Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка
до дипломної роботи

магістерський
(рівень вищої освіти)

на тему: "Розроблення соціальної мережі "4students" з використанням SignalR"

Виконав: студент 6 курсу групи КН-61м
спеціальності 122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Дідик П.Ю.
(прізвище та ініціали)

Керівник Крошний І.М.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Львів – 2021

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Крошний І. М.
"____" _____20___ року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Дідику П.Ю
(прізвище, ім'я, по батькові)

1. Тема роботи «Розроблення соціальної мережі "4students" з використанням SignalR» керівник роботи *доц., к.т.н., Крошний .І.М.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "31" грудня 2020 року № N С-593.
2. Термін подання студентом роботи 10 грудня 2021
3. Вихідні дані до роботи Постановка задачі та її формалізація
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ХАРАКТЕРИСТИКА ОБ'ЄКТУ ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ
Огляд нейроботів та інтернет-аналогів для онлайн-знайомств
Постановка завдань і вимог системи розробки
ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ
Технічний огляд серверної частини
Технічний огляд клієнтської частини
Технічний огляд створення роботів
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ
Загальна структура програмного продукту
Огляд інших бібліотек і фреймворків
Написання програмного забезпечення для проекту
ЕКСПЕРЕМЕНТАЛЬНА ЧАСТИНА
Вимоги до обладнання
Опис інтерфейсу користувача
Огляд інтерфейсу сервера та інтерфейсу бази даних
РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ
Опис ідеї проекту
Організаційний та фінансовий плани
Ризики проекту
5. Дата видачі завдання 18 грудня 2020 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел та інтернет-ресурсів за вказаною тематикою.	03.02.2021 р. 22.04.2021 р.	
2	Постановка задачі та вибір елементної бази для проектування соціальної мережі	23.04.2019 р. 15.05.2021 р.	
3	Проектування бази даних системи	15.05.2021 р. 10.06.2021 р.	
4	Розробка серверної частини	11.06.2021 р. 21.07.2021 р.	
5	Розробка клієнтської частини	22.07.2021 р. 17.08.2021 р.	
6	Експерементальна частина та оформлення клієнтської частини	18.08.2021 р. 11.09.2021 р.	
7	Тестування серверної частини системи	12.09.2021 р. 14.09.2021 р.	
8	Оформлення записки до дипломної роботи.	15.10.2021 р. 09.11.2021 р.	
9	Здача пояснювальної записки на рецензування.	10.12.2021 р. 12.12.2021 р.	
10	Підготовка доповіді та мультимедійної презентації.	13.12.2020 р. 17.12.2020 р.	

Студент _____ **Дідик П.Ю.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Крошний І.М.**
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна магістерська робота містить 63 сторінки пояснювальної записки, 28 рисунків, 1 таблицю, 2 додатки, 20 джерел.

Ключові слова:

.Net, Angular, Signalr ,Чат, Бот, Соціальні мережі, Чат-бот, Аналіз даних.

АНОТАЦІЯ

У рамках кваліфікаційної магістерської роботи здійснюється розробка веб-додатку з використанням Signalr. Користувач має можливість вести окрему приватну розмову з іншою особою. Під час процесу спілкування кожна особа має свого робота-помічника, з яким можна зв’язатися в будь-який час, і він може використовувати дані користувача для організації зустрічей та аналітики Google . Оскільки базова архітектура програмного продукту використовує традиційну архітектуру клієнт-сервер, сервер буде написаний з використанням цибулевої архітектури, що дозволяє створювати продукти, які легко розширювати. Клієнтська частина буде розроблена як SPA на фреймворку Angular, який має компонентну архітектуру.

Систему можна розробити як окремий продукт, оскільки має власну базу даних, процес реєстрації та аутентифікації, а також особистий кабінет. Для використання програмного продукту користувач повинен зареєструватися, щоб система могла розпізнати користувача та додатково аналізувати його персональні дані для розробки бота.

ABSTACT

Diploma work contains 63 pages of explanatory note, 28 figures, 1 table, 2 appendices, 20 sources.

Keywords:

.Net, Angular, Chat, Bot, Social-network, Chat-bot, Data analysis.

ANNOTATION

As part of the qualifying master's work, a web application is developed using Signalr. The user has the opportunity to have a separate private conversation with another person. During the communication process, each person has their own assistant robot, who can be contacted at any time and can use user data to organize meetings and Google analytics. Because the underlying architecture of the software product uses the traditional client-server architecture, the server will be written using an onion architecture that allows you to create products that are easy to extend. The client part will be developed as a SPA on the Angular framework, which has a component architecture.

The system can be developed as a separate product, as it has its own database, registration and authentication process, as well as a personal account. To use the software product, the user must register so that the system can recognize the user and further analyze his personal data to develop a bot.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

ПЗ – програмне забезпечення;

Solid - збірник принципів об’єктно орієнтованого програмування;

Endpoint - точка доступу на серверній частині;

Backend - серверна частина;

Frontend - клієнтська частина;

UI - графічний вигляд;

AI – штучний інтелект;

ВСТУП

Актуальність дослідження. З розвитком науки і техніки та прогресом суспільства виникають проблеми у спілкуванні людей. Якщо в минулому спілкування відбувалося лише в реальності, тобто віч-на-віч, то на сьогоднішній день ми можемо спостерігати картину того, що спілкування в Інтернеті є невід'ємною частиною сучасного світу.

Різні чати та месенджери зараз все частіше використовуються для обміну даними, особистою інформацією та більш ефективного і менш ресурсовитратного спілкування. Це питання дуже актуальне, адже в сучасному світі все більше людей звертаються до спілкування в мережі, і “онлайн” знайомства не є винятком.

Метою дослідження магістерської кваліфікаційної роботи є розробка програмного продукту для популяризації знайомств і стимулювання проводження спільного часу, адже за допомогою нейроботів будуть обрані місця, які потенційно можуть зацікавити співрозмовників.

Боти зі штучним інтелектом з кожним роком стають все більш популярними, а штучний інтелект починає заполоняти все більше життєвих сфер діяльності. Детальний аналіз даного питання на сьогодні є одним з головних завдань для розробників, які намагаються створити нову систему.

Об’єкт досліджень магістерської роботи — це набір алгоритмів, які використовуються для аналізу та навчання нейронних роботів, методів його реалізації зі сторони як серверної так і клієнтської частин, а також аналіз соціальних мереж, що зараз представлені нам на світовому ринку, які використовуються для знаходження нових знайомств.

Предмет дослідження магістерської роботи – використання системи розробки нейронних роботів у сучасному фреймворку, тобто використання Microsoft Bot Framework для реалізації та навчання нейронних мереж, а також потенційна можливість створення таких мереж на

сервері та метод обміну. даних між користувачами.

Основні задачі які підлягали дослідженню у магістерсьокої кваліфікаційні роботі є:

– методи комунікації в Інтернеті;

– проблема покладання нейронної основи робота на стороні клієнта, його безпека та надійність;

– вивчення та впровадження служб Google для обробки даних;

– розподіл навантаження серверної частини під час пікової активності користувача;

– дослідження та аналіз літератури та програмних продуктів, які зараз пропонуються нам ринком для використання;

– економічна оцінка розвитку та функціонування системи.

– проведення системного аналізу;

– дослідження та розробка системного алгоритму;

– використання сучасні технології для розробки програмних продуктів;

Практична цінність магістерської робота полягає в розробці системи, яка може бути використана як вихідний стан продукту та її базова структура як єдиний незалежний продукт, а також збільшення та покращення бази навчання нейроботів.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ОБ'ЄКТУ ДОСЛІДЖЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Огляд нейроботів та інтернет-аналогів для онлайн-знайомств

Технологічний прогрес набирає все більших обертів, у сучасному житті продовжує з'являтися велика кількість цікаві технологій і запроваджуються нові тенденції, левова частка яких має значний вплив на ІТ-індустрію. Наразі великий ажіотаж викликають програми, які не так давно почали з'явилятися в закритих чатах, і, які пророкують доволі хорошу перспективу сферам маркетингу та комунікації.

Що ж означає «чат-бот»? Чат-ботом називають певний додаток на основі платформи обміну повідомлень, який дає можливість користувачам здійснювати взаємодію разом зі сторонніми сервісами за допомогою простого інтерфейсу звичайного чату.

Іншими словами, для отримання деяких даних чи інформації для користувача нема жодної потреби у виході за межі Месенджера, а вистачить лише обрати та надіслати певну команду із наданого списку, яку бот зможе інтерпретувати. Наприклад, нема потреби здійснювати велику кількість маніпуляцій, щоб знайти де ж подивитися прогноз погоди на сьогодні, а достатньо лише обрати одну команду, щоб отримати на екрані готовий результат (Рис. 1.1):

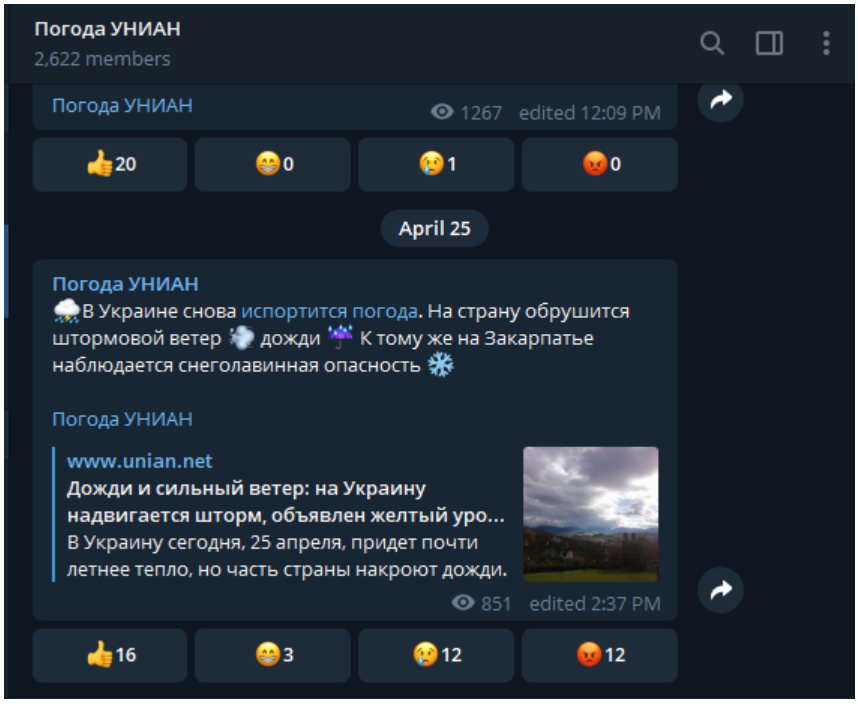

Рис. 1.1 Телеграм бот прогнозу погоди

Або для того, щоб прочитати останні актуальні новини, не потрібно йти до магазину та купувати газету, а достатньо в кілька кліків отримати найактуальнішу інформацію (Рис. 1.2):

Рис. 1.2 Телеграм бот ТСН новин

Це допоміжні речі, які дають можливість для користувачів швидкого отримання необхідної інформації здійснюючи мінімальну кількість зусиль, при цьому всьому немає жодної потреби виходити за межі активного вікна Месенджера. З того часу, як користувачам було надано до користування так званих ботів для чату влітку 2015 року кількість роботів, які вирішують різноманітні завдання, збільшується загрозливою швидкістю. З їх допомогою можна швидко отримувати інформацію про погоду, курси валют, новини, здійснювати швидкі переклади слів та фраз, чи навіть чогось більшого, розважатися(є й такі, що дають можливість деякій кількості людей дистанційно зіграти славнозвісну "Мафію"), здійснювати обмін файлами , а також вирішувати багато важливих запитань чи завдань. Нижче наведено підбірку корисних роботів:

- @ novaposhta_bot – його головним призначенням являється можливість відстеження доставки товару. Все, що для цього потрібно, просто відправити номер накладної і очікувати відповідь, яка повинна надійти протягом декількох секунд. - @CurrencyUABot – можливість отримання оперативної інформації про актуальні станом на даний момент курси валют України, НБУ, а також іншу інформацію, яка стосується валюти.

- @vkmusic_bot – надає можливість прослуховування та завантаження музичних файлів на сайт VK.comю Окрім того, є ще можливість додавати треків у свою музичну колекцію(плейлист). Для прослуховування є можливість використання свого списку відтворення або ж в цілях пошуку нового треку можна використати плейлист свого друга. Окрім того, є можливість відтворення музичних композицій в рандомному порядку.

- @temp_mail_bot – якщо людині швидко потрібно провести якісь маніпуляції і при цьому нема потреби надавати особисту електронну адресу, то можна сміливо використовувати даний бот, який має змогу надати тимчасову та обмежену в часі адресу електронної пошти. Її можна використати в різних цілях: для реєстрації на певних сайтах чи в різних службах, в яких в подальшому не буде жодної потреби. Електронна адреса, яку було надано користувачу даним чат-ботом, буде доступна протягом 10 хвилин, після чого вона самознищиться.

- @ ytranslatebot – дає можливість здійснювати переклад тексту будь-якою з запропонованих мов. І в сучасному світі використання Telegram-ботів для бізнесу має велику кількість переваг:

- швидке повідомлення користувачів в онлайн режимі (сповіщення про різноманітні акції, знижки, а також про наявність тих чи інших товарів у актуальному каталозі); - інформація знаходиться у вільному доступі та

здійснюється швидке реагування на запити від користувачів(від однієї до трьох секунд); - надається можливість простого звернення до технічної підтримки або отримання необхідних консультацій;

- спілкування безпосередньо з потенційно можливими замовниками; - певна доступність для розробки (визначається за загально витраченим часом і кошторисом); - доступна синхронізація даних між активними пристроями – це діалогове вікно можна використовувати на будь-якому комп'ютері та мобільному пристрої, просто увійдіть у свій обліковий запис.

