МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

_____Аліна САВЧЕНКО

«___»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **«МАГІСТР»**

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

Тема: «Додаток дослідження сторінок соціальних мереж за технологією Asp Net Core»

Виконавець: студент групи УС-212М Здоровець Максим Сергійович

Керівник: _____к.т.н., доцент Холявкіна Тетяна Володимирівна

Нормоконтролер:_____ Ігор РАЙЧЕВ

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та техгнології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____Аліна САВЧЕНКО
«_____»_____ 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційої роботи студента

Здоровця Максима Сергійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Додаток дослідження сторінок соціальних мереж за технологією Asp Net Core» затверджена наказом ректора № 1774/ст від 28.09. 2022р.

2. Термін виконання роботи: з 26 вересня 2022 року по 27 листопада 2022 року.

3. Вихідні дані до роботи: мова розробки додатку – C#, технології розробки додатку – MVC.

4. Зміст пояснювальної записки: вступ, аналіз вбудованої статистики соціальних мереж, вимоги і проектування системи, представлення системи, висновки.

5. Перелік обов'язкового ілюстративного матеріалу: актуальність теми, функціональні вимоги, нефункціональні вимоги, архітерктура ПЗ, результа

6. Календарний план-графік

№ п/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Проаналізувати літературу та джерела за темою дипломної роботи	26.09.22 – 01.11.22р.	
2.	Розроблення та затвердження плану дипломної роботи	02.11.22 – 04.11.22р.	
3.	Привести консультації з науковим керівником щодо створення першого розділу	05.11.22 – 06.11.22р.	
4.	Розробка розділу 1	01.09.22 – 15.09.22р.	
5.	Розробка розділу 2	16.09.22 – 10.10.22р.	
6.	Розробка розділу 3	11.10.22 – 09.11.22р.	
7.	Висновки та оформлення пояснювальної записки дипломної роботи	09.11.22 – 10.11.22р.	
8.	Підписання необхідних документів у встановленому порядку	16.11.22 –18.11.22р.	
9.	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломної роботи	19.11.22 – 22.11.22р.	

7. Дата видачі завдання 26 вересня 2022р.

Керівник дипломної роботи _____ Тетяна ХОЛЯВКІНА
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Здоровець Максим
(підпис випускника) (П.І.Б.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Додаток дослідження сторінок соціальних мереж за технологією Asp Net Core» викладена на 74 сторінках, містить 50 рисунків, 6 літературних джерел.

Ключові слова: СТАТИСТИКА, СОЦІАЛЬНІ МЕРЕЖІ, ВЕБ ДОДАТОК, АНАЛІЗ.

Об’єкт дослідження – веб додаток по збору статистики з різних соціальних мереж.

Предмет дослідження – веб додаток по збору статистики з різних соціальних мереж на основі технології ASP.NET Core MVC.

Мета дипломної роботи – спрощення процесу збору статистики для полегшення керування та просування облікових записів користувачів соціальних мереж.

Наукова значущість дипломної роботи полягає у застосуванні методів проектування та розробки програмного забезпечення, а також програмних методів до розробки нової системи керування та просування облікових записів користувачів соціальних мереж.

Практична цінність. Розроблений програмний продукт дозволяє автоматизувати та уніфікувати процес збору статистики для полегшення керування та просування облікових записів користувачів соціальних мереж.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

MVC – шаблон коду: модель, представлення, контролер.

UI – інтерфейс користувача.

ER – коефіцієнт залученості.

ПЗ – програмне забезпечення.

СМ – соціальні мережа.

ВСТУП

У соціальних мереж є можливість збирати та зберігати велику кількість інформації про своїх користувачів. Соціальні мережі – є інструментом для спілкування з людьми, також для отримання інформації та соціалізації індивідів у всіх сенсах цього слова. Приблизно у 90% вільного часу ми використовуємо соцмережі [1]. Багато часу ми використовуємо на розваги, відео, ігри, розважальні пабліки та групи. На сторінках ми можемо транслювати свій настрій, думки та погляди на події і життя. Все це впливає на нашу аудиторію, яка підписана на наші сторінки і слідкує за всіма нашими змінами. А які вони з цього роблять висновки, вже не є нашим клопітом, бо вони є індивідуами, які мають право транслювати інформацію таким чином, яким забажають.

