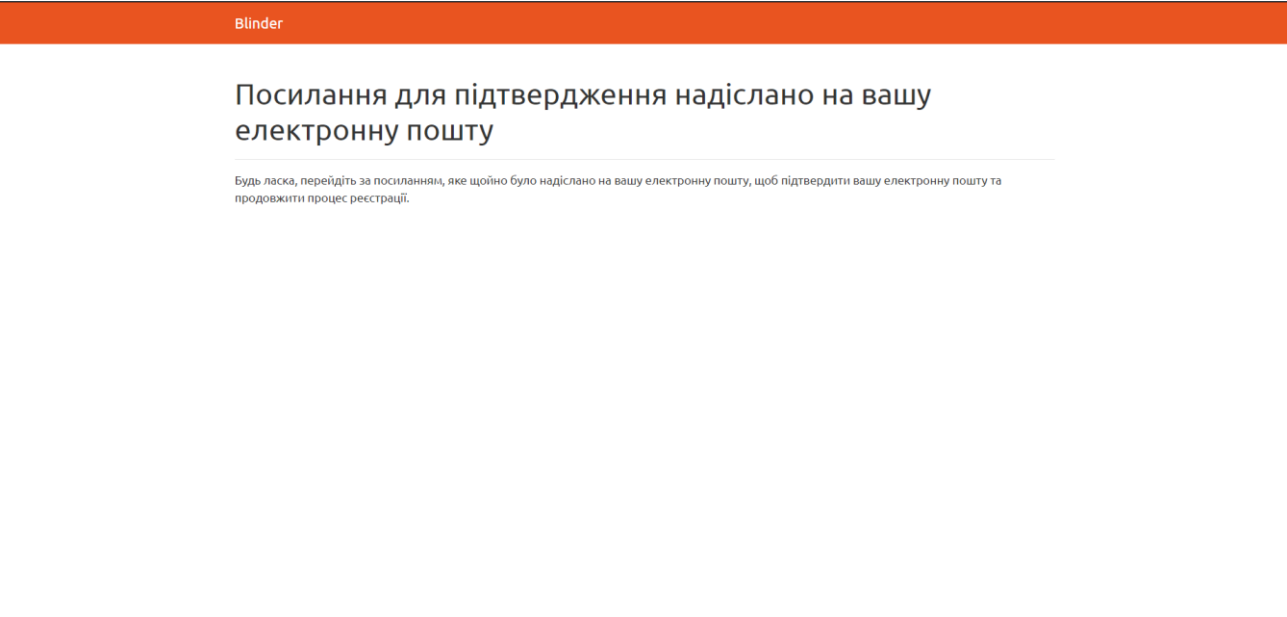


Рис. 3.4 - Сторінка, що повідомляє користувача, що потрібно перевірити Email

Отримуємо листа для підтвердження реєстрації (рис. 3.5).

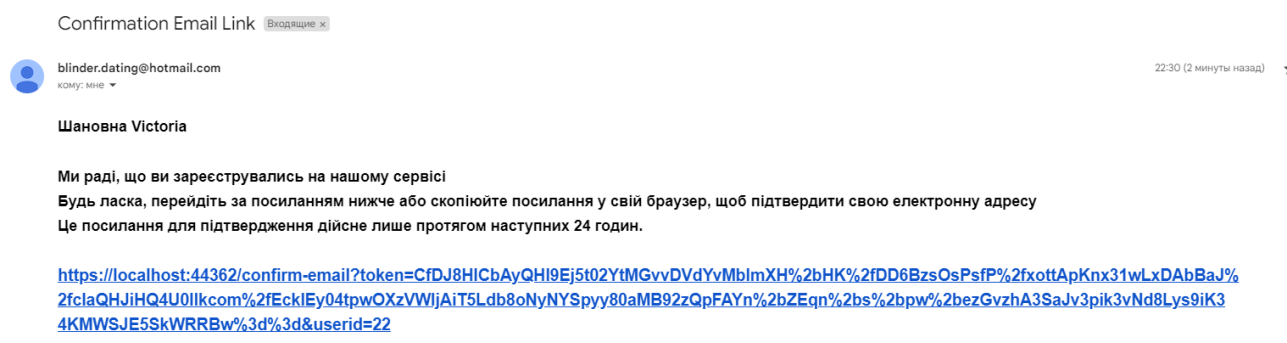


Рис. 3.5 - Лист з підтвердженням реєстрації

Після переходу по посиланню бачимо сторінку, що запрошує авторизуватись (рис. 3.6).