Бот може працювати виключно з певними веб-сайтами чи інтернет-магазинами, а також може отримувати інформацію з будь-якого сервісу.

Слід розуміти, що функціональність роботів поки обмежена в порівнянні з повноцінними мобільними аплікаціями або веб-додатками. Та, тим не менш, багато компаній доволі обережно використовують їхні вказівки, не знаючи, чи потрібні вони зараз.

Не здивувати нікого, що люди зараз мають знайомства і побачення «онлайн». Існує багато сервісів, які можуть забезпечити це, найпопулярніші з яких:

- Tinder - чи не найпопулярніша на сьогодні мобільна платформа для забезпечення можливості швидкої та легкої зустрічі з іншими.

- Badoo (Бадоо) - доволі схожий за функціоналом та можливостями аналог вищезгаданої соцмережі.

То чому б не використати можливість об'єднання сервісу для зустрічі разом з чат-ботами в нейромережах, які можуть допомогти вам спілкуватися з людьми та придумувати цікаві теми для розмови. Тож давайте продовжимо обговорення технології,

1.2. Постановка завдань і вимог системи розробки

Основною метою системи є надання зручних та ефективних послуг для користувачів, які хочуть отримати нові знайомства. Окрім цього, це також можливість, щоб використовувати чат для продовження зав'язаного спілкування. Системі необхідно продовжувати працювати з великим напливом користувачів та одночасним використанням ними даної системі у різних куточках світу. І працювати при цьому всьому системі необхідно стабільно, так як за несправної роботи, кількість користувачів почне поступово зменшуватися. Одними з базових вимог до системи вважаються наступні: – висока швидкодія та максимально можливий короткий час відповіді;

– надійність в роботі;

– забезпечення конфіденційності обробки персональних даних;

– гнучкість дизайну та розширення.

Розробка системи повинна здійснюватися відповідно до гнучкої методології Kanban, а кожен стан системи має зберігатися на відкритій платформі (наприклад GitHub), на якій працює GIT.

Сучасний світ більше не сприймає старі способи спілкування, людям доводиться чекати відповідей або використовувати складні для розуміння інтерфейси. З прискоренням технологічного розвитку продовжують виникати нові стандарти, а впровадження нових стандартів зробить продукцію конкурентоспроможною та самореалізовувану. Спілкування між людьми – це сфера, яка ніколи не припиняється, тому при розробці системи слід розглянути можливість впровадження нових стандартів і технологій розробки системи, оскільки вона незабаром «застаріє».

1.3. Висновки до розділу

Отже, в цьому розділі ми отримали інформацію, щодо аналогів додатку. Дізнались, як боти спрощують життя людей просто отримуючи певне повідомлення, яке надає коротку і актуальну інформацію. А також дізнались про загальні вимоги до сучаних додатків.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Технічний огляд серверної частини

Написання серверної частини буде проводитись з використанням C#, тому що вона має всі інструменти для звязку з клієнтськими додатками. Тому ми будемо її використовувати для розробки API. Отже, давайте розглянемо, що таке Web Api і як ми будемо його використовувати.

API – система, яка складається з відкритих методів з простими точками дотику і використання.

Як говорилось, API - це простий інтерфейс. Цей інтерфейс дозволяє розробникам використовувати готові блоки для створення програм. Його можна використовувати в розробі мобільних додатків, веб-систем або може бути бібліотекою «розумного дому».

API зазвичай використовує дані для повернення даних в одному з двох форматів: XML або JSON. Я буду використовувати JSON формат для своєї API, тому що JSON зрозуміліший та читабельніший, ніж XML.

Веб-додаток (наприклад, Github) має також власний API, який можна вільно та відкрито використовувати програмістам. Для прикладу, Github API надає загальну інформацію про аватар, дані про корисутувача, репозиторії, читачів.

Наприклад, якщо ви б хотіли використосувати API Facebook, інтерфейс може надавати інформацію про дані користувачів, їх новини, а також доступ до груп і пабліків читачів. Це лише деякі з функцій, які програміст може використовувати для власних додатків. Також Facebook API нажає можливості аутентифікації користувачів через їхній програмний код.

Різні клієнти Twitter і Facebook розроблені на основі API, наприклад, карти 2ГІС. Саме тому, що супутні сервіси мають якісні API, і всі їхні функції реалізовані.

Запит даних від стороннього API виглядає так:

```
curl -X POST "https://api.dev.clikk.app/api/Auth/LoginWithSocial" -H  "accept: text/plain" -H  "Content-Type: application/json" -d "{\"socialName\":0,\"socialAccessToken\":\"string\",\"name\":\"string\",\"dateOfBirth\":\"2021-12-20T12:45:53.430Z\",\"email\":\"string\"}"
```

Навіщо потрібен API програми?

Отже, для чого може використовуватись API:

1. Розділення клієнта і сервера. Що дає зможу використовувати додатки незалежно і в любий момент відмовитись від частини.
2. Мобільний клієнт. Використовуючи ці API працюють усі мобільні додатки та різні служби. Підтримавши API і зробивши простий мобільний додаток - клієнт з телефоном скачавши певний додаток може отримувати інформацію що надає API. Це зручно, швидко і просто.
3. Відкрите джерело. Можна розробити самостійне API і виставити в відкритий доступ, його можуть підхопити інші програмісти і використовувати в власних цілях або провджувати розвивати.

Також на стороні сервера буде викристовуватись база даних SQL для зберігання даних, які потрібні для функіонування системи. Буде використовуватись MSSQL для проектування бази даних.

Буде використовуватись технологія Entity Framework для проектування бази даних. Давайте дізнаємося більше про цей фреймворк в нашій системі.

Entity Framework (EF) — це кросплатформна версія технології доступу до даних з відкритим вихідним кодом Entity Framework.

EF Core надає можливість розрорбити реаляційну базу даних з простим інтерфесом для .Net розробника.

У EF Core використовуються модель для доступу до даних. Тобто, програмісту слід написати модель і ця модель через фреймворк імплементується в таблицю SQL.

Для дослідження серверних методів і для зручності використання API було підєднано технологію Swagger. Програмістам клієнтської частини буде простіше її використовувати, оскільки все буде задокументовано.

По суті, Swagger - документація API. Його суть полягає в тому, що він дозволяє не тільки переглядати документацію, а й відправляти запити - так званий інтерфейс Swagger UI виглядає так (Рис. 2.1):

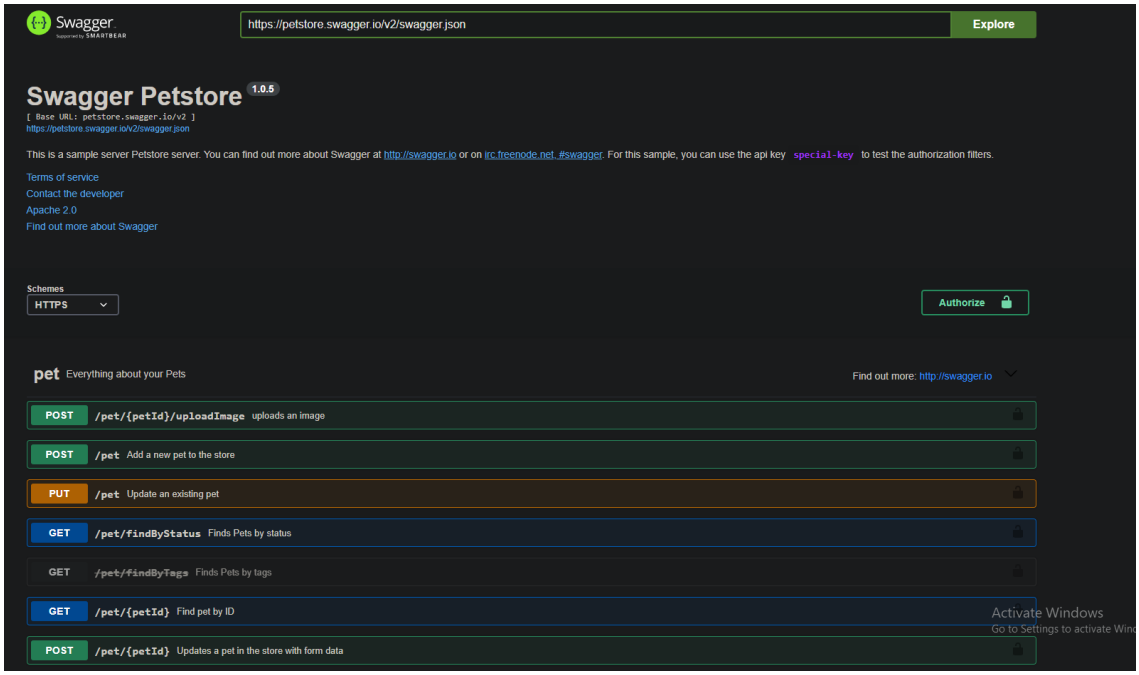

Рис. 2.1 Огляд Swagger

Тут зображено інтерфейс, а саме: назви контроллера, методи, параметри.

2.2. Технічний огляд клієнтської частини

Для розробки клієнтської частини було взято SPA Angular.

Angular заснований на концепції «модель-вигляд-контролер». Автор фреймворка називає це поняття «модель-погляд-*» або навіть «модель-погляд-будь-де».

Фреймворк надає можливості, що дозволяє розробникам створювати односторінкові програми. Фреймворк розділяє логіку програми від DOM. Його призначення — створення динамічних сторінок.

Причини чому я вирішив використовувати Angular для розробки веб-сайту:

1. Angular написаний на TypeScript. До JS доєднується можливість класів, методів, інтерфейси.
2. Angular є кросплатформним, а отже написаний код один раз може використовуватись на різних девайсах, такі як мобільні додатки, настільні додатки, веб сайти і системи.
3. Angular має інструменти для створення програм. Такі як створення динамічної розмітки чи створення динамічної валідації для форми.

4. Angular надає шаблони для створення та підтримки проектів. Якщо правильно створити проект, то створиться проект з простою і ясною структурою, який буде просто розширювати.
5. Від'єднані компоненти. Angular може легко замінити всі існуючі компоненти використовуючи NodeJS.
6. Angular надає можливості для написання Unit тестів використовуючи Jasmine і Protractor фреймворки.
7. Angular — кросплатформний фреймворк.
8. Angular активно підтримується, випускаються нові версії, має величезну спільноту та екосистему. Ви можете знайти багато готових компонентів, матеріалів і корисних сторонніх інструментів.

Висновок: Angular підходить для розробки клієнта нашої системи.

Для розмітки буде використовуватись Bootstrap і ngx-bootstrap, для спрощення дизайну через використання готових компонентів.

2.3. Технічний огляд створення роботів

Я вибрав Microsoft Bot Framework для створення ботів, тому що він повністю підтримує .Net, є відкритим вихідним кодом і простим у використанні та розумінні.

Щоб створити бота, вам потрібно надати попередню версію Microsoft Bot Framework і відповідний API, про які ми розповімо нижче.

Для роботи з ботом будуть використовувати наступні пакети:

- Пакет SDK Bot Builder (для C # і Node.js) призначений для створення основних функцій роботи. Він заснований на WebAPI і визначає протокол зв'язку між роботою і зовнішнім світом. Пакет SDK включає симулятор налаштування та набір класів, які забезпечують реалізацію окремих ключових абстракцій (наприклад, доші діалогове вікно стану).
- Bot Connector прив'язує бота до одного або кількох каналів зв'язку, таких як Skype, Slack, Telegram тощо.

Нейронна мережа - це тип машинного навчання, який імітує роботу людського мозку Створює штучну нейронну мережу, яка використовує алгоритми, щоб комп'ютер міг навчатися, додаючи нові.

Для реалізації бота, йому слід написати програму і надати ресурси для навчання, ось 3 найпопулярніші стратегії навчання бота:

1. Контрольоване навчання – найпростіший варіант стратегії, тому що вам слід вказати набір даних і кінцевий результат і лиш очікувати допоки бот не дойде до цілі.
2. Навчання без нагляду – стратегія, коли немає набору даних. Нейронна мережа аналізує набір даних, а потім певна функція повідомляє нейронній мережі, наскільки вона віддалена від цілі. Згодом можна звузити круг даних звідки ми отримаємо більш точні результати.
3. Навчання з підкріпленням – цей варіант можна використовувати для проходження цілі, а саме, якщо бот зазнає невдачі то він повністю анулює цю спробу і більше не повертається до неї, таким чином він аналізує всі спроби , а якщо жодна не підійшла, то він знаходить найбільш успішну і розбирає її вже більш детально, щоб зрозуміти де помилки і пройти фазу найбільш успішно.