Соціальні мереж мають певний алгоритм роботи, який надає перевагу особистим сторінкам у показі новин, та може їх транслювати по декілька разів по певним правилам, задля розширення кількості аудиторії, що їх продивляться. Всю інформацію, що ми транслюємо чи споживаємо в тому чи іншому вигляді, може побачити бачать не тільки пошукові системи соцмереж, а й роботодавці і рекрутери, що можуть скласти певне враження про вас як про робітника. Будь-який контент, що ми виставлямо у соцмережах автоматично є доступним усім користувачам цієї мережі, якщо ми не налаштуємо свій акаунт, як приватний. У мережах є стандартні налаштування, що надають дозвіл на індексацію пошуковими системами.

Також в СМ є закрита інформація, якою може володіти виключно та мережа якою ви користуєтесь. Аналізуючи діяльність користувача, СМ створює аватар споживача для рекламодавців. Саме тому, для багатьох людей є дуже важливим, як вони ведуть свої соціальні мережі. Це такі люди, як: блогери, відомі особистості, бізнесмени і т.д., загалом люди, в яких їх сторінки є їх відображенням і способом заробітку. А для того, щоб якісно і максимально ефективно просувати себе, як бренд на рівні соціальних мереж,

треба мати дуже багато інформації і статистики, яку проблематично отримати напряму. Зазвичай ці дані надаються в недостатньому обсязі і користувачу стає необхідним використовувати багато всіляких сторонніх ресурсів. Тому й виникла потреба створити ресурс, на якому в нас буде змога отримувати, досліджувати, аналізувати й правильно використовувати всю необхідну нам інформацію.

Розроблений ресурс буде містити статистику й модуль аналізу з порадами для поетапного просування акаунту користувача. З його допомогою, користувач позбавиться необхідності збирати інформацію по різним соціальним мережам, а буде мати її в одному місці. Таким чином розроблений ресурс стане мостом між користувачем і соціальною мережею, що значно полегшить процес розвитку акаунту.

РОЗДІЛ 1. АНАЛІЗ ВБУДОВАНОЇ СТАТИСТИКИ СОЦІАЛЬНИХ МЕРЕЖ

1.1. Дослідження предметної області.

В нашому світі відбуваються кардинальні зміни, як в способах поширення так і в використанні інформації, що зумовлюють еволюцію в багатьох сферах діяльності і сприяють активній інтеграції бізнесу в інтернет-простір.

За останні роки сервіси соціальних серйозно трансформувалися. Розвинулися їх бізнес-функції, бази даних користувачів, алгоритми та технологічні можливості. Статистики дані свідчать про те, що 90% користувачів інтернет-ресурсів слідкують хоча б за однією бізнес компанією в соціальних мережах, тому розробка маркетингових стратегій в даних мережах — це крок вперед. Стає відомо, що впровадження тих чи інших стратегічних планів є неможливим без контролю за показниками розвитку облікового запису. І тому, статистика облікового запису соціальної мережі — це те, на що слід звернути особливу увагу. Одними з найпопулярніших соціальних мереж є Instagram і Telegram.

1.1.1. Вбудована статистика Instagram

Для того, щоб була можливість оцінювати успіх впровадженого маркетингу в СМ, а саме в Instagram та оптимізовувати контент для більшого ефекту його поширення та вартості, в першу чергу використовується вбудована статистика Instagram акаунту. Процес збору даних та проведення його подальшого аналізу, дозволяють нам відслідковувати прогрес, а також виявляти тенденції та більш докладно оцінювати необхідну нам цільову аудиторію [2].

Ще однією опцією є отримання інформації, про найчастіше використовувані хештеги, та про те, у яких фотографій є найбільша взаємодія від аудиторії і коли саме їх краще публікувати.