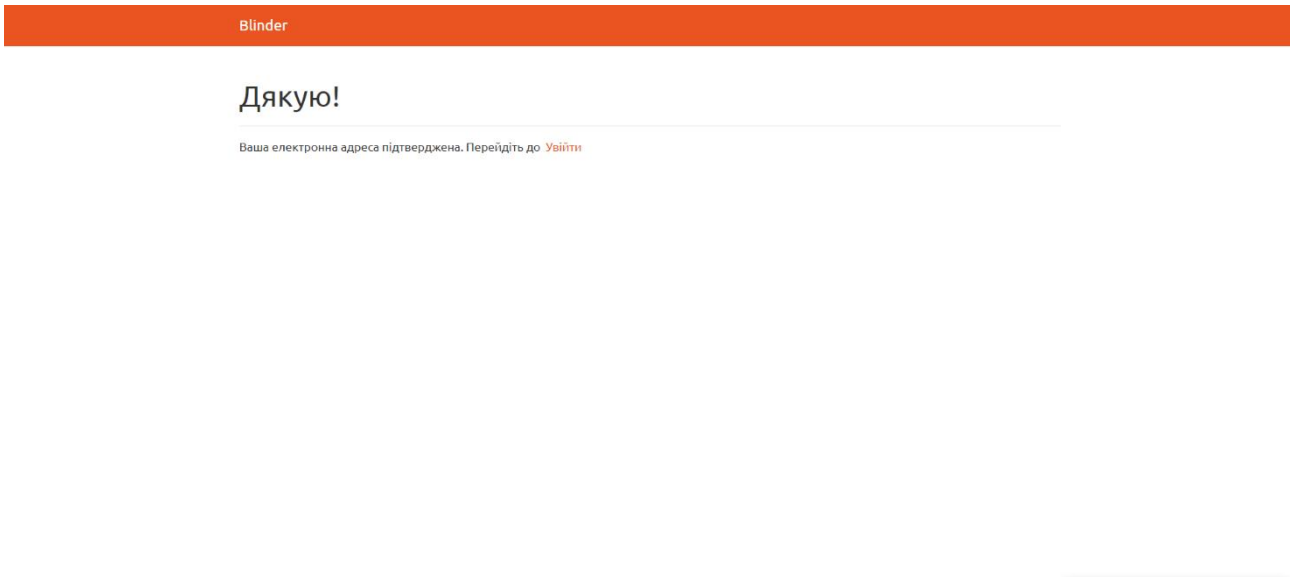


Рис. 3.6 - Сторінка, що запрошує авторизуватись

Далі можемо потрапити на сторінку Метчі, де можемо по фільтру шукати партнерів (рис. 3.7). При наведенні на профіль користувача, можемо побачити 3 кнопки: профіль, вподобання та повідомлення.

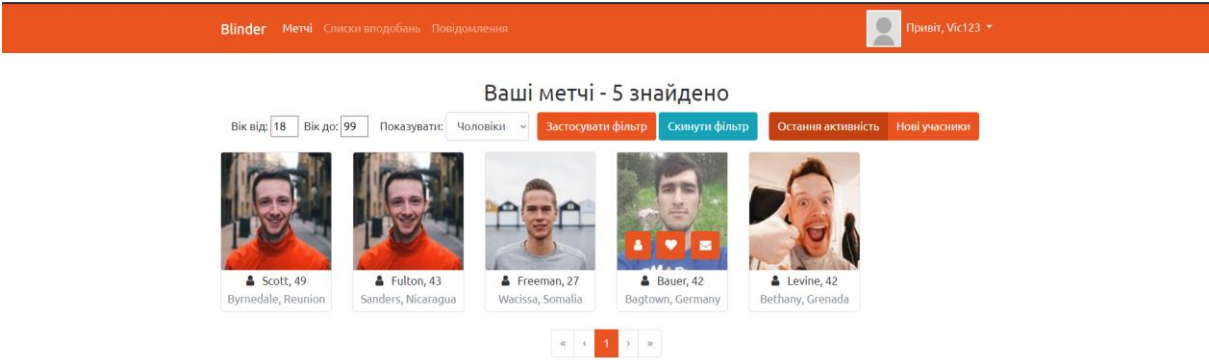


Рис. 3.7 - Сторінка Метчі

Першою натиснемо кнопку «Профіль» та ознайомимось зі змістом профіля користувача (рис 3.8).

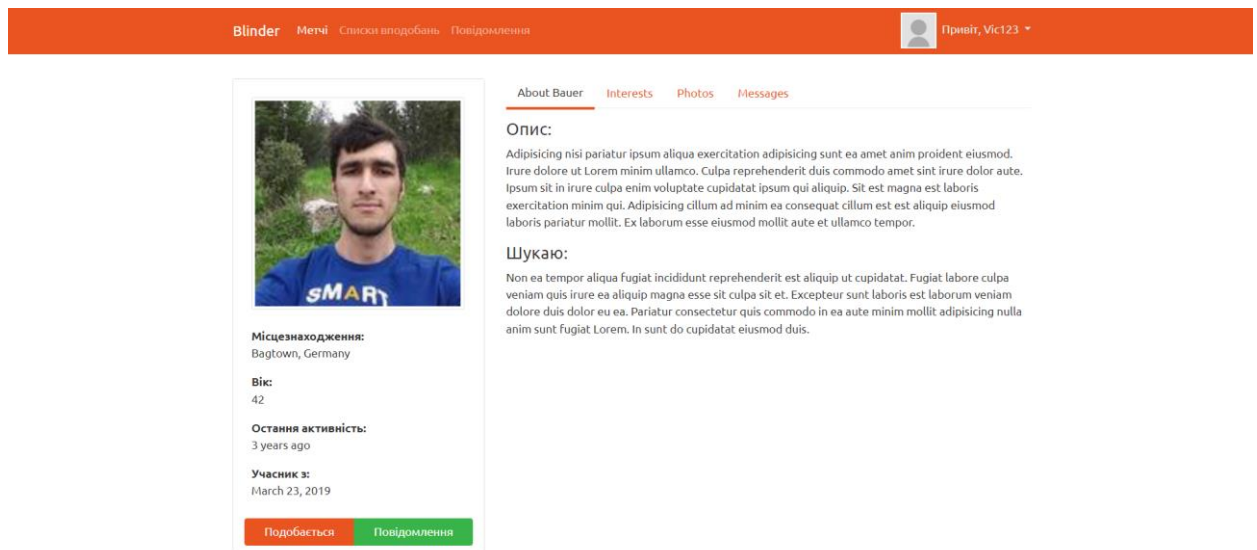


Рис. 3.8 - Сторінка профіля користувача

Після того, як натиснемо кнопку сердечка (подобається), перейдемо на сторінку «Списки вподобань» (рис. 3.9).