Схема нейронної мережі показана на малюнку (Рис. 2.2):

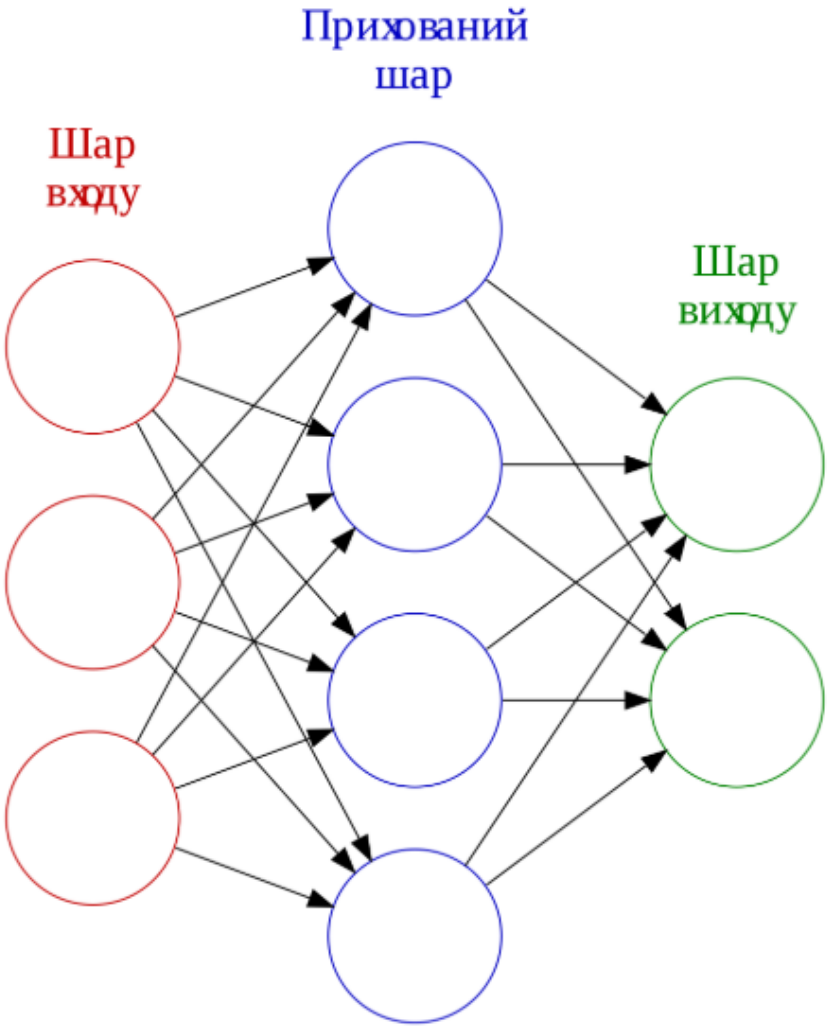

Рис. 2.2 Схема нейронної мережі

Штучний інтелект розділяють на 2 типи: слабкий або сильний. Слабкий штучний інтелект (також званий вузьким штучним інтелектом) — це система штучного інтелекту, розроблена та підготовлена для виконання конкретних завдань. Віртуальні персональні помічники (наприклад, Apple Siri) — це ослаблений штучний інтелект. Сильний штучний інтелект— це система штучного інтелекту з широкими когнітивними можливостями людини. Перед обличчям незнайомих завдань потужна система штучного інтелекту находить рішення без втручання людини.

Хоча люди досі не дійшли єдиного висновку, чи штучний інтелект таки приносить користь чи навпаки робить наше життя гіршим.

2.4. Висновки до розділу

Отже, у цьому розділі ми отримали інформацію і розяснення щодо стеку технлогій при подальшій розробці додатку.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Загальна структура програмного продукту

Розробка ПЗ для нашого додатку ділиться на 3 частини:

- Бекенд — це частина, яка працює з базою даних і обробляє базову бізнес-логіку. Зв'язок з цією частиною здійснюється через веб-додаток (інтерфейс програмного забезпечення веб-додатків) за допомогою HTTP-запитів;
- Фронтенд – клієнтська частина проекту, якою керує користувач. Важливими частини є: зручність використання, дизайн, та інформаційний вміст;
- Бот – частина проекту, яка включає в себе штучний інтелект, а також робота з google analytic services.

Архітектура бекенду – оніон архітектура показана нижче (Рис. 3.1):

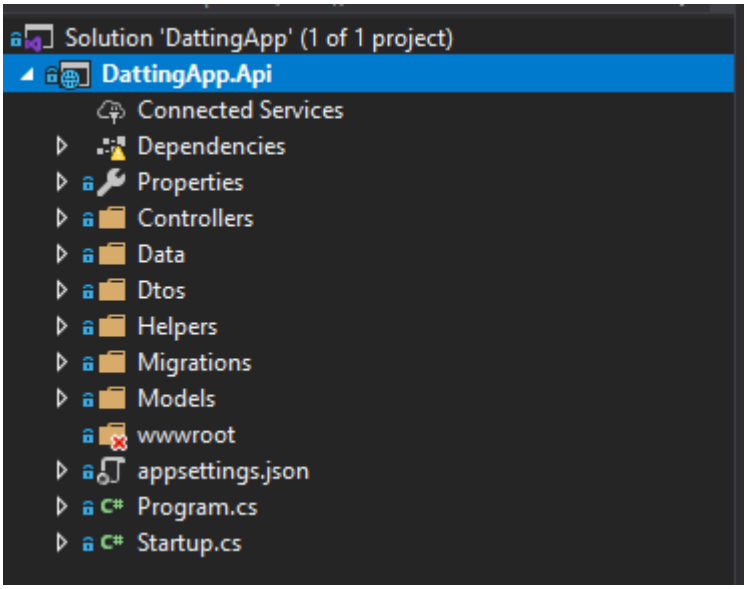

Рис 3.1 Структура серверної частини ПЗ

Давайте розглянемо кожен компонент цибульної архітектури (Рис 3.2):

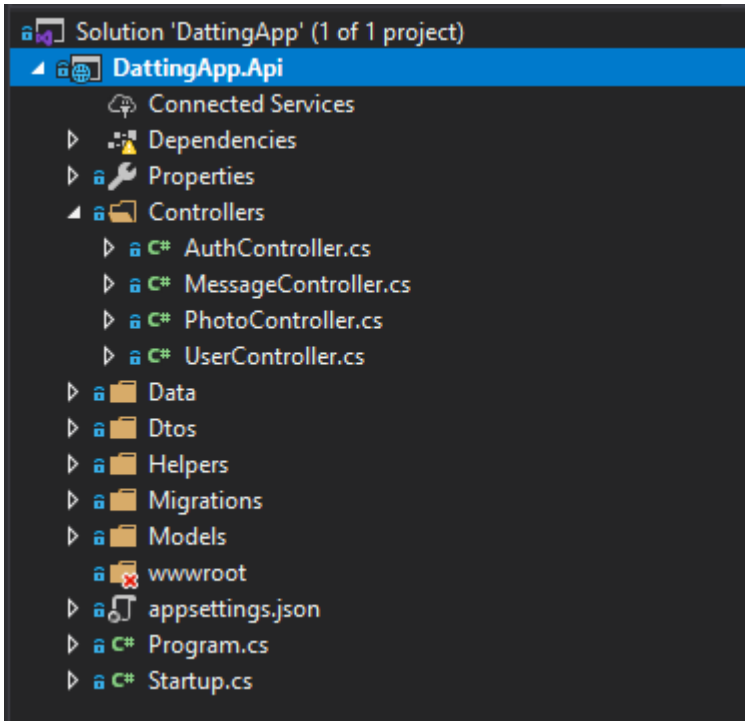

Рис 3.2 Зовнішній рівень архітектури

Зовнішний рівень - тут приймаються всі запити від бекенда, який є місцем для входу та спілкування з клієнтами.

Тут ми маємо перевірку моделі, базову бізнес-логіку та обробку даних від клієнта, потім дані переходять до наступного рівня.

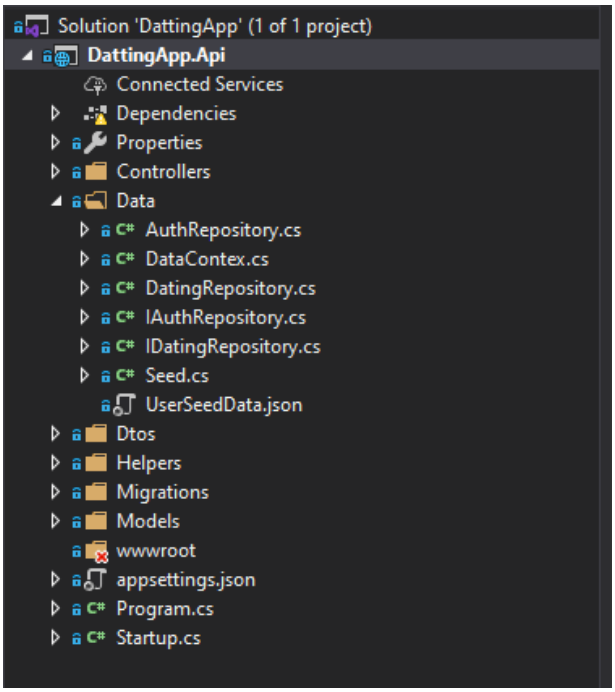

Рис 3.3 Рівень домейну проекту

Наступний рівень архітектури — домейн сховища. Тут знаходяться всі операції з базою даних, які відокремлені від базової бізнес-логіки.

Також використовуючи цю архітектуру, ми маємо два типи моделей: Dto та entity. Dtos використовуються для обміну даними між сервером і клієнтом, і вони також включають перевірку моделі для обробки об'єктів. Вони тут (Рис 3.4):

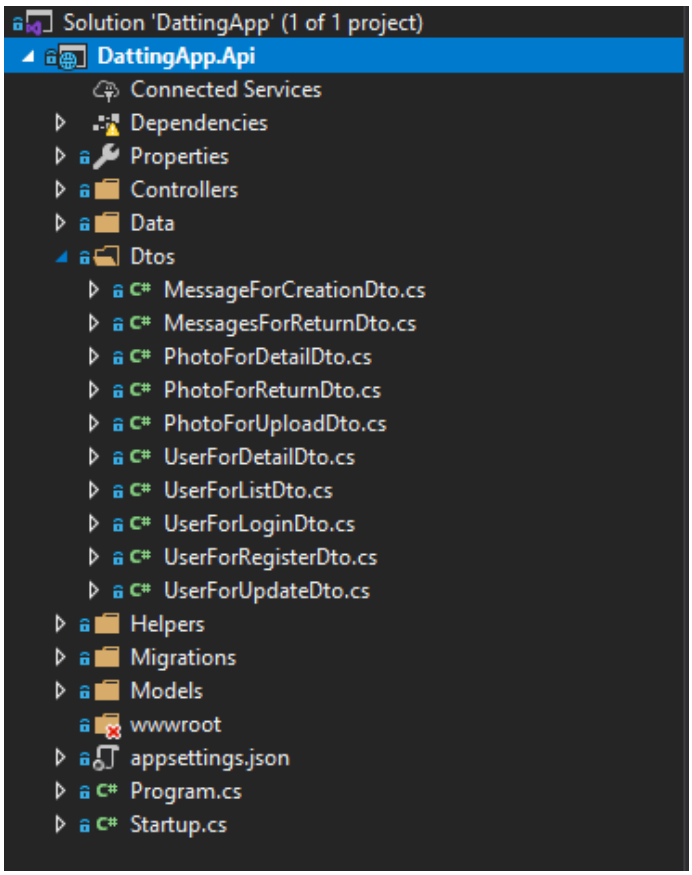

Рис 3.4 Структура ViewModel

Щоб зберегти та витягнути дані з бази даних, ми будемо використовувати entity модель (Рис. 3.5):

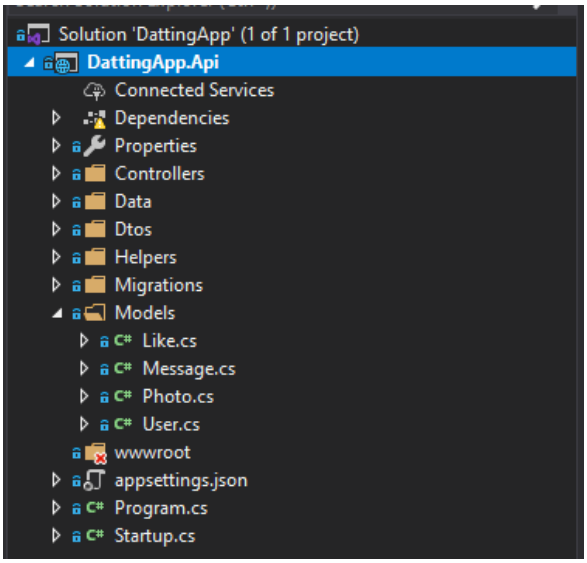

Рис 3.5 Рівень моделей бази даних

Клієнтська архітектура – проста компонентська архітектура Angular зображена на малюнку (Рис. 3.6):

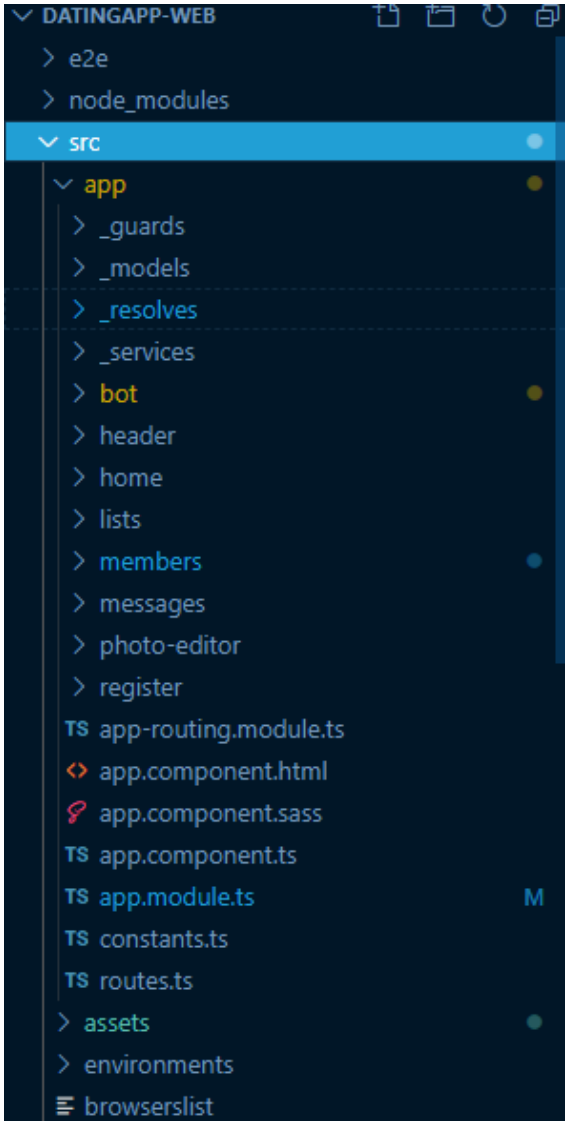

Рис 3.6 Архітектура клієнта

Тут можна побачити, що кожен елемент розбитий на певні компоненти.