Сенс в тому, що показники, які нам стають доступними, можуть дати нам уявлення про те, яка наша діяльність є успішною, а що потрібно змінити, щоб воно стало успішним. Також, нам надається можливість порівнювати свої результати з конкурентськими.

Якщо використовувати статистичні дані, що надаються, то є певна гарантія, що власник того, чи іншого бізнесу не має ризиків, при змінах в своїй стратегії, бо прийняті рішення є обґрунтованими на реальних показниках зі статистик.

Для можливості перегляду статистичних даних, необхідно заводити професійний акаунт. Для того, щоб отримати професійний акаунт необхідно перейти в налаштування застосунку СМ Instagram і перейти до налаштувань професійної панелі (рис. 1.1), яка має наступний вигляд:

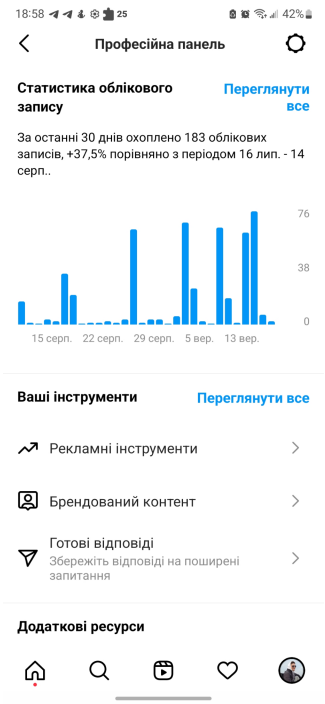


Рис.1.1. Професійна панель

З цієї панелі є можливість перейти до пакету інших необхідних інструментів, що надає соціальна мережа, а саме: робота над брендуванням контенту, налаштування онлайн магазинів, створення відеоконтенту.

Але є один нюанс, якщо у вас виникне бажання просунувати ваш персональний профіль і разом з тим відстежувати свої проміжні результати, то зможете побачити лише показники залученості – тобто кількість написаних коментарів та позначок вподобань. Інші дані не є доступними і звісно ж це є недоліком.

1.1.1.1. Статистика профілю

«Статистика профілю» дає нам доступ до кількох важливих розділів, де можна можна побачити таку інформацію:

- Аудиторія. Тут показується кількість аудиторії, яка охоплює ваші публікації. Якщо точніше, то в цій статистиці можна побачити кількість унікальних юзерів, які споживали ваш контент, а саме: posts, stories, video, lives та реклама. Статистика формується на користувачах, які продивились ваш уонтент хоча б один раз;
- Акаунти. В цій статистиці показано сторінки з яких відбулася взаємодія з вашою персональною сторінкою. До взаємодії вхлдять такі види активів: likes, saves, comments, publics та requests;
- Контент, що був опублікований. У цій статистиці вказуються розширені дані про викладений вами контент, а точніше про його кількість і вид. Якщо точнішеґ. То там показується загальна кількість posts, stories, video, lives та реклами.

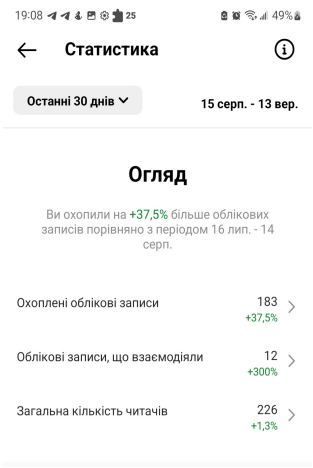


Рис. 1.2. Статистика профілю

1.1.1.2. Взаємодія з контентом

Статистика про залучені облікові записи — це ні що інше, як кількість юзерів, що проводили ту чи іншу взаємодію з вашим контентом, а не просто прокручувала стрічки. У цій вкладці є доступним велика кількість різних метрик.