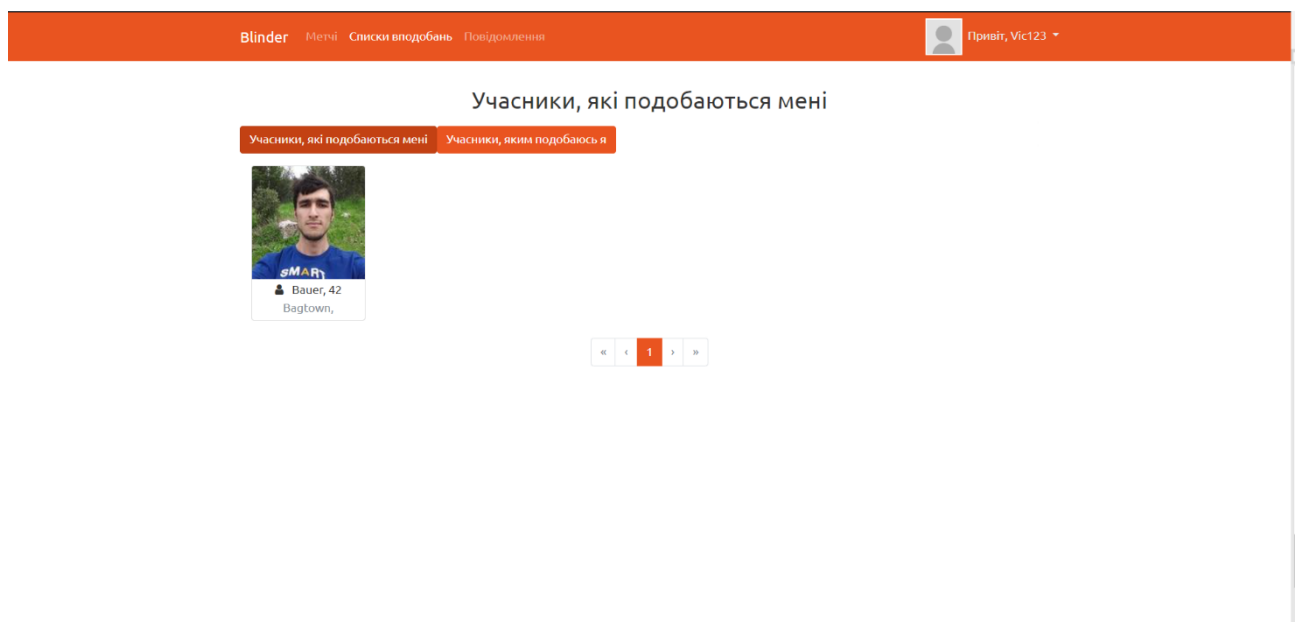


Рис. 3.9 - Сторінка «Списки вподобань».

Далі маємо можливість відправити повідомлення учаснику (рис. 3.10).

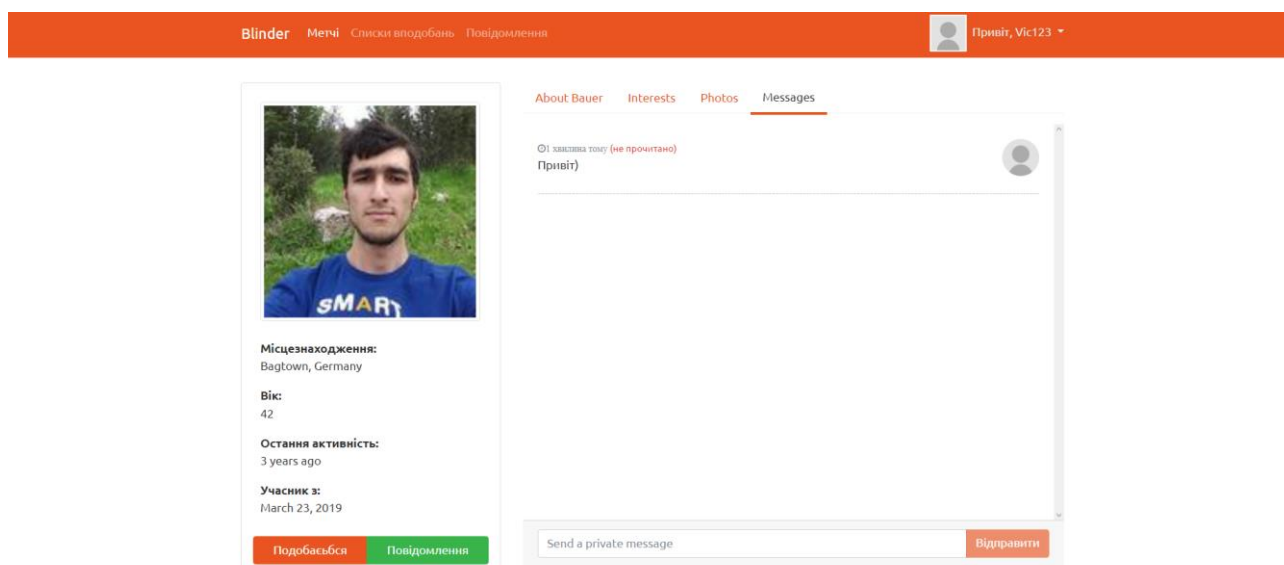


Рис. 3.10 - Відправлення повідомлення учаснику

Онлайн користувачі підсвічуються зеленою іконкою (рис. 3.11).