Структуру проекту бота (Рис. 3.7):

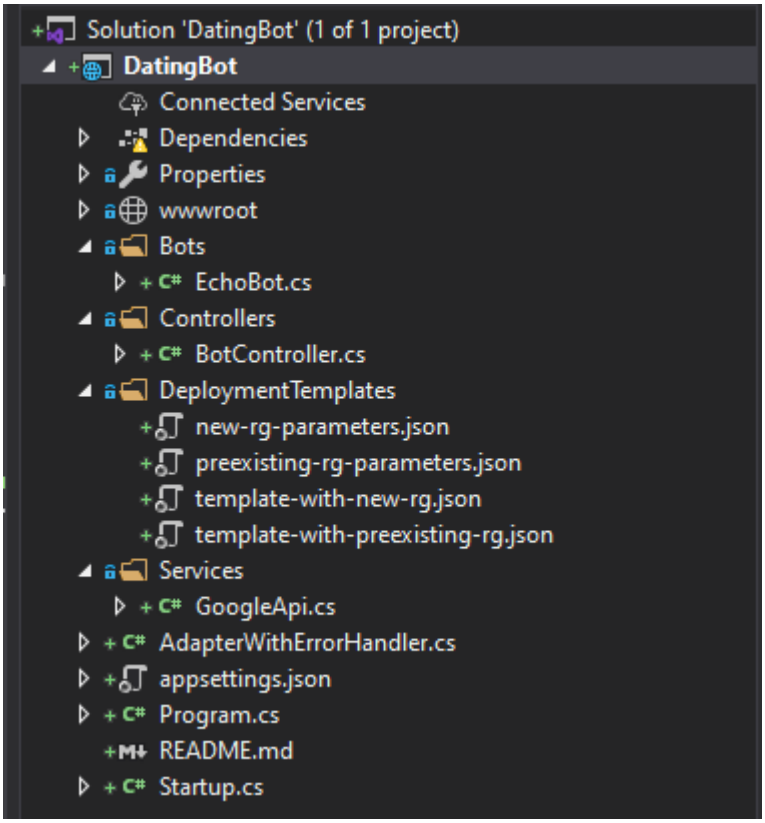

Рис 3.7 Архітектура бот проекту

3.2 Огляд інших бібліотек і фреймворків

Розглянемо бібліотеки та фреймворки, які будуть використовуватись в розробці продукту:

- Automapper – це картограф об’єкта. Відображення об’єкт-об’єкт працює шляхом перетворення вхідного об’єкта одного типу у вихідний об’єкт іншого типу. Що робить AutoMapper цікавим, так це тим, що він надає деякі цікаві конвенції, які допомагають вирішити брудну роботу, пов’язану з тим, як зіставити тип A з типом B. Поки тип B відповідає встановленій конвенції AutoMapper, для відображення двох типів потрібна майже нульова конфігурація;
- Entity Framework Core – це фреймворк з відкритим вихідним кодом, легкий, розширюваний і кросплатформна версія технології доступу до даних Entity Framework. Entity Framework — це структура об’єктного/реляційного відображення (O/RM). Це вдосконалення ADO.NET, яке надає розробникам автоматизований механізм доступу та зберігання даних у базі даних. EF Core

призначений для використання з додатками .NET Core. Однак його також можна використовувати зі стандартними додатками на основі .NET 4.5+..

- Swagger – потужність інструментів Swagger починається зі специфікації OpenAPI — промислового стандарту для проектування RESTful.
- API Bootstrap 4 і ngx-bootstrap – бібліотека, яка містить готові компоненти макету, які легко застосувати у фреймворку Angular.
- RxJS - є однією з найпопулярніших бібліотек у веб-розробці сьогодні. Пропонуючи потужний, функціональний підхід до роботи з подіями та з точками інтеграції у зростаючу кількість фреймворків, бібліотек і утиліт, випадок для вивчення Rx ще ніколи не був таким привабливим. Поєднайте це з можливістю використовувати свої знання практично на будь-якій мові, добре володіючи реактивним програмуванням і тим, що воно може запропонувати, здається, неважким.

3.3 Написання програмного забезпечення для проекту

Розробка програмного забезпечення включає в себе написання кінцевих точок для проекту, наприклад, з урахуванням шляху, через який відбувається авторизація в системі.

Вигляд точки входу для бота (Рис. 3.8):

```
[HttpPost( template: "login" )]
0 references | 0 requests | 0 exceptions
public async Task<IActionResult> Login (UserForLoginDto userForRegister ) {
    var userFromRepo = await _repo.Login( userForRegister.Username.ToLower(), userForRegister.Password );

    if ( userFromRepo == null ) {
        return Unauthorized();
    }

    var claims = new[] {
        new Claim( type: ClaimTypes.NameIdentifier, value: userFromRepo.Id.ToString() ),
        new Claim( type: ClaimTypes.Name, value: userFromRepo.UserName ),
    };

    var key = new SymmetricSecurityKey( key: Encoding.UTF8.GetBytes( _config.GetSection( key: "AppSettings:Token" ).Value ) );

    var creds = new SigningCredentials( key, algorithm: SecurityAlgorithms.HmacSha512Signature );

    var tokenDescriptor = new SecurityTokenDescriptor {
        Subject = new ClaimsIdentity( claims ),
        Expires = DateTime.Now.AddDays( 1 ),
        SigningCredentials = creds
    };

    var tokenHandled = new JwtSecurityTokenHandler();

    var token = tokenHandled.CreateToken( (tokenDescriptor) );


    return Ok( new {
        token = tokenHandled.WriteToken( token ),
        user = userFromRepo
    } );
}
```

Рис 3.8 Огляд коду бота

Можемо зрозуміти, що виклик до системи авторизації буде тригернутий методом «Login» використовуючи метод HTTP Post, який ми передали до моделі DTO, як показано нижче. (Рис. 3.9):

```
1  using System.ComponentModel;
2  using System.ComponentModel.DataAnnotations;
3
4  namespace DattingApp.Api.Dtos
5  {
       1 reference
6      public class UserForLoginDto
7      {
8          [Required]
           1 reference | 0 exceptions
9          public string Username { get; set; }
10         [Required]
11         [StringLength(maximumLength: 8, MinimumLength = 4, ErrorMessage = "You mast specify correct password")]
           1 reference | 0 exceptions
12         public string Password { get; set; }
13     }
14 }
```

Рис 3.9 Огляд ViewModel системи

На цьому рівні ми можемо побачити, що саме тут відбувається відповідний вхід у базовій системі бізнес-логіки та системі повідомлень.

Відповідно до моделі видно, які поля потрібно заповнити конкретним методом, і відповідну перевірку.

Процес збереження даних показано на малюнку (Рис. 3.10):

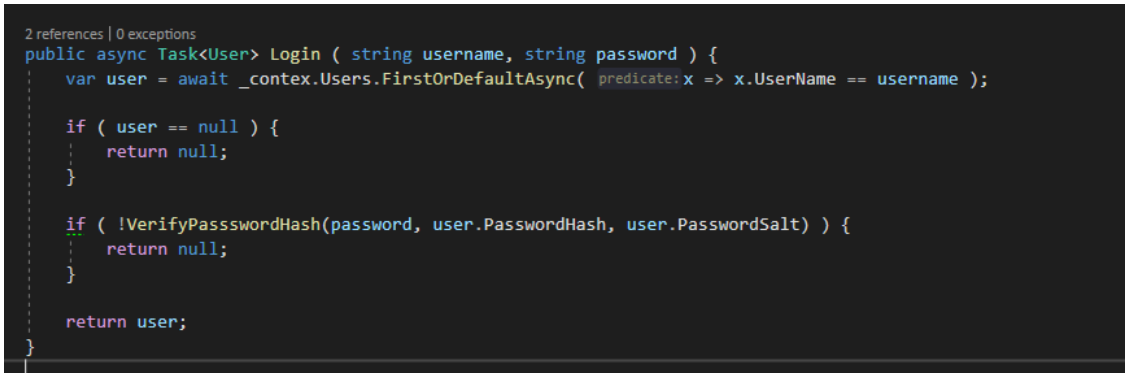

Рис 3.10 Бізнес логіка процесу авторизації серверної частини

Ми використовуємо базу даних проекту розробки.

Розмітка для авторизації зображена на малюнку (Рис. 3.11).

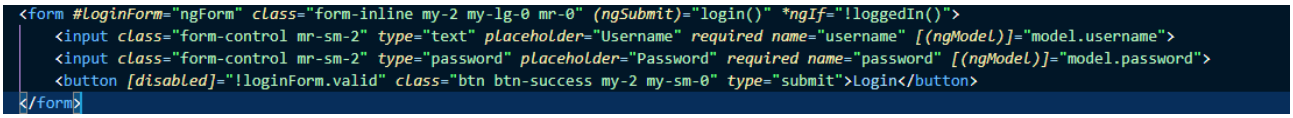

Рис 3.11 Огляд розмітки клієнта

На ньому зображено форму, через яку користувач вводить свої дані, і обробку валідності даних введених користувачем.

Рис 3.12 Огляд методу авторизації клієнта

Відправивши форму, яку заповнив користувач, дані формуються в правильні моделі і формується HTTP-запит для зв’язку з сервером.

```
login(model: any) {
  return this.http.post(environment.baseUrl + 'auth/login', model).pipe(
    map((response: any) => {
      const user = response;
      if (user) {
        localStorage.setItem('token', user.token);
        localStorage.setItem('userId', user.user.id);
        this.decodedToken = this.jwtHelper.decodeToken(user.token);
      }
    })
  );
}
```

Рис 3.13 Огляд endpoint клієнту

Ось є результат проходження дій користувачів через усі етапи.

3.4. Висновки до розділу.

В цьому розділі ми побачили основні моменти імплементації додатку, його роботи та функції зі всіх сторін.

РОЗДІЛ 4. ЕКСПЕРЕМЕНТАЛЬНА ЧАСТИНА

4.1 Вимоги до обладнання

Сьогодні потреби в обладнанні віднесені до інвалідів. Функціональні вимоги змінюють такі функції системи, як безпека, надійність, продуктивність, стійкість, зниження версії та доступність. Вони знаходяться в різних областях, таких як слабкі сторони або обмеження системи. Вони забезпечують функціональність і продуктивність усіх систем. Оцінка системи не може бути незалежною від типу завдань для внутрішніх користувачів і читачів або може бути пов'язана з іншими відповідними органами.

Вимоги до продукту:

– комп'ютерний сервер з базою даних, повинен мати процесор з частотою не менше 3,9 ГГц, жорсткий диск ємністю 1 ТБ і не менше 16 Гб пам'яті. Операційна система Windows/Linux з MS SQL Server і ASP.NET Core 5.1 і вище;
– Швидкість Інтернету має бути не менше 100 Мбіт/с.

Вимоги до процесу:

– можливість одночасної роботи в системі ~ 500 клієнтів;
– використання протоколу HTTP/HTTPS;
– можливість обробляти близько 100 запитів одночасно.

Зовнішні вимоги:

– Мова інтерфейсу – англійська;
– Операційна система Windows/Linux;
– Локальна мережа, вихід в Інтернет;
– Комп'ютер-сервер;
– Швидкість Інтернету має бути не менше 100 Мбіт/с.
– Тип мережі: Ethernet або Fast Ethernet;

4.2 Опис інтерфейсу користувача

Інтерфейс користувача починається з вікна авторизації (Рис 4.1), де користувач може створити новий акаунт або ввійти вже в існуючий.