У даному розділі, де йдеться про залучену аудиторію, показується скільки саме унікальних юзерів зробило лайк, комент, збереження, репост або реакцію на опублікований контент. Ця аудиторія називається — зацікавлена. В цьому розділі ця аудиторія буде відсортованою за такими показниками, як: вік, країна та місто.

У даному блоці, де показно взаємодію з контентом, нам стає доступним відображення загальної кількісті таких дій, як: лайк, коментар, збереження, репост та реакція. Потім число, яке виводиться в статистиці, розбивається з урахуванням кількості постів. Стає можливо побачити відображення своїх найефективніших повідомлень в кожній категорії.

Рис. 1.3. Взаємодія з контентом

1.1.1.3. Статистика опублікованого контенту

Цей розділ надає нам інформація про статистику таких дій:

- лайк;
- коментар;
- репост;
- збереження контенту.

Якщо конкретніше, то в цій статистиці буде доступною інформація про кількість людей, яка переглянуло опублікований контент, також інформація про джерела з яких був здійснений перехід, а саме: профіль, головна сторінка, «Цікаве», хештег.

У розділі, що має назву «Охоплення» буде предоставлено інформацію про кількість унікальних юзерів, що були ознайомлені з контентом, що ви виклали. Ця статистика розділяється на такі показники: статус, статті, вікова категорія, країни та міста. Також там показано охоплену вашою рекламою кількість людей, що дає можливість зрозуміти, наскільки ваша реклама є цікавою і ефективною для сегменту людей, на яку ви її робили.

Статистика під назвою «Покази». Під назвою покази закладено статистику про кількість здійснених переглядів контенту, включаючи повторні ознайомлення одними і тими самими юзерами.

В цілому, ця інформація, нам показує статистичні дані про локацію показів, а саме (рис. 4):

- з вкладки “Головна”;
- з профілю;
- з інших джерел;
- з розташування.

Рис. 1.4. Статистика допису

1.1.1.4. Історії

Статистика в наведеному розділі є аналогічною з тією, що була показана у пункті 1.1.1.3.

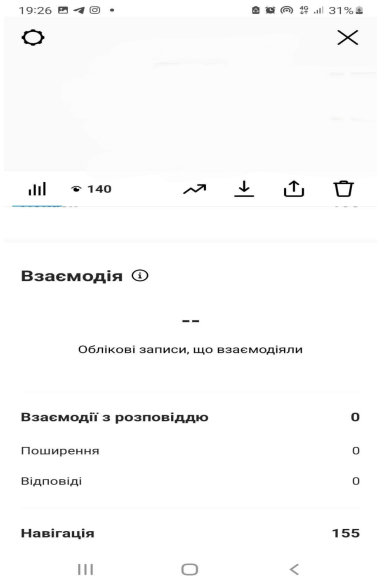


Рис. 1.5. Взаємодія з контентом історій

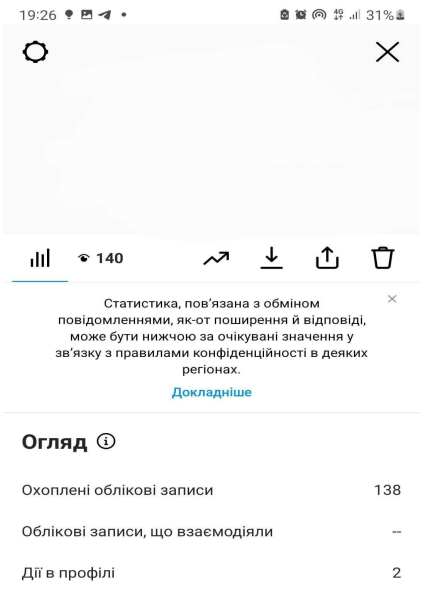


Рис. 1.6. Огляд історії

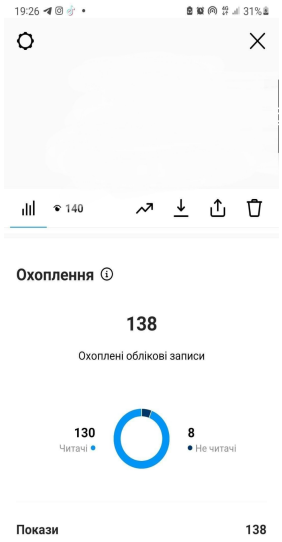


Рис. 1.7. Графік охоплення аудиторії

1.1.1.5. Переваги й недоліки

Провівши дослідження вбудованої статистики СМ Instagram можна вказати на такі переваги й недоліки:

Переваги:

- є можливість для створення своєї стратегії, що може призвести до досягнення успіху в просуванні бренду. В статистиці є корисні дані, що можуть допомогти оптимізувати рекламну компанію і в результаті збільшити обсяги конверсії;
- є можливість краще аналізувати свою цільову аудиторію, використовуючи такі дані, як: вік, місцезнаходження, мову і т.д; Ці дані стають доступні, коли користувач стає власником облікового запису;
- є можливість підвищити ефективність розподілення ресурсів та часу, використовуючи статистику, що показує такі показники, як: контент, що є цінним для юзерів, відзначений контент і вид контенту, що вимагає оновлення. Стає легше розподілити вид і кількість контенту, що є привабливим для користувачів;

- є можливість зрозуміти, по яким закономірностям працює СМ і як їх використати для розвитку бізнесу. Статистику у Instagram можна використовувати для порівняння показників сторінки у різні періоди її існування. Це дозволяє побачити і усвідомити, які зміни були під час піків.

Недоліки:

- вбудована статистика СМ є дуже вузькою і є доступною тільки з смартфонів;
- для просування персонального профілю та відстеження своїх результатів надається дуже мало інформації. Ви зможете ознайомитись лише з такою інформацією, як: кількість коментарів і кількість вподобань. Інші статистичні дані не будуть доступними;
- немає можливості «кастомізувати» візуалізацію даних у необхідному для вас форматі;
- функціоналу по експорту статистики являється відсутнім.

1.1.2. Вбудована статистика Telegram

Статистика надає нам дозвіл на отримання інформації про такі показники, як: охоплення створених вами публікацій, час активності аудиторії, джерела з яких здійснювався перегляд та інше.

Статистичні інформація є доступною лише для адміністраторів каналів, які мають обсяг аудиторії більше 1000 користувачів. Для того, щоб у адміністратора була можливість перегляду інформації, він повинен мати права на публікацію постів [3].

Вбудована статистика СМ в повному обсязі містить десять графічних відображень інформації:

1. **Зростання**

Графік під назвою "Зростання" надає інформацію про зміни кількісті підписаних на ваш канал користувачів. Ця інформація дає можливість для

оцінки динаміки розвитку вашого каналу, а саме: присутність або відсутність різкого приросту, спад, які періоди є піковими, тощо.

2. **Підписники**

Графік під назвою"Підписники" надає інформацію про зміни кількісті користувачів, що підписалися та відписалися від публічного каналу. Максимальним періодом для збору цієї статистики є близько півроку. Отримати деталізацію про такі дані, як: кількість нової підписаної аудиторії та кількість втраченої підписаної аудиторії, можна аж до кожного дня. В цій статистиці можна обирати відображення графіків, тобто показується або обидва графіки або тільки один.

3. **Повідомлення**

Графік під назвою "Повідомлення" надає інформацію про зміни кількісті користувачів, що увімкнули або вимкнули сповіщення про повідомлення на каналі. Це може допомогти відстежувати закономірності — який контент стає причиною для людей відключати сповіщення про повідомлення, а який навпаки мотивує не пропускати нових публікацій. Також надається інформація про періодичність «постінгу», яка стимулює підписників «замутити» або «розмутити» канал тощо.

4. **Перегляди по годинниках**

Графік під назвою "Перегляд по годинниках" надає інформацію, за допомогою якої можна провести аналіз по відстеженню часу, в який ваш контент переглядається активніше або пасивніше. Є можливість побачити відображення середнього показнику перегляду контенту за термін - протягом тижня. На графіку зображено дві лінії з яких одна відображає інформацію за поточним тижнем, а інша — за попереднім.