Рис. 3.11 - Онлайн користувач

Сторінка повідомлень, де користувач може вести кілька діалогів одночасно (рис 3.12) виглядає так:

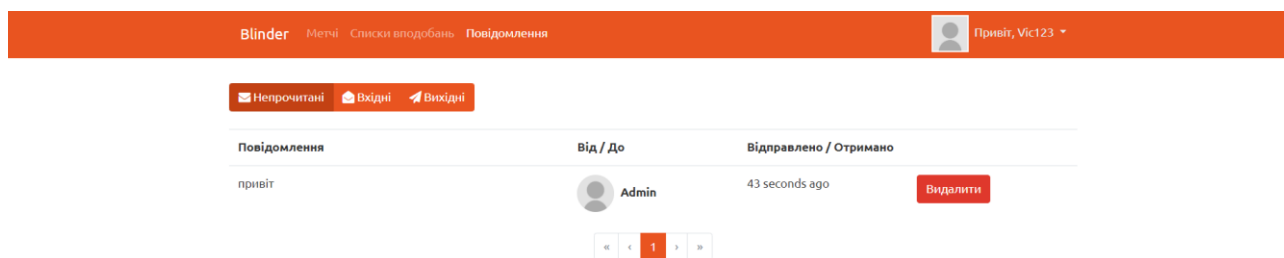


Рис. 3.12 - Сторінка повідомлень

Тепер подивимось Адмінську панель. Перша сторінка це «Управління користувачами» (рис 3.13).

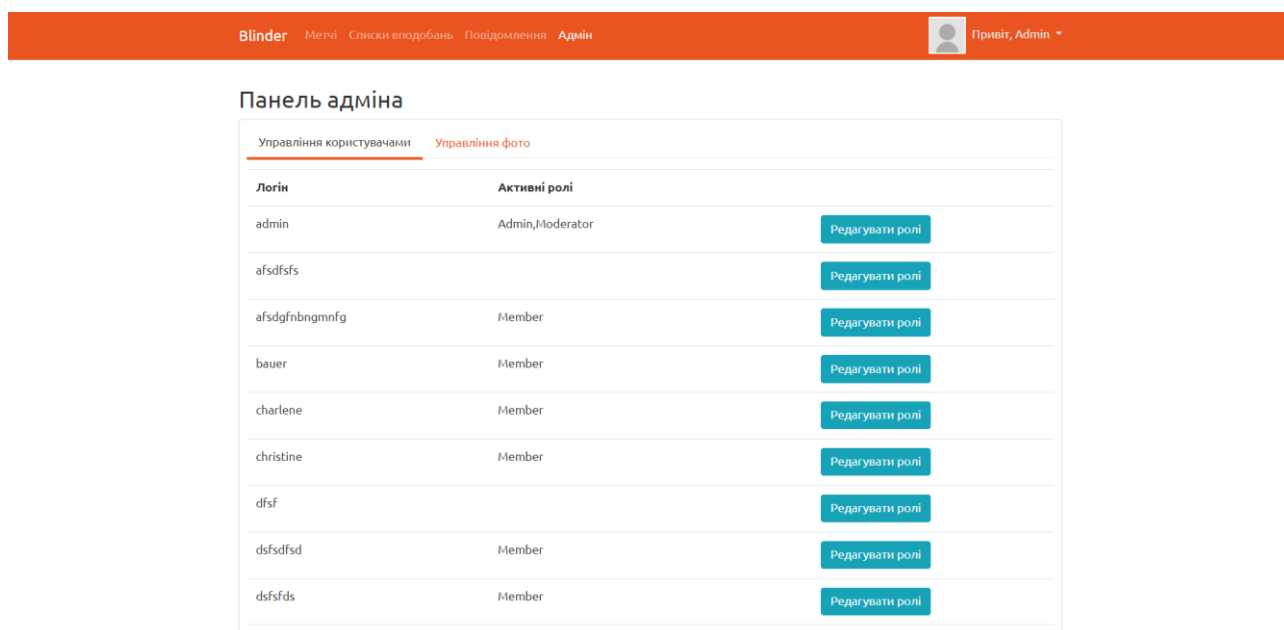


Рис. 3.13 - Адмінська панель (управління користувачами)

Також адмін має модерувати зміну фото користувачів (рис 3.14).

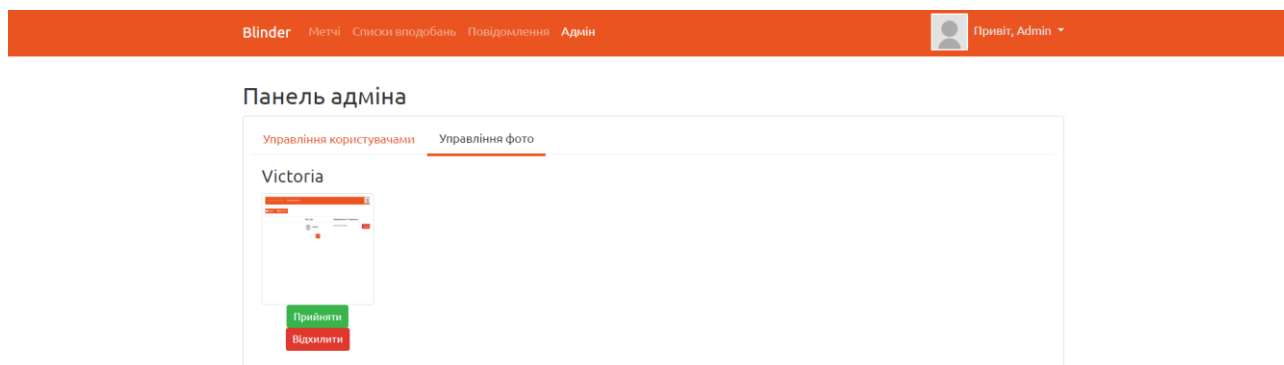


Рис 3.14 - Адмінська панель (управління фото)

3.3 Тестування роботи системи

У цьому розділі описуються методики, інструменти та процедури, використовувані для проведення тестування.

3.1. Планування тестування

У даному підрозділі розробляється план тестування, який визначає стратегію, обсяг та послідовність проведення тестів. Визначаються основні функціональні блоки системи, які будуть перевірятися, а також критичні сценарії використання, що мають бути покриті тестами.