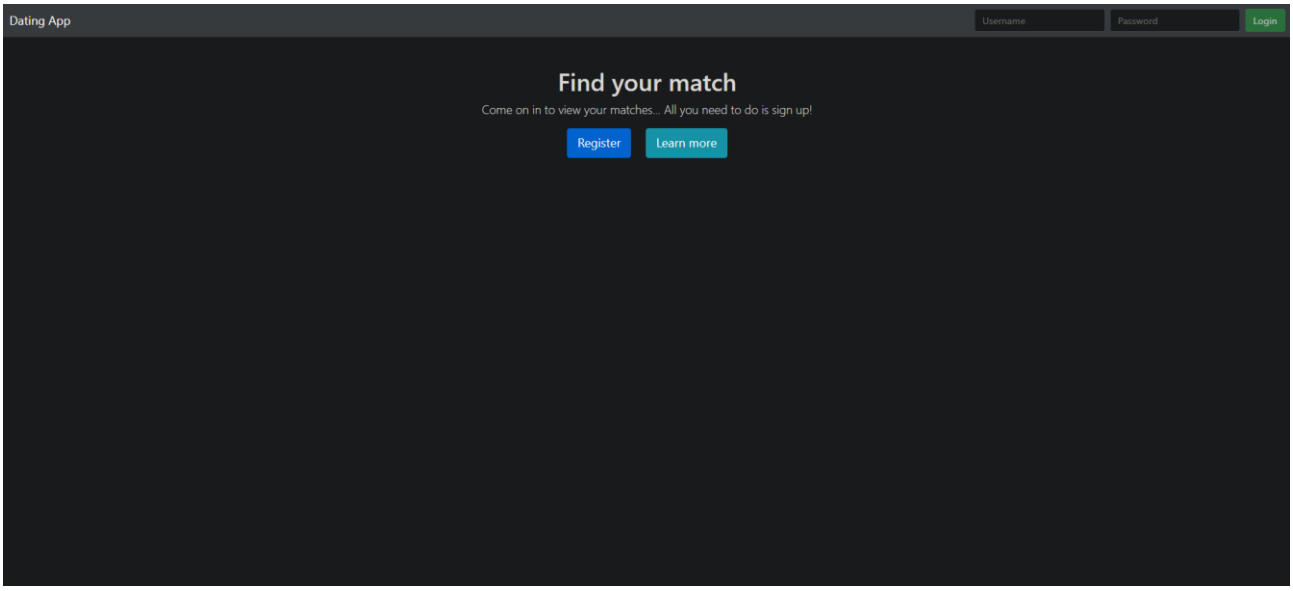

Рис 1.4 Вікно авторизації користувача

Якщо ви новий користувач, вам потрібно зареєструватися (рисунок 4.2) і ввести свої дані у відповідні поля.

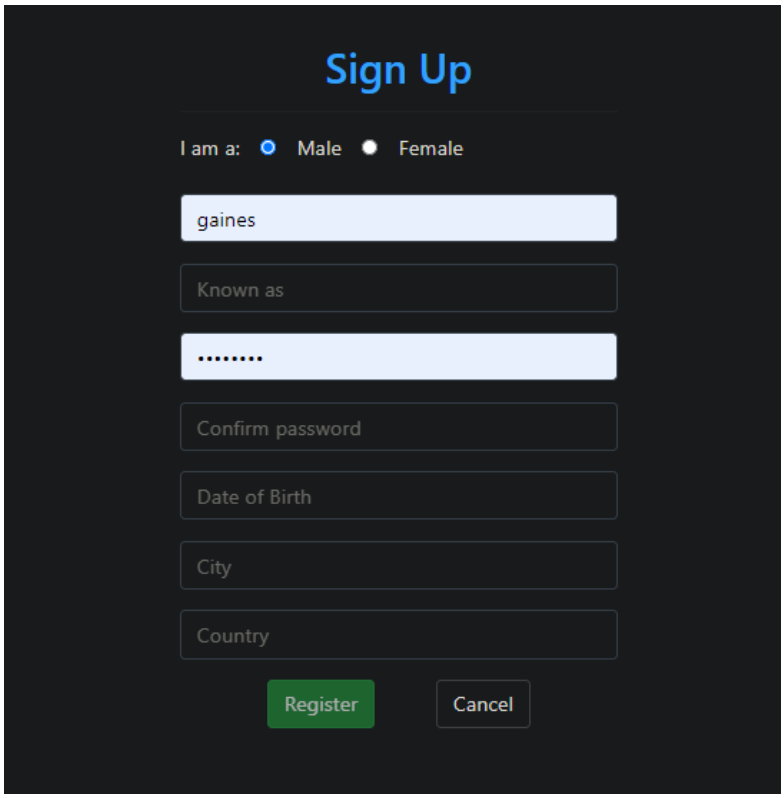

Рис 4.2 Вікно реєстрації

Після успішної авторизації ми попадаємо на сторінку пошуку (Рис 4.3), де можемо вказати необхідні параметри пошуку.

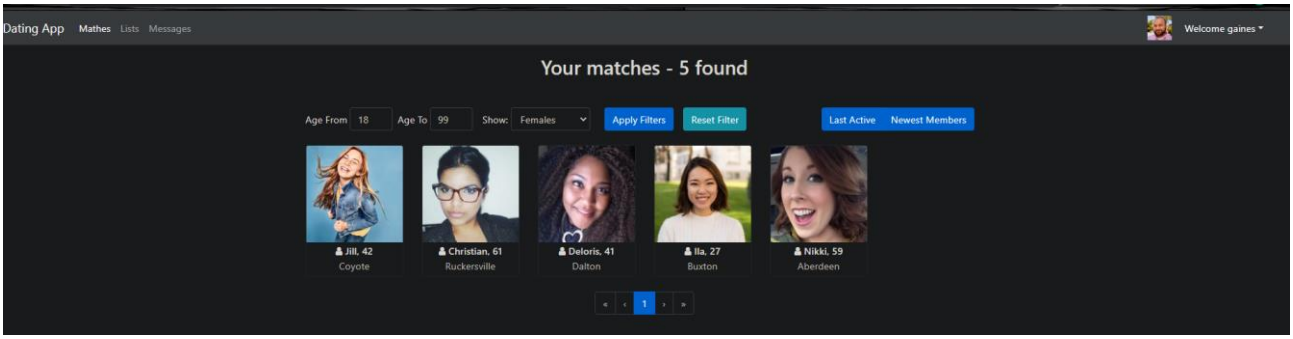

Рис 4.3 Сторінка пошуку

Через меню ми можемо зайди до списку вподобаних людей у системі (Рис 4.4) або до месседжера (Рис 4.5).

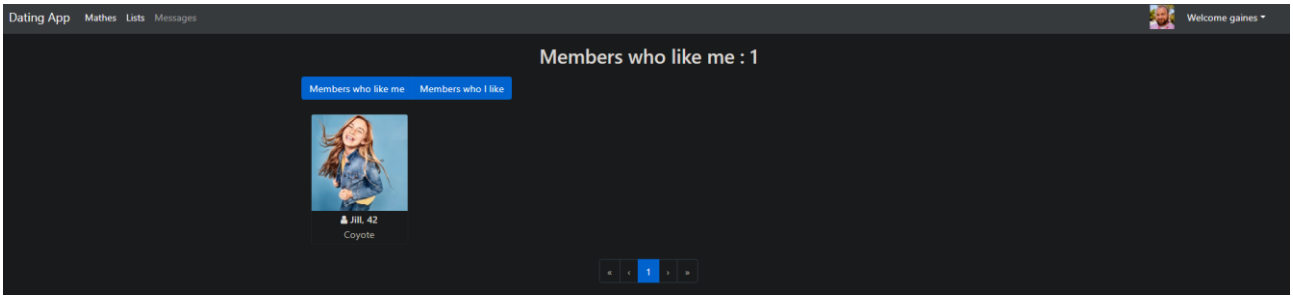

Рис. 4.4 Сторінка симпатій

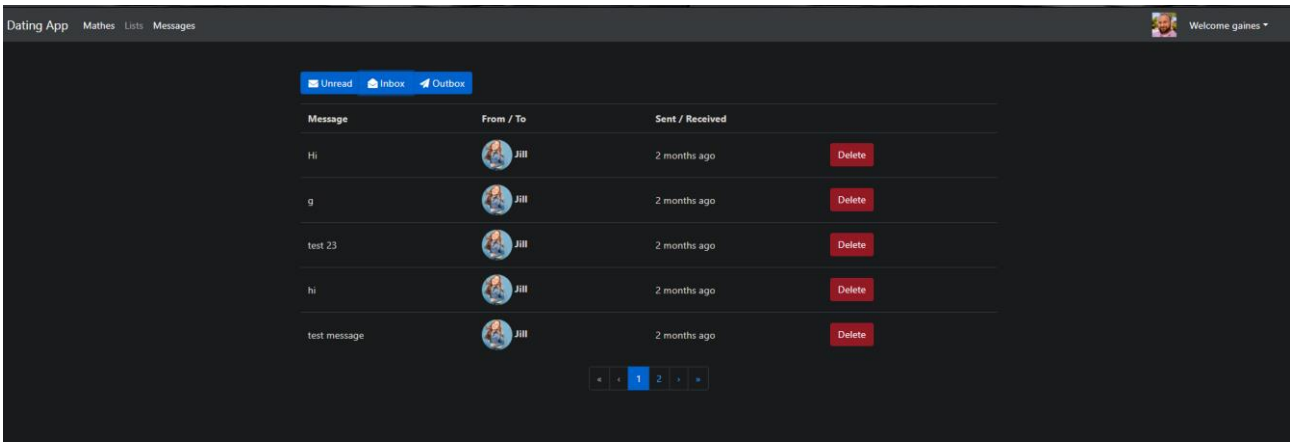

Рис 4.5 Сторінка месседжа

Також в системі існує зміна в любий момент приватної інформації (рис. 4.6), а також завантажування фотографій чи створення альбому (рис. 4.7).