3.2. Функціональне тестування

Функціональне тестування спрямоване на перевірку відповідності функціональних можливостей системи вимогам та очікуванням користувачів. В цьому підрозділі описуються сценарії тестування, включаючи перевірку реєстрації користувачів, пошуку та фільтрації профілів, взаємодії з повідомленнями тощо. Використовуються автоматизовані тести, ручне тестування та моделювання реальних сценаріїв.

3.3. Продуктивність тестування

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						50

У цьому підрозділі оцінюється продуктивність системи за різних навантажень та обсягів даних. Виконуються тести на швидкодію веб-сторінок, відповідь сервера, час завантаження профілів користувачів та інші важливі метрики продуктивності. Застосовуються інструменти для вимірювання та аналізу продуктивності, такі як Apache JMeter, LoadRunner тощо.

3.4. Безпека тестування

В цьому підрозділі проводиться перевірка системи на наявність потенційних вразливостей та заходів безпеки. Виконуються тести на перехоплення трафіку, захист від SQL-ін'єкцій, кросс-сайтових скриптів, аутентифікації та авторизації, обмеження доступу до конфіденційних даних тощо. Використовуються відомі методи тестування безпеки, такі як пенетраційне тестування та сканування вразливостей [27].

Тестування функціональності. Нижче наведено тест-кейси роботи основних модулів додатку.

Реєстрація нового користувача

Тест-кейс: ТС-001

Опис: Перевірка функціональності реєстрації нового користувача в системі.

Кроки:

- Відкрити веб-браузер та перейти на головну сторінку веб-сервісу для знайомств.
- Натиснути на кнопку "Реєстрація".
- Заповнити обов'язкові поля форми реєстрації (ім'я, електронна пошта, пароль).
- Натиснути кнопку "Зареєструватися".
- Перевірити, чи з'являється повідомлення про успішну реєстрацію та перехід на сторінку входу в систему.

Очікуваний результат: Після успішної реєстрації користувача, система повинна вивести повідомлення про успішну реєстрацію та перенаправити на сторінку входу.

Вхід в систему

Тест-кейс: ТС-002

Опис: Перевірка функціональності входу в систему для зареєстрованого користувача.

Кроки:

- Відкрити веб-браузер та перейти на головну сторінку веб-сервісу для знайомств.
- Натиснути на кнопку "Увійти".
- Заповнити обов'язкові поля форми входу (електронна пошта, пароль).
- Натиснути кнопку "Увійти".
- Перевірити, чи здійснено успішний вхід в систему та перехід на головну сторінку.

Очікуваний результат: Після успішного входу в систему, користувач повинен бути перенаправлений на головну сторінку веб-сервісу.

Пошук користувачів

Тест-кейс: ТС-003

Опис: Перевірка функціональності пошуку користувачів за заданими критеріями.

Кроки:

- Відкрити веб-браузер та здійснити вхід в систему.
- У полі пошуку ввести критерії пошуку (вік, стать, інтереси тощо).
- Натиснути кнопку "Пошук".
- Перевірити, чи відображаються користувачі, що відповідають заданим критеріям.

Очікуваний результат: Після натискання кнопки "Пошук", система повинна відобразити список користувачів, що відповідають заданим критеріям пошуку.

Відправка повідомлення

Тест-кейс: ТС-004

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						52

Опис: Перевірка функціональності відправки повідомлення іншому користувачу.

Кроки:

- Відкрити веб-браузер та здійснити вхід в систему.
- Знайти потрібного користувача за допомогою пошуку або перейти на його профіль.
- Натиснути кнопку "Написати повідомлення".
- Заповнити поле повідомлення та натиснути кнопку "Надіслати".

Очікуваний результат: Після натискання кнопки "Надіслати", система повинна відправити повідомлення обраному користувачу.

Результати тестування

У процесі тестування було перевірено основні функціональні можливості веб-сервісу для знайомств. Всі тест-кейси були успішно пройдені, що свідчить про правильну реалізацію функцій системи та їх відповідність вимогам.

Висновки до розділу 3

У розділі було розглянуто процес встановлення та налаштування параметрів системи, включаючи налаштування доступу, підключення до бази даних та інші конфігураційні кроки. Це дозволяє забезпечити належне функціонування системи та готовність до роботи.

Структура інтерфейсу веб-орієнтованої системи була розглянута з точки зору користувача. Описано основні елементи інтерфейсу, їх функціональні можливості та способи взаємодії з системою. Це допомагає забезпечити зручну та ефективну роботу користувачів з системою.

Тестування роботи програмного комплексу було проведено з метою перевірки правильності реалізації функціональних та нефункціональних вимог, а також виявлення можливих помилок та недоліків. Тестування дозволяє переконатися, що програмний комплекс працює стабільно, відповідає вимогам та задовольняє потреби користувачів.

В результаті проведення розділу 3 було встановлено, що веб-сервіс готовий до використання, налаштований на потреби користувачів та пройшла успішне тестування.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						54

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано веб-сервіс для знайомств:

- Було проведено детальне дослідження та аналіз функціональних вимог, архітектурних патернів та інструментальних засобів для розробки веб-сервісу для знайомств.
- В рамках аналізу предметної області було визначено основні функціональні можливості, які має мати веб-сервіс для знайомств, такі як створення профілів користувачів, пошук та фільтрація, спілкування та інші.
- Були оглянуті існуючі на ринку аналоги, з'ясовано їхні переваги та недоліки, а також виявлено прогалини у функціональних можливостях, які можуть бути враховані при розробці нового веб-сервісу.
- Обґрунтовано вибір інструментальних засобів, таких як .NET Web API та Angular для розробки веб-сервісу, Visual Studio та Visual Studio Code для розробки в середовищі, SQLite як база даних, SignalR для реалізації чату, Bootstrap компоненти для фронтенду, Microsoft Identity для реєстрації та авторизації, і MailKit для розсилки електронних листів.
- Для забезпечення ефективного зберігання та отримання даних була розроблена база даних системи. Використання бази даних SQLite було обрано з метою простоти встановлення та використання, а також для забезпечення масштабованості системи.
- Проектування та реалізація алгоритмів роботи системи були проведені з урахуванням функціональних вимог, таких як пошук потенційних партнерів та надсилання повідомлень. Були розроблені алгоритми, які забезпечують ефективну та зручну роботу системи.
- Структура інтерфейсу веб-орієнтованої системи була розглянута з точки зору користувача. Описано основні елементи інтерфейсу, їх функціональні можливості та способи взаємодії з системою. Це допомагає забезпечити зручну та ефективну роботу користувачів з системою.

– Тестування роботи програмного комплексу було проведено з метою перевірки правильності реалізації функціональних та нефункціональних вимог, а також виявлення можливих помилок та недоліків. Тестування дозволяє переконатися, що програмний комплекс працює стабільно, відповідає вимогам та задовольняє потреби користувачів.

Загальний аналіз і обґрунтування вибору показали, що запропоновані технології та архітектурні рішення мають потенціал задовольнити функціональні вимоги та очікування користувачів веб-сервісу для знайомств. Реалізований програмний комплекс готовий до використання.

Розроблена система веб-сервісу для знайомств пропонує базовий набір функціональності, що включає створення профілів користувачів, пошук і спілкування між ними, а також налаштування критеріїв пошуку. Однак, ця система має потенціал для подальшого розширення та вдосконалення. Можливість використання інтелектуального пошуку, що використовує штучний інтелект для аналізу та рекомендацій потенційних партнерів на основі інтересів та поведінкових зразків користувачів.

Крім того, розробниками може бути розглянута можливість створення мобільної версії системи, що дозволить користувачам зручно використовувати сервіс на різних мобільних пристроях. Це розширення забезпечить більшу доступність та зручність використання для користувачів, що активно використовують мобільні пристрої в повсякденному житті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Н.Р. Балик, В.І. Мандзюк Тернопільський національний педагогічний університет: Сучасні клієнт-серверні технології та їх застосування при вивченні систем управління базами даних [Електронний ресурс] – Режим доступу до ресурсу: https://fi.npu.edu.ua/files/Zbirnik_KOSN/12/29.pdf
2. Bumble [Електронний ресурс] – Режим доступу до ресурсу: <https://bumble.com>
3. OkCupid [Електронний ресурс] – Режим доступу до ресурсу: <https://www.okcupid.com>
4. Tinder [Електронний ресурс] – Режим доступу до ресурсу: <https://tinder.com/>
5. Developer Mozilla: MVC [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>.
6. TechTarget: MVC [Електронний ресурс] – Режим доступу до ресурсу: <https://www.techtarget.com/whatis/definition/model-view-controller-MVC>
7. Developer Mozilla: MVC [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/mvc-framework-introduction/>
8. Microsoft Learn: Microservice Architecture style [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>
9. Microservice Architecture: Microservice Architecture Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://microservices.io/patterns/microservices.html>
10. Microservice Architecture: Microservice Architecture Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://microservices.io/patterns/microservices.html>
11. Medium: Microservices [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9>

12. Microsoft Learn: Create a web API with ASP.NET Core [Электронный ресурс]
– Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio>
13. Microsoft Learn: Create a web API with ASP.NET Core [Электронный ресурс]
– Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0>
14. .NET API Reference [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.microsoft.com/en-us/dotnet/api/>
15. Angular Docs – Режим доступа до ресурсу: <https://angular.io/docs>
16. Dev Docs : Angular – Режим доступа до ресурсу: <https://devdocs.io/angular/>
17. Visual Studio Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.microsoft.com/en-us/visualstudio/>
18. Visual Studio Code Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://code.visualstudio.com/docs>
19. SQLite.org [Электронный ресурс] – Режим доступа до ресурсу: <https://www.sqlite.org/>
20. Clickhouse: Sqlite [Электронный ресурс] – Режим доступа до ресурсу: <https://clickhouse.com/docs/ru/engines/database-engines/sqlite>
21. SQLite Database System Design and Implementation - Antti Salonen
22. Microsoft Learn: Introduction to SignalR [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>
23. Bootstrap Docs [Электронный ресурс] – Режим доступа до ресурсу: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
24. Hubspot: How to Add Bootstrap to Angular [Электронный ресурс] – Режим доступа до ресурсу: <https://blog.hubspot.com/website/add-bootstrap-to-angular-2>
25. Microsoft Learn: Microsoft Identity Web authentication library [Электронный ресурс] – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/azure/active-directory/develop/microsoft-identity-web>

- 26.Mailkit: features [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.mailkit.com/features>
- 27.QaGroup: Види тестування [Електронний ресурс] – Режим доступу до ресурсу:
<https://qagroup.com.ua/publications/vydy-testuvannya-ta-vidminnosti-mizh-nymy/>
- 28.Microsoft Learn: Getting Started with EF Core [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>
- 29.Microsoft Learn: Migrations overview [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli>
- 30.Learn Visual Studio for C# - Ed Freitas