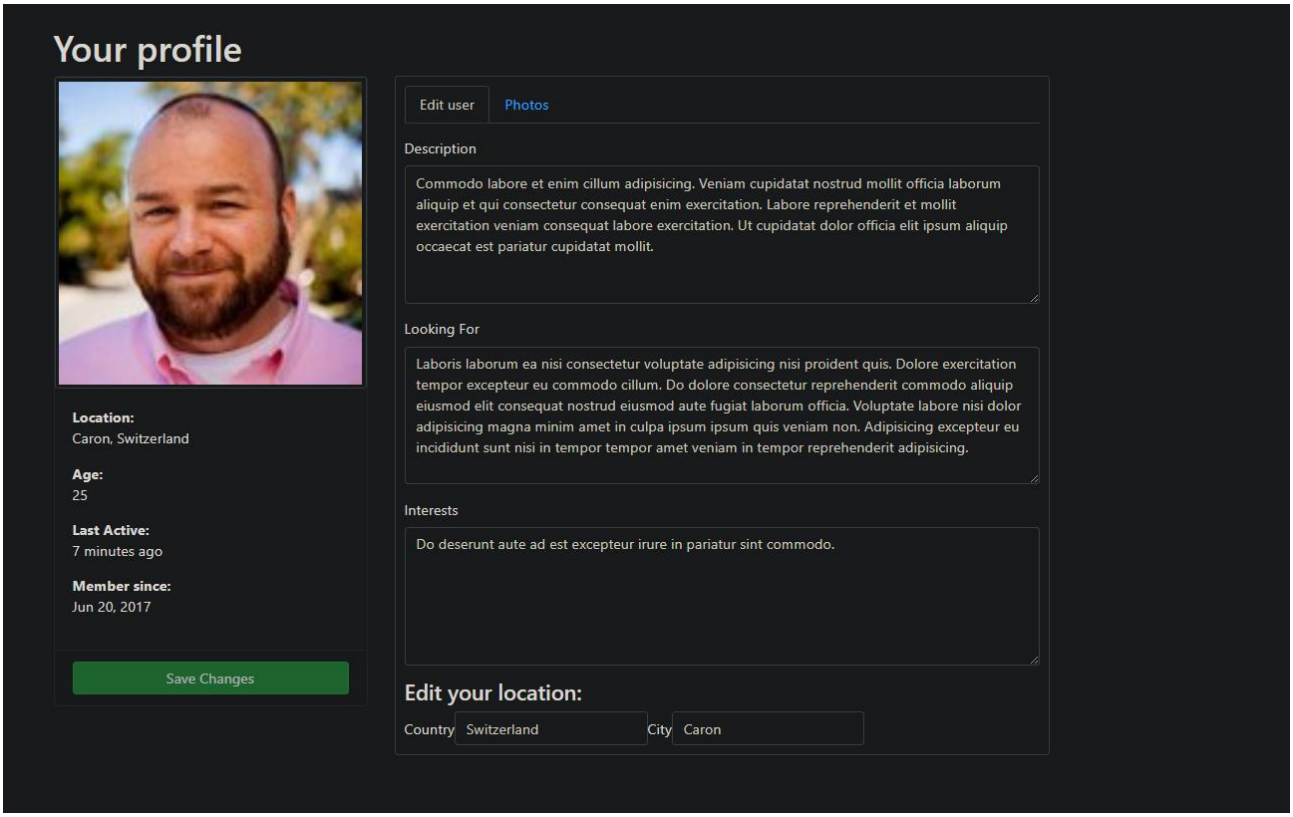

Рис 4.6 Сторінка профілю користувача

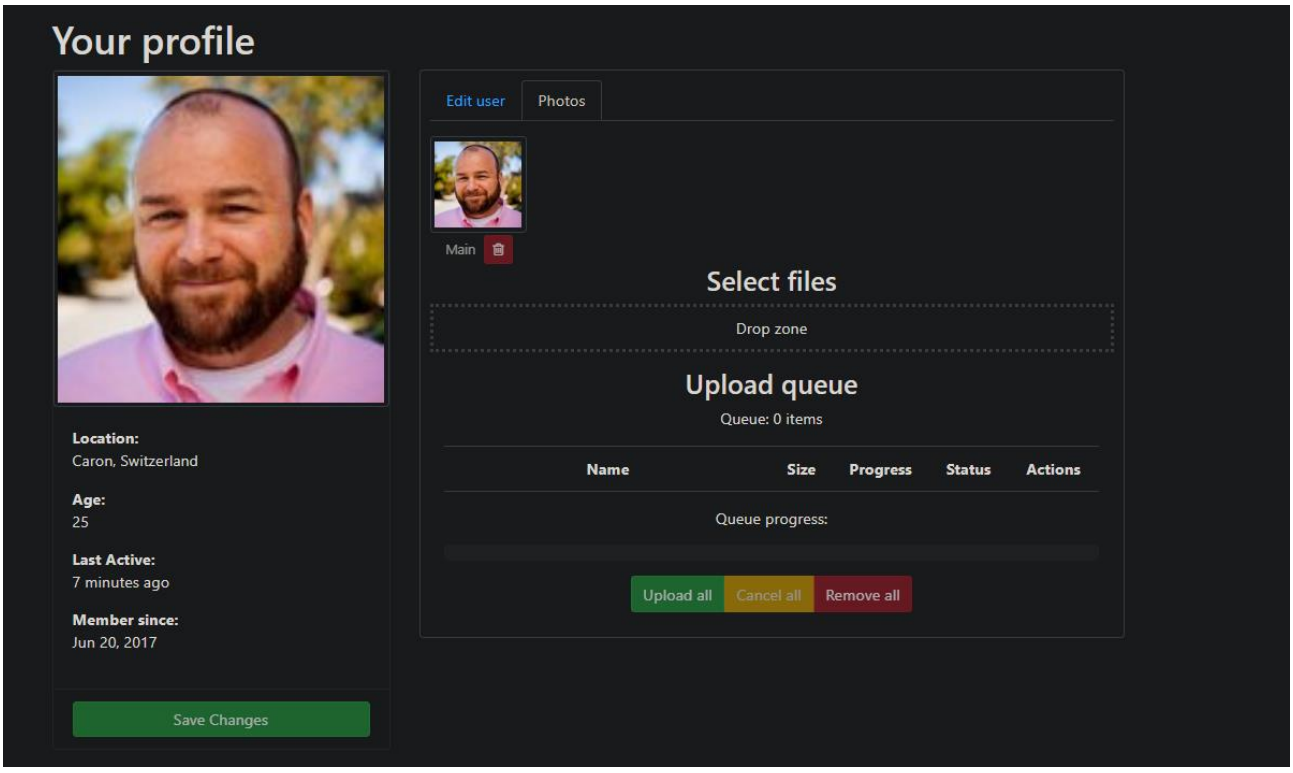

Рис 4.7 Фото-галерея

Також з цієї сторінки ви можете перейти до спілкування з ботом(Рис 4.8).

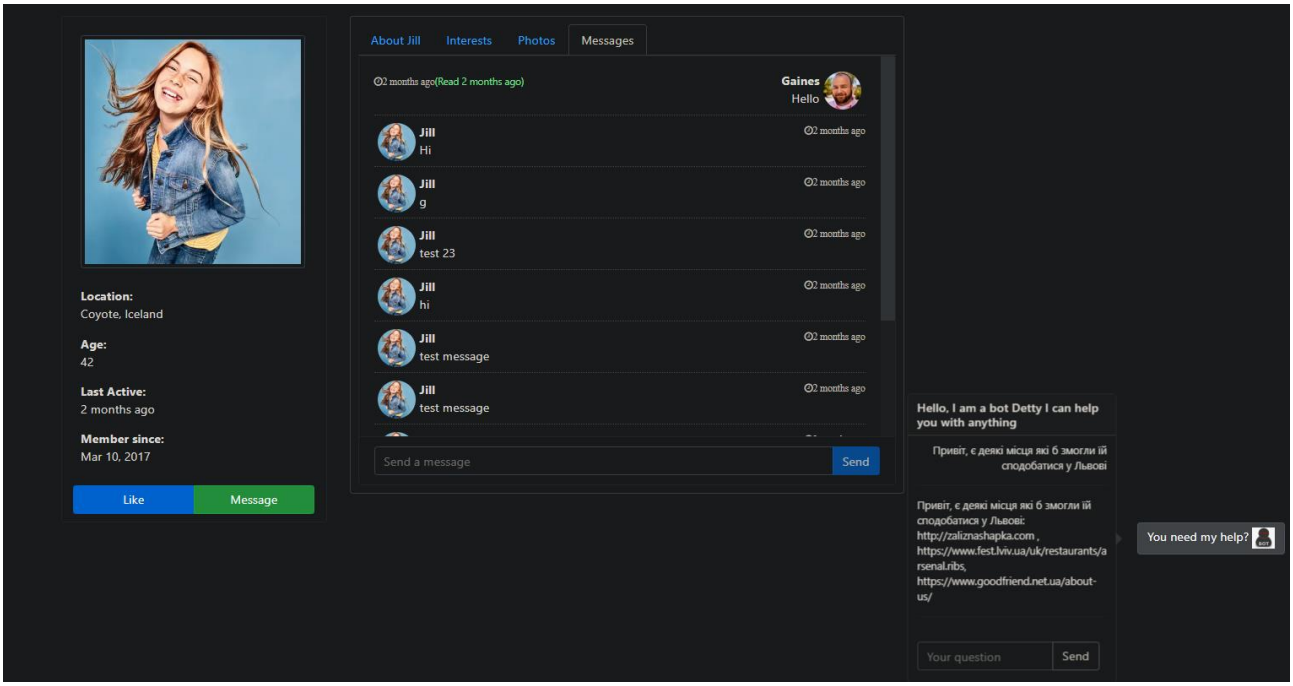

Рис 4.8 Користувацький інтрефейс бота

4.3 Огляд інтерфейсу сервера та інтерфейсу бази даних

Бібліотека Swagger використовується для опису документації та перевірки серверної частини (Рис 4.9), де доступні всі наявні методи, які нам надає розроблене API та відповідні моделі (Рис 4.10), які використовуються для взаємодії з методами серверу.

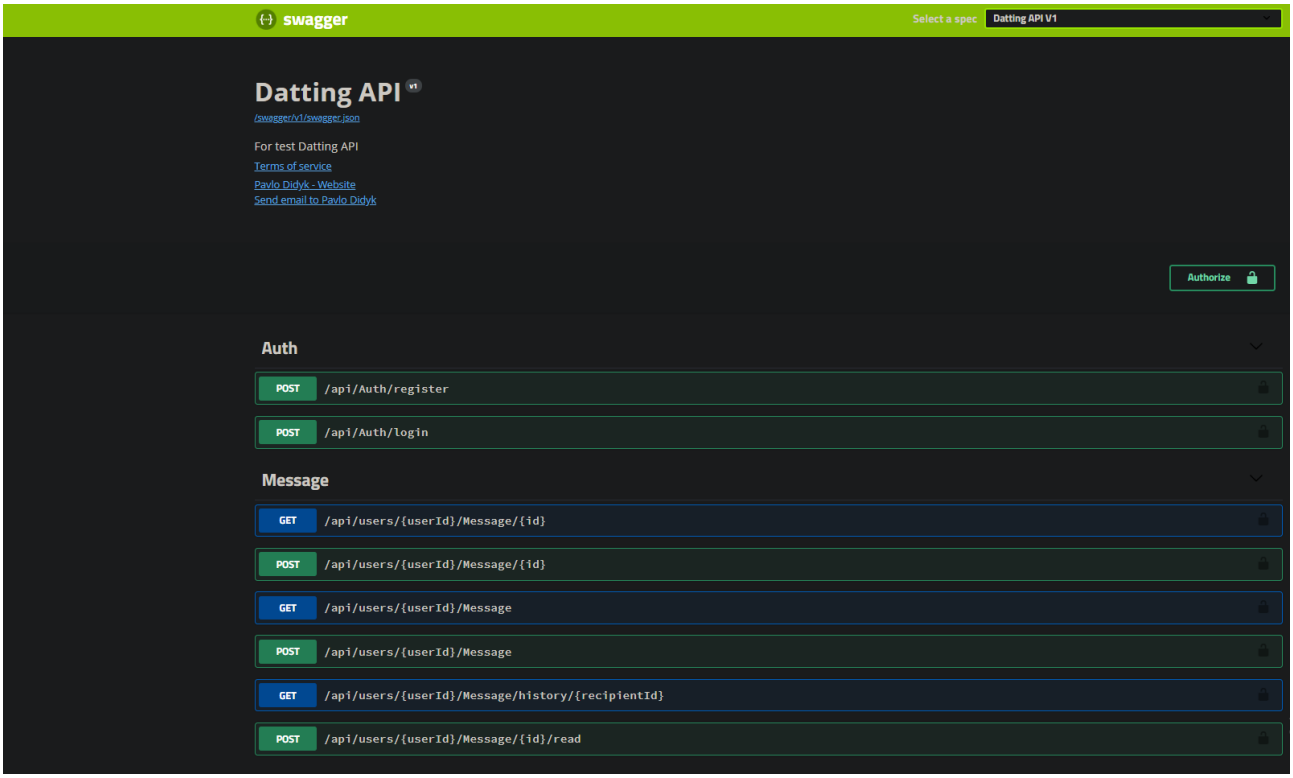

Рис 4.9 Огляд бібліотеки Swagger

Рис 4.10 Огляд моделей Swagger

Використання підходу CodeFirst в Entity framework з результатом, в якому можна побачити створені таблиці через Microsoft SQL Server Management Studio (Рис 4.11).

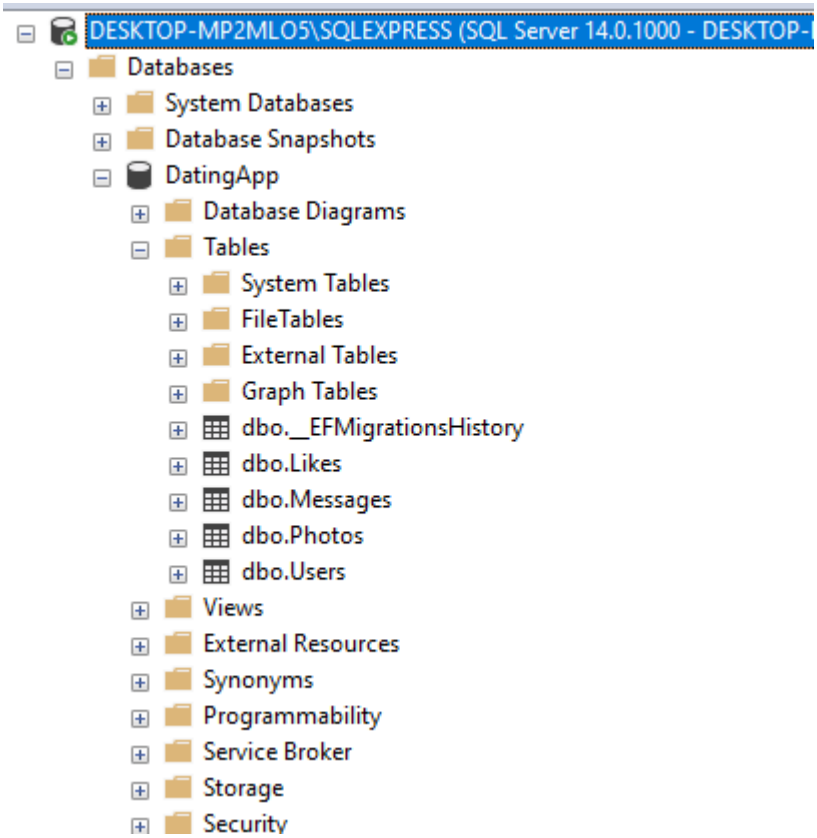

Рис 4.11 База даних

4.4 Висновки до розділу

У цьому розділі розглянуто роботу системи, а також показано роботу з інтерфейсом серверної частини.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1 Опис ідеї проекту

Аби спростити розуміння ідеї проекту наведено інфографіку проекту 5.1, яка містить усі необхідні дані.

Табл.5.1. Інформаційна карта проекту

Вид проекту	Розробка програмного забезпечення
Назва проекту	Соціальна мережа “4Students” з використатнням сучасних технологій
Компанія	Компанія Neadevis
Цілі та вимоги для розвитку	1. Удосконалити вже розроблену соціальну мережу 2. Підтримувати і оновлювати останні версії фреймворків 3. Вивести і підтримувати мережу онлайн 4. Почати рекламну компанію на залучення користувачів
Стислий опис проекту	Проект являє собою соціальну мережу, яка може працювати як окрема система, зі своєю реєстрацією, автентифікацією, особистою сторінкою, а також вбудованим месенджером який побудований з використанням Signalr.
Термін виконання	Демо – продукт за 6 місяців
Запланований бюджет	$ 24,000

5.2 Організаційний та фінансовий плани

Основними вимогами до розробників будуть базові знання мов програмування C# і TS, а також досвід комерційної розробки більше 2 років. Вимога кандидата – розуміти основи OOD, OOP, багатопоточності, WebServices, Angular та GUI розробки.

Розширення та комерціалізація розроблюваного продукту включає реалізацію певних планів шляхом ітеративної розробки програмного забезпечення за допомогою методології SCRUM. Стартовий капітал, який є необхідним для запуску процесів розробки, розраховується шляхом

додавання витрат для працюючого персоналу, а також супутніх ресурсів, обладнання, ліцензії та дозволи. Враховуючи поточну ситуацію в нашій країні, є можливість заощадження на оренді офісного приміщення. У поточній ситуації витратні ресурси включають наступне: 1. Стаціонарний комп'ютер x2 = $2 000.

2. Шість місяців роботи (стандартний 5-денний робочий тиждень) двох розробників = $ 20 000.

3. Ліцензійне ПО = $2 000

Таким чином, загальна вартість розробки програмного забезпечення складе - $24 000.