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						60

Контролер Адміна

AdminController.cs

```

using API.Entities;
using API.Interfaces;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;

namespace API.Controllers
{
    public class AdminController : BaseApiController
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly IUnitOfWork _unitOfWork;
        private readonly IPhotoService _photoService;

        public AdminController(UserManager<AppUser> userManager, IUnitOfWork
unitOfWork, IPhotoService photoService)
        {
            _userManager = userManager;
            _unitOfWork = unitOfWork;
            _photoService = photoService;
        }

        [Authorize(Policy = "RequireAdminRole")]
        [HttpGet("users-with-roles")]
        public async Task<ActionResult> GetUsersWithRoles()
        {
            var users = await _userManager.Users
                .Include(r => r.UserRoles)
                .ThenInclude(r => r.Role)
                .OrderBy(o => o.UserName)
                .Select(u => new
                {
                    u.Id,
                    Username = u.UserName,
                    Roles = u.UserRoles.Select(r => r.Role.Name).ToList()
                })
                .ToListAsync();

            return Ok(users);
        }

        [HttpPost("edit-roles/{username}")]
        public async Task<ActionResult> EditRoles(string username, [FromQuery]string
roles)
        {
            var selectedRoles = roles.Split(",").ToArray();
            var user = await _userManager.FindByNameAsync(username);
            if (user == null) return NotFound("Could not find user: " + username);

            var userRoles = await _userManager.GetRolesAsync(user);

            var result = await _userManager.AddToRolesAsync(user,
selectedRoles.Except(userRoles));
            if (!result.Succeeded) return BadRequest("Failed to add user to roles");
        }
    }
}

```

```

        result = await _userManager.RemoveFromRolesAsync(user,
userRoles.Except(selectedRoles));
        if (!result.Succeeded) return BadRequest("Failed to remove user to
roles");

        return Ok(await _userManager.GetRolesAsync(user));
    }

    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpGet("photos-to-moderate")]
    public async Task<ActionResult> GetPhotosForModeration()
    {
        var photos= await _unitOfWork.PhotoRepository.GetUnapprovedPhotos();

        return Ok(photos);
    }

    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpPost("approve-photo/{photoId}")]
    public async Task<ActionResult> ApprovePhoto(int photoId)
    {
        var photo = await _unitOfWork.PhotoRepository.GetPhotoById(photoId);
        if (photo == null) return NotFound("Could not find photo");

        photo.IsApproved = true;

        var user = await _unitOfWork.UserRepository.GetUserByPhotoId(photo.id);
        if (!user.Photos.Any(photo => photo.IsMain)) photo.IsMain = true;

        if (await _unitOfWork.Complete()) return Ok();

        return BadRequest("Failed to approve a photo");
    }

    [Authorize(Policy = "ModeratePhotoRole")]
    [HttpPost("reject-photo/{photoId}")]
    public async Task<ActionResult> RejectPhoto(int photoId)
    {
        var photo = await _unitOfWork.PhotoRepository.GetPhotoById(photoId);
        if (photo == null) return NotFound("Could not find photo");
        if (photo.PublicId != null)
        {
            var result = await _photoService.DeletePhotoAsync(photo.PublicId);
            if (result.Result == "ok")
            {
                _unitOfWork.PhotoRepository.RemovePhoto(photo);
            }
        }
        else
        {
            _unitOfWork.PhotoRepository.RemovePhoto(photo);
        }

        if (await _unitOfWork.Complete()) return Ok();

        return BadRequest("Failed to reject a photo");
    }
}
}

```

Контролер повідомлень

MessagesController.cs

```
using API.DTOS;
using API.Entities;
using API.Extensions;
using API.Helpers;
using API.Interfaces;
using AutoMapper;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace API.Controllers
{
    [Authorize]
    public class MessagesController : BaseApiController
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public MessagesController(IUnitOfWork unitOfWork, IMapper mapper)
        {
            _unitOfWork = unitOfWork;
            _mapper = mapper;
        }

        [HttpPost]
        public async Task<ActionResult<MessageDto>> CreateMessage(CreateMessageDto createMessage)
        {
            var username = User.GetUsername();
            if (createMessage.RecipientUsername.ToLower() == username.ToLower())
                return BadRequest("You cannot send messages to yourself");

            var sender = await
            _unitOfWork.UserRepository.GetUserByUsernameAsync(username);
            var recipient = await
            _unitOfWork.UserRepository.GetUserByUsernameAsync(createMessage.RecipientUsername);

            if (recipient == null) return NotFound();

            var message = new Message
            {
                Sender = sender,
                Recipient = recipient,
                SenderUsername = sender.UserName,
                RecipientUsername = recipient.UserName,
                Content = createMessage.Content
            };

            _unitOfWork.MessageRepository.AddMessage(message);

            if (await _unitOfWork.Complete()) return
            Ok(_mapper.Map<MessageDto>(message));
            return BadRequest("Failed to send Message to " + recipient.UserName);
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<MessageDto>>>
        GetMessagesForUser([FromQuery] MessageParams messageParams)
        {
            messageParams.Username = User.GetUsername();
        }
    }
}
```

```

        var messages = await
_unitOfWork.MessageRepository.GetMessagesForUser(messageParams);
        Response.AddPaginationHeader(messages.CurrentPage, messages.PageSize,
messages.TotalCount, messages.TotalPages);

        return messages;
    }

    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteMessage(int id)
    {
        var username = User.GetUsername();

        var message = await _unitOfWork.MessageRepository.GetMessage(id);

        if (message.Sender.UserName != username && message.Recipient.UserName !=
username)
            return Unauthorized();

        if (message.Sender.UserName == username) message.SenderDeleted = true;
        if (message.Recipient.UserName == username) message.RecipientDeleted =
true;

        if (message.SenderDeleted && message.RecipientDeleted)
            _unitOfWork.MessageRepository.DeleteMessage(message);

        if (await _unitOfWork.Complete()) return Ok();

        return BadRequest("Problem Deleting the message");
    }
}
}