5.3 Ризики проекту

Початок проекту може зіткнутися з багатьма ризиками. Нижче наведено найважливіші з потенційних загроз.

Ризик можливостей. Можливий варіант того, що стартап- компанія не може вчасно розширити певні можливості на необхідному рівні.

Ризик проблемного дизайну. Можливість того, що дизайн продукту чи якісь конкретні послуги будуть не відповідними необхідним стандартам ефективності.

Економічний ризик. Успіх компанії чутливий до зовнішніх економічних факторів.

Законодавчий та політичний ризик (також "Регулюючий ризик"). Ризик полягає в тому, що законодавчі та політичні зміни призведуть до збільшення витрат на запуск.

Ризик управління. Ризики того, що команда, яка займається керівництвом не володіє достатньою кількістю навичок та досвіду для реалізації підприємницьких планів.

Ризик закупівель. Ризики, пов'язані зі здатністю стартапів купувати необхідну кількість та ціну дефіцитних ресурсів.

Ризик дослідження. Існує ризик того, що якість оригінального дослідження, на якому базуються ключові припущення компанії, є значною мірою недосконалою.

Ризик обсягу / попиту. Фактичний ринковий попит на товари чи послуги не створює ризику очікуваних продажів.

Щоб уникнути перерахованих вище ризиків, необхідно проводити якісне дослідження ринку та попиту на продукцію, перевіряти юридичний зміст країни дистрибуції, наймати компетентних менеджерів та фінансовий персонал для контролю витрат та досягати високоякісного продукту швидше, ніж конкуренти.

5.4 Висновки до розділу

У цьому розділі описується план розробки та впровадження стартап-проекту. Він також визначає деякі проблеми, які можуть виникнути при його створенні, та способи їх вирішення або уникнення. Крім того, була розрахована вартість створення програмного забезпечення.

Як висновок цього розділу, стартап-проект може бути створений для отримання комерційного прибутку.

ВИСНОВКИ

Отримати повну модель програмного продукту можливо завдяки аналізу наявних відсутніх продуктів та загальної літератури в галузі дослідження даної магістерської роботи.

У цій магістерській роботі особлива увага приділяється розробці онглайн обміну пакетів даних, використовуючи веб сокети через фреймворк Signalr.

Незалежна архітектура клієнт-сервер дозволяє абстрагуватися від частини інтерфейсу користувача та бізнес-логіки розроблюваного продукту. Завдяки архітектурі типу Onion серверну частину можна легко та швидко розширити до інших версій або структур баз даних.

Розроблений в даній магістерській кваліфікаційній роботі програмний продукт в подальшому можу представляти хорошу комерційну перспективу, що означає можливість його існування як окремої та незалежної соціальної мережі для нових знайомств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цимбал Ю. В., Нейромережевий метод симетричного шифрування – Львів: Національний університет «Львівська політехніка», 2018.
2. Джонсон Джефф Умный дизайн. Простые приемы разработки пользовательських интерфейсов: Пер. с англ. – СПб. : Питер, 2012.
3. Дронов В. HTML 5, CSS3 и Web.2.0. Разработка современных Web-сайтов. –СПб.: БХВ- Петербург, 2011.
4. Катренко А.В., Пасічник В.В., Прийняття рішень: Теорія та практика: - Підручник. – Львів: «Новий Світ- 2000», 2013.
5. Катренко А.В. Системний аналіз: Навчальний посібник. - Львів: «Новий Світ- 2000», 2009.
6. Марманис Х., Бабенко Д., Алгоритми інтелектуального інтернету.- Пер.с англ.- СПб.: Символ- Плюс, 2011.
7. Морвиль П., Розенфельд Л., Информационные архитектура в Интернете, 3 Изд. Перев. с англ. .- СПб.: Символ- Плюс, 2010.
8. Мацяшек Л.А., Аналіз вимог і проектування систем. Розробка інформаційних систем. Перев. с англ.- «Вильямс» 2005.
9. Архипенко С. Нові технології в СППР, Сімферополь, 2008 – 94с.
10. Тролсон Е., Мова програмування C# і платформа .NET , Видавничий дім «Вільямс», Київ, 2013 – 1340с.
11. Федоров О., Microsoft Visual Studio, Microsoft, 2009 – 43с.
12. Аблазов В.І. Проектування баз даних SQL Server, 2014 – 104 с.
13. Ларман К. Застосування UML і шаблонів проектування, Видавничий дім «Вільямс», Київ, 2002 – 624 с.
14. Скит Дж. C# для профессионалов. Тонкости программирования, Видавничий дім «Вільямс», Київ, 2014 – 608 с.
15. Кеннеди Б., Муссиано Ч., HTML и XHTML. Детальне пояснення (HTML & HXTML. The Definitive Guide), Київ, 2018 – 456 с.
16. Richter J., CLR via C# (4th Edition), Видавничий дім «Вільямс», Київ, 2013 – 896с.

17. Інтернет стаття «Ангуляр» [електронний ресурс] : [Веб-сайт] - https://uk.wikipedia.org/wiki/AngularJS – Назва з екрану.

18. Документація по Entity Framework Core [електронний ресурс] : [Веб-сайт] – https://metanit.com/sharp/mvc5/23.1.php – Назва з екрану.

19. Патерни проектування [електронний ресурс] : [Веб-сайт] – https://docs.microsoft.com/ru-ru/aspnet/mvc – Назва з екрану.

20. Niklas Donges, наукова стаття «Pros and Cons of Neural Networks» [електронний ресурс] : [Веб-сайт] – https://towardsdatascience.com/hype-disadvantages-of-neural-networks – Назва з екрану.

Додаток А

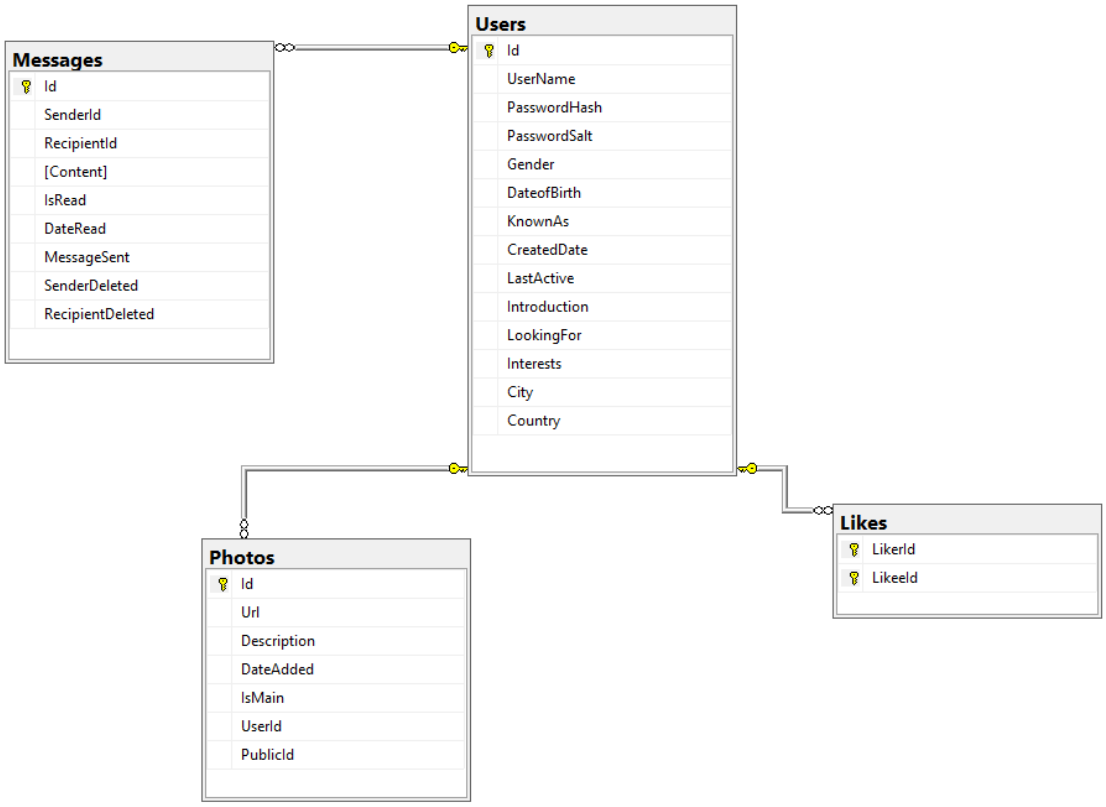

Рис А.1 Схема бази даних веб-аплікації

Додаток Б

Код файлу “UserController”

```
using System;
using System.Collections.Generic;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web.Http.Filters;
using AutoMapper;
using DattingApp.Api.Data;
using DattingApp.Api.Dtos;
using DattingApp.Api.Helpers;
using DattingApp.Api.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DattingApp.Api.Controllers
{
    [Authorize]
    [Route("api/[controller]")]
   [ServiceFilter(typeof(LogUserActivity))]
    [ApiController]
    public class UserController : ControllerBase
    {
        private readonly IDatingRepository _repo;
        private readonly IMapper _mapper;
```

```
public UserController(IDatingRepository repo, IMapper mapper)
{
    _repo = repo;
    _mapper = mapper;
}

[HttpGet]
public async Task<IActionResult> GetUsers([FromQuery]UserParams userParams)
{
    int curId;
    bool success = int.TryParse((User.FindFirst(ClaimTypes.NameIdentifier).Value), out curId);
    if (curId > 0)
    {
        userParams.UserId = curId;
    }
    var userFormRepo = await _repo.GetUser(curId);
    if (string.IsNullOrEmpty(userParams.Gender))
    {
        userParams.Gender = userFormRepo.Gender == "male" ? "female" : "male";
    }

    var users = await _repo.GetUsers(userParams);
    var usersToReturn = _mapper.Map<IEnumerable<UserForListDto>>(users);
```

```
        Response.AddPagination(users.CurrentPage, users.PageSize,
users.TotalCount, users.TotalPages);


        return Ok(usersToReturn);
    }


    [HttpGet("{id}")]
    public async Task<IActionResult> GetUser(int id)
    {
        var user = await _repo.GetUser(id);
        var userToReturn = _mapper.Map<UserForDetailDto>(user);
        return Ok(userToReturn);
    }


    [HttpPut("{id}")]
    public async Task<IActionResult> UpdateUser(int id, UserForUpdateDto
userForUpdate)
    {
        int curId;
        bool success =
int.TryParse((User.FindFirst(ClaimTypes.NameIdentifier).Value), out curId);
        if (success && curId != id)
        {
            return Unauthorized();
        }
        var curUser = await _repo.GetUser(id);
        _mapper.Map(userForUpdate, curUser);
```

```
            if (await _repo.SaveAll())
            {
                return NoContent();
            }
            throw new Exception($"Updaiting user {id} failed");
        }

        [HttpPost("{id}/like/{likeeId}")]
        public async Task<IActionResult> Like(int id, int likeeId)
        {
            int curId;
            bool success =
int.TryParse((User.FindFirst(ClaimTypes.NameIdentifier).Value), out curId);
            if (success && curId != id)
            {
                return Unauthorized();
            }

            if (await _repo.GetUser(likeeId) == null)
            {
               return NotFound("Like user wasn't found");
            }

            if (await _repo.GetLike(id, likeeId) != null)
            {
              return BadRequest("You already like this person");
```

```
            }

            var like = new Like
            {
                LikerId = id,
                LikeeId = likeeId
            };

            _repo.Add<Like>(like);

            if (await _repo.SaveAll())
            {
                return Ok();
            }

            return BadRequest("You can't like this person");
        }
    }
}
```

Код файлу “DatingRepository”

```
using DattingApp.Api.Helpers;
using DattingApp.Api.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;

namespace DattingApp.Api.Data
{
    public class DatingRepository : IDatingRepository
    {
        private DataContex _contex;

        public DatingRepository(DataContex ctx)
        {
            _contex = ctx;
        }

        public void Add<T>(T entity) where T : class
        {
            _contex.Add(entity);
        }

        public void Delete<T>(T entity) where T : class
        {
            _contex.Remove(entity);
        }

        public async Task<bool> SaveAll()
        {
            return await _contex.SaveChangesAsync() > 0;
```

```
    }

    public async Task<PagingList<User>> GetUsers(UserParams userParams)
    {
        var users = _contex.Users.Include(p => p.Photos).OrderByDescending(u =>
u.LastActive).AsQueryable();
        users = users.Where(u => u.Id != userParams.UserId);
        users = users.Where(u => u.Gender == userParams.Gender);

        if (userParams.Likers)
        {
            var userLikers = await GetUserLikes(userParams.UserId,
userParams.Likers);
            users = users.Where(u => userLikers.Contains(u.Id));
        }

        if (userParams.Likees)
        {
            var userLikees = await GetUserLikes(userParams.UserId,
userParams.Likers);
            users = users.Where(u => userLikees.Contains(u.Id));
        }

        if (userParams.MinAge != 18 || userParams.MaxAge != 99)
        {
            var minDob = DateTime.Today.AddYears(-userParams.MaxAge - 1);
            var maxDob = DateTime.Today.AddYears(-userParams.MinAge);
```

```
            users = users.Where(u => u.DateofBirth >= minDob && u.DateofBirth <=
maxDob);
        }


        if (userParams.OrderBy != null)
        {
            switch (userParams.OrderBy)
            {
                case "created":
                    users = users.OrderByDescending(u => u.CreatedDate);
                    break;
                default:
                    users = users.OrderByDescending(u => u.LastActive);
                    break;
            }
        }
        return await PagingList<User>.CreateAsync(users, userParams.PageNumber,
userParams.PageSize);
    }


    private async Task<IEnumerable<int>> GetUserLikes(int id, bool likers)
    {
        var user = await _contex.Users
            .Include(u => u.Likers)
            .Include(u => u.Likees)
```

```
            .FirstOrDefaultAsync(u => u.Id == id);
        if (likers)
        {
            return user.Likers.Where(u => u.LikeeId == id).Select(l => l.LikerId);
        }
        else
        {
            return user.Likees.Where(u => u.LikerId == id).Select(l => l.LikeeId);
        }
    }


    public async Task<User> GetUser(int id)
    {
        return await _contex.Users.Include(p => p.Photos).FirstOrDefaultAsync(u =>
u.Id == id);
    }

    public async Task<Photo> GetPhotoAsync(int id)
    {
        return await _contex.Photos.FirstOrDefaultAsync(p => p.Id == id);
    }

    public async Task<Photo> GetMainPhotoOfUserAsync(int userId)
    {
```

```
        return await _contex.Photos.Where(user => user.UserId ==
userId).FirstOrDefaultAsync(photo => photo.IsMain);
    }

    public async Task<Like> GetLike(int likerId, int likeeId)
    {
        return await _contex.Likes.FirstOrDefaultAsync(u => u.LikerId == likerId
&& u.LikeeId == likeeId);
    }

    public async Task<Message> GetMessage(int id)
    {
        return await _contex.Messages.FirstOrDefaultAsync(m => m.Id == id);
    }

    public async Task<PagingList<Message>> GetUserMessages(MessagesParams
messagesParams)
    {
        var messages = _contex.Messages
            .Include(u => u.Sender).ThenInclude(p => p.Photos)
            .Include(u => u.Recipient).ThenInclude(p => p.Photos)
            .AsQueryable();

        switch (messagesParams.MessageContainer)
        {
            case TypeOfMessages.Inbox:
```

```
                messages = messages.Where(u => u.RecipientId ==
messagesParams.UserId && u.RecipientDeleted == false);

                break;

            case TypeOfMessages.Outbox:

                messages = messages.Where(u => u.SenderId ==
messagesParams.UserId && u.SenderDeleted == false);

                break;

            default:

                messages = messages.Where(u => u.RecipientId ==
messagesParams.UserId

                                        && u.RecipientDeleted == false && u.IsRead ==
false);

                break;

        }


        messages = messages.OrderByDescending(d => d.MessageSent);

        return await PagingList<Message>.CreateAsync(messages,
messagesParams.PageNumber, messagesParams.PageSize);

    }


    public async Task<IEnumerable<Message>> GetMessageHistory(int userId, int
recipientId)

    {

        var messages = await _contex.Messages

            .Include(u => u.Sender).ThenInclude(p => p.Photos)

            .Include(u => u.Recipient).ThenInclude(p => p.Photos)

            .Where(m => m.RecipientId == userId && m.SenderId == recipientId &&
m.RecipientDeleted == false ||
```

```
                    m.RecipientId == recipientId && m.SenderId == userId &&
m.SenderDeleted == false)
                .OrderByDescending(m => m.MessageSent)
                .ToListAsync();
            return messages;
        }
    }
}
```

Код файлу “EchoBot”

```
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Schema;

namespace DatingBot.Bots
{
    public class EchoBot : ActivityHandler
    {
        protected override async Task
OnMessageActivityAsync(ITurnContext<IMessageActivity> turnContext,
CancellationToken cancellationToken)
        {
            var replyText = $"Echo: {turnContext.Activity.Text}";
            var googleApi = new Services.GoogleApi();
            googleApi.GetResultFromGoogle();
```

```
            await turnContext.SendActivityAsync(MessageFactory.Text(replyText,
replyText), cancellationToken);
        }


        protected override async Task
OnMembersAddedAsync(IList<ChannelAccount> membersAdded,
ITurnContext<IConversationUpdateActivity> turnContext, CancellationToken
cancellationToken)
        {
            var welcomeText = "Hello and welcome!";
            foreach (var member in membersAdded)
            {
                if (member.Id != turnContext.Activity.Recipient.Id)
                {
                    await
turnContext.SendActivityAsync(MessageFactory.Text(welcomeText, welcomeText),
cancellationToken);
                }
            }
        }
    }
}
```

Код файлу “BotController”

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
```

```
namespace DatingBot.Controllers
{
    [Route("api/messages")]
    [ApiController]
    public class BotController : ControllerBase
    {
        private readonly IBotFrameworkHttpAdapter Adapter;
        private readonly IBot Bot;

        public BotController(IBotFrameworkHttpAdapter adapter, IBot bot)
        {
            Adapter = adapter;
            Bot = bot;
        }

        [HttpPost, HttpGet]
        public async Task PostAsync()
        {
            await Adapter.ProcessAsync(Request, Response, Bot);
        }
    }
}
```

Код фалу “GoogleApi”

```
using System;
using System.Collections;
using Hammock.Web;
```

```
using System.Collections.Generic;
using Hammock;
using Newtonsoft.Json.Linq;
using RestSharp;
using SerpApi;
using RestRequest = RestSharp.RestRequest;
using RestClient = RestSharp.RestClient;

namespace DatingBot.Services
{
   public class GoogleApi
   {
      public void GetResultFromGoogle()
      {
         String apiKey = "AIzaSyCC92EmxdT1q98b1GK-FfY5Ju2U8k96x5o";
         Hashtable ht = new Hashtable();
         ht.Add("q", "Coffee");
         ht.Add("location", "Lviv Oblast, Ukraine");
         ht.Add("hl", "uk");
         ht.Add("gl", "ua");
         ht.Add("google_domain", "google.com.ua");

         try
         {
            GoogleSearchResultsClient client = new
GoogleSearchResultsClient(apiKey);
```

```
            JObject data = client.GetJson();
            JArray results = (JArray)data["organic_results"];
            var a = results;
        }
        catch (Exception ex)
        {
            Console.WriteLine("Exception:");
        }
    }

}
```

}

Код файлу “member-edit.component.html”

```
<div class="container mt-4">
  <div class="row">
    <div class="col-sm-4">
      <h1>Your profile</h1>
    </div>
    <div class="col-sm-8">
      <div *ngIf="editForm.dirty" class="alert alert-info">
        <strong>Information:</strong> You made changes, all unsaved changes will be lost!
      </div>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-4">
      <div class="card">
        <img src="{{user?.photoUrl || '../../../../../assets/user.png'}}" alt="{{user?.knownAs}}" class="card-img-
top img-thumbnail">
        <div class="card-body">
          <div>
            <strong>Location:</strong>
            <p>{{user?.city}}, {{user?.country}}</p>
          </div>
          <div>
            <strong>Age:</strong>
```

```
          <p>{{user?.age}}</p>
        </div>
        <div>
          <strong>Last Active:</strong>
          <p>{{user?.lastActive | timeAgo}}</p>
        </div>
        <div>
          <strong>Member since:</strong>
          <p>{{user?.createdDate | date}}</p>
        </div>
      </div>
      <div class="card-footer">
        <button [disabled]="!editForm.dirty" form="editForm" class="btn btn-success btn-
block">Save Changes</button>
      </div>
    </div>
  </div>
  <div class="col-sm-8">
    <div class="tab-panel">
      <tabset>
        <tab heading="Edit user">
          <form #editForm="ngForm" id="editForm" (ngSubmit)="updateUser()">
          <label class="mt-3" for="introduction">Description</label>
          <textarea class="form-
control" name="introduction" rows="6" [(ngModel)]="user.introduction"></textarea>
          <label class="mt-3" for="lookingFor">Looking For</label>
          <textarea class="form-
control" name="lookingFor" rows="6" [(ngModel)]="user.lookingFor"></textarea>
          <label class="mt-3" for="interests">Interests</label>
          <textarea class="form-control" name="interests" rows="6" [(ngModel)]="user.interests"></textarea>
          <h4 class="mt-3">Edit your location:</h4>
          <div class="form-inline">
            <label for="country">Country</label>
            <input class="form-control" type="text" name="country" [(ngModel)]="user.country">
            <label for="city">City</label>
            <input class="form-control" type="text" name="city" [(ngModel)]="user.city">
          </div>
          </form>
        </tab>
        <tab heading="Photos">
          <app-photo-editor [photos]="user.photos"></app-photo-editor>
        </tab>
      </tabset>
    </div>
  </div>
 </div>
</div>
```

Код файлу “member-edit.component.ts”

```
import { Component, OnInit, OnDestroy, ViewChild, HostListener } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
```

```
import { ToastrService } from 'ngx-toastr';
import { User } from 'src/app/_models/user';
import { Subscription } from 'rxjs';
import { NgForm } from '@angular/forms';
import { UserService } from 'src/app/_services/user.service';
import { AuthService } from 'src/app/_services/auth.service';

@Component({
  selector: 'app-member-edit',
  templateUrl: './member-edit.component.html',
  styleUrls: ['./member-edit.component.sass']
})
export class MemberEditComponent implements OnInit, OnDestroy {
  user: User;
  private _subscriptions: Subscription = new Subscription();
  @ViewChild('editForm', { static: true }) editForm: NgForm;
  @HostListener('window:beforeunload', ['$event'])
  unloadNotification($event: any) {
    if (this.editForm.dirty) {
      $event.returnValue = true;
    }
  }
  constructor(private toastr: ToastrService, private route: ActivatedRoute, private userService: UserService,
    private authService: AuthService) { }

  ngOnInit() {
    this._subscriptions.add(
      this.route.data.subscribe(data => {
        this.user = data['user'];
      })
    );
  this.userService.mainPhoto.next(this.user.photoUrl);
  this._subscriptions.add(
    this.userService.mainPhoto.subscribe(url => this.user.photoUrl = url)
  );
  }
  ngOnDestroy(): void {
    this._subscriptions.unsubscribe();
  }

  updateUser() {
    this._subscriptions.add(
      this.userService.updateUser(this.authService.decodedToken.nameid, this.user).subscribe($ => {
        this.toastr.success('You successfull update your profile');
        this.editForm.reset(this.user);
      }, error => this.toastr.error(error)));
  }

}
```

Код файлу “user.service.ts”

```
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environment';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Observable, BehaviorSubject } from 'rxjs';
import { User } from '../_models/user';
import { PaginationResult } from '../_models/pagination-result';
import { map } from 'rxjs/operators';
import { Constants } from '../constants';
import { Message } from '../_models/message';


@Injectable({
  providedIn: 'root'
})
export class UserService {
  baseUrl = environment.baseUrl;
  mainPhoto = new BehaviorSubject(null);
  constructor(private http: HttpClient) { }

  getUsers(page?, itemsPerPage?, userParams?, likesParam?): Observable<PaginationResult<User[]>> {
    const paginationResult: PaginationResult<User[]> = new PaginationResult<User[]>();
    let params = new HttpParams();

    if (page !== null && itemsPerPage !== null) {
      params = params.append('pageNumber', page);
      params = params.append('pageSize', itemsPerPage);
    }
    if (userParams != null) {
      params = params.append('minAge', userParams.minAge);
      params = params.append('maxAge', userParams.maxAge);
      params = params.append('gender', userParams.gender);
      params = params.append('orderBy', userParams.orderBy);
    }

    if (likesParam === Constants.Likers) {
      params = params.append('Likers', 'true');
    }

    if (likesParam === Constants.Likees) {
      params = params.append('Likees', 'true');
    }
    return this.http.get<User[]>(this.baseUrl + 'user', { observe: 'response', params })
      .pipe(
        map(Response => {
          paginationResult.result = Response.body;
          if (Response.headers.get('Pagination') != null) {
            paginationResult.pagination = JSON.parse(Response.headers.get('Pagination'));
          }
          return paginationResult;
```

```
      })
    );
}

getUser(id: Observable<User> {
  return this.http.get<User>(this.baseUrl + 'user/' + id);
}

updateUser(id: number, user: User) {
  return this.http.put(this.baseUrl + 'user/' + id, user);
}

like(id: number, likeeId: number) {
  return this.http.post(this.baseUrl + 'user/' + id + '/like/' + likeeId, {});
}

getMessages(id: number, page, itemsPerPage, messageContainer?) {
  const paginationResult: PaginationResult<Message[]> = new PaginationResult<Message[]>();
  let params = new HttpParams();

  params = params.append('MessageContainer', messageContainer);

  if (page !== null && itemsPerPage !== null) {
    params = params.append('pageNumber', page);
    params = params.append('pageSize', itemsPerPage);
  }

  return this.http.get<Message[]>(this.baseUrl + 'users/' + id + '/message', { observe: 'response', params })
    .pipe(
      map(response => {
        paginationResult.result = response.body;
        if (response.headers.get('pagination') !== null) {
          paginationResult.pagination = JSON.parse(response.headers.get('pagination'));
        }
        return paginationResult;
      })
    );
}

getUserHistory(id: number, recipientId: number) {
  return this.http.get<Message[]>(this.baseUrl + 'users/' + id + '/message/history/' + recipientId);
}

sentMessage(id: number, message: Message) {
  return this.http.post(this.baseUrl + 'users/' + id + '/message', message);
}

deleteMessage(id: number, userId: number) {
    return this.http.post(this.baseUrl + 'users/' + userId + '/message/' + id, {});
}
```

```
  markAsRead(userId: number, messageId: number) {
  this.http.post(this.baseUrl + 'users/' + userId + '/message/' + messageId + '/read', {}).subscribe();
  }
}
```