```

Контролер аккаунта користувача

AccountController.cs

```
using API.DTOS;
using API.Entities;
using API.Interfaces;
using AutoMapper;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Threading.Tasks;
using EmailService;
using Message = EmailService.Message;
using Microsoft.AspNetCore.Authorization;
using System;
using Microsoft.Extensions.Configuration;
using System.Web;
using API.Helpers;

namespace API.Controllers
{
    public class AccountController : BaseApiController
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly SignInManager<AppUser> _signInManager;
        private readonly ITokenService _tokenService;
        private readonly IMapper _mapper;
        private readonly IEmailSender _emailSender;
        private readonly IConfiguration _config;

        public AccountController(
            UserManager<AppUser> userManager,
            SignInManager<AppUser> signInManager,
            ITokenService tokenService,
            IMapper mapper,
            IEmailSender emailSender,
            IConfiguration config
        )
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _tokenService = tokenService;
            _mapper = mapper;
            _emailSender = emailSender;
            _config = config;
        }

        [HttpPost("register")]
        [AllowAnonymous]
        public async Task<ActionResult<UserDto>> Register(RegisterDto registerDto)
        {
            if (await UserExist(registerDto.Username)) return BadRequest("UserName
already taken.");

            var user = _mapper.Map<AppUser>(registerDto);

            user.UserName = registerDto.Username.ToLower();
            user.Email = registerDto.Email;
            user.EmailConfirmed = true;

            var result = await _userManager.CreateAsync(user, registerDto.Password);

            if (!result.Succeeded) return BadRequest(result.Errors);
        }
    }
}
```

```

var token = await _userManager.GenerateEmailConfirmationTokenAsync(user);

string urlPath = "";
var env = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");
if (env.ToLower() == "development")
{
    urlPath = _config["returnPaths:ConfirmEmail"];
}
else
{
    urlPath =
Environment.GetEnvironmentVariable("ReturnPaths:ConfirmEmail");
}

var confirmationLink = URLBuilder.BuildUrl(urlPath, token,
user.Id.ToString());

var message = new Message(new string[] { user.Email }, "Confirmation Email
Link",
    URLBuilder.BuildContext(URLBuilder.MessageType.ConfirmEmail,
confirmationLink, user.UserName), null);
await _emailSender.SendEmailAsync(message);

var roleResult = await _userManager.AddToRoleAsync(user, "Member");
if (!roleResult.Succeeded) return BadRequest(roleResult.Errors);

return new UserDto
{
    Username = user.UserName,
    Token = await _tokenService.CreateToken(user),
    KnownAs = user.KnownAs,
    Gender = user.Gender,
};
}

[HttpPost("login")]
public async Task<ActionResult<UserDto>> Login(LoginDTO loginDto)
{
    if (loginDto.Username == null) return Unauthorized("Missing Username!");
    if (loginDto.Password == null) return Unauthorized("Missing Password!");

    var user = await _userManager.Users
        .Include(p => p.Photos)
        .SingleOrDefaultAsync(x => x.UserName == loginDto.Username.ToLower());

    if (user == null) return Unauthorized("Username not Found");

    var result = await _signInManager.CheckPasswordSignInAsync(user,
loginDto.Password, false);

    if (result == Microsoft.AspNetCore.Identity.SignInResult.Failed)
    {
        return Unauthorized("Invalid Password.");
    }
    else if (result == Microsoft.AspNetCore.Identity.SignInResult.NotAllowed)
    {
        return Unauthorized("User cannot sign in without a confirmed email.");
    }

    return new UserDto
    {
        Username = user.UserName,
        Token = await _tokenService.CreateToken(user),
        PhotoUrl = user.Photos.FirstOrDefault(a => a.IsMain == true)?.Url,
        KnownAs = user.KnownAs,
    }
}

```

```

        Gender = user.Gender
    };
}

private async Task<bool> UserExist(string username)
{
    return await _userManager.Users.AnyAsync(x => x.UserName.ToLower() ==
username.ToLower());
}

[HttpPost("confirmemail")]
public async Task<ActionResult> ConfirmEmail(ConfirmEmailDto confirmEmail)
{
    var user = await _userManager.FindByIdAsync(confirmEmail.userid);
    if (user == null) return Unauthorized("Username not Found");

    var confirm = await _userManager.ConfirmEmailAsync(user,
confirmEmail.token);
    if (confirm.Succeeded) return Ok();

    return BadRequest("Unable to confirm email");
}

[HttpPost("forgotpassword")]
public async Task<ActionResult> ForgotPassword(ForgotPasswordDto
forgotPassword)
{
    if (forgotPassword.Email == null) return BadRequest("Missing Email!");

    var user = await _userManager.Users
        .IgnoreQueryFilters()
        .Where(e => e.Email.ToLower() ==
forgotPassword.Email.ToLower())
        .FirstOrDefaultAsync();

    if (user == null) return Unauthorized("Username not Found");

    var token = await _userManager.GeneratePasswordResetTokenAsync(user);

    string urlPath = "";
    if (Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT").ToLower()
== "development")
    {
        urlPath = _config["returnPaths:PasswordChange"];
    }
    else
    {
        urlPath =
Environment.GetEnvironmentVariable("ReturnPaths:PasswordChange");
    }

    var changePasswordLink = URLBuilder.BuildUrl(urlPath, token,
user.Id.ToString());

    var message = new Message(new string[] { user.Email }, "Reset Password
link",
        URLBuilder.BuildContext(URLBuilder.MessageType.ResetPassword,
changePasswordLink,user.UserName), null);
    await _emailSender.SendEmailAsync(message);

    return Ok();
}

[HttpPost("resetpassword")]
public async Task<ActionResult> ResetPassword(ResetPasswordDto resetPassword)

```

```

    {
        var user = await _userManager.FindByIdAsync(resetPassword.UserId);
        if (user == null) return Unauthorized("Username not Found");

        var result = await _userManager.ResetPasswordAsync(user,
resetPassword.Token, resetPassword.Password);
        if (result.Succeeded) return Ok();

        return BadRequest("Unable to reset password");
    }
}

```