5. **Джерела переглядів**

Графік під назвою «Джерела переглядів» надає інформацію, за допомогою якої можна відстежити, де користувачі СМ Telegram бачать ваші публікації.

Для відображається надається шість джерел:

- Підписники – це юзери, які підписані саме на ваш канал.
- Канали — це юзери, які переглянули ваш контент на іншому каналі.
- URL — це перегляди, які були здійснені за допомогою прямого посилання на пост.
- Групи — це перегляди, що отримала ваша публікація після репоста до певної групи.
- Особисті чати — це репости, що були здійснені в певні чати.
- Пошук — це перегляди, що були зібрані методом пошуку в вашому каналі.

Цей графік дає вам можливість відстежувати результати просування контенту, а також зафіксувати можливе «вірусування» посту і початок його поширення по чатах та групах.

6. **Джерела підписників**

Графік під назвою «Джерела підписників» є подібним з попереднім, але його задача фіксувати саме тих користувачів, що підписалися на канал. Джерелами підписання користувачі тут вказуються: групи, канали, особисті чати, пошук та посилання.

7. **Мови**

Графік під назвою "Мови" надає інформацію, за допомогою якої ви можете дізнатися, якою мовою користується ваша аудиторія в інтерфейсі Telegram. Дані відображаються у вигляді діаграм. Тут ви можете налаштовувати відключення деяких мов, а також проводити деталізацію часу в діапазоні від дня до тижня.

8. **Активність**

Графік під назвою "Активність" надає інформацію, за допомогою якої ви можете дізнатися, динаміку активності юзерів, які користуються вашим каналом. Для зручності було введено два показники: репости та перегляди.

9. **Активність у Instant View (IV)**

Графік під назвою "Активність у Instant View" надає інформацію, за допомогою якої ви можете дізнатися про статті миттєвого перегляду, наприклад матеріали, що були зверстані через telegra.ph. Тут дані мають відображення у вигляді переглядів та репостів прямо зі сторінок IV. Якщо ви не користуєтесь публікуцією таких матеріалів, то у статистиці буде відображатись лише порожнє поле.

10. **Нещодавні посади**

Графік під назвою "Нещодавні посади" надає інформацію, за допомогою якої ви можете дізнатися про репости та перегляди свіжо викланих публікацій. Максимальний період, що надається для відстеження цієї інформації – тиждень. Щоб дослідити статистику за конкретним постом, вам необхідно натиснути на нього і тоді відкриється детальна статистика саме по ньому.

1.1.2.1. Переваги й недоліки

Провівши дослідження вбудованої статистики СМ Telegram можна вказати на такі переваги й недоліки:

Переваги:

- є можливим проаналізувати і зробити висновки по розвитку вашого каналу за допомогою даних що надається самою СМ. Йде мова про такі статистичні дані, як: джерела переглядів, аудиторія каналу, час коли вони проявляють активність;
- є можливість краще аналізувати свою цільову аудиторію, використовуючи такі дані, як: вік, місцезнаходження, мову і т.д; Ці дані стають доступні, коли адмін має певну кількість підписників;
- є можливість підвищити ефективність розподілення ресурсів та часу, використовуючи статистику, що показує такі показники, як: контент, що є цінним для юзерів, відзначений контент і вид контенту, що вимагає

оновлення. Стає легше розподілити вид і кількість контенту, що є привабливим для користувачів;

- є можливість зрозуміти, по яким закономірностям працює СМ і як їх використати для розвитку бізнесу. Статистику у Instagram можна використовувати для порівняння показників сторінки у різні періоди її існування. Це дозволяє побачити і усвідомити, які зміни були під час піків.

Недоліки:

- статистика призначена тільки для просунутих великих каналів. Вам надається дозвіл на отримання статистики тільки в випадку набирання вами певної кількості аудиторія, а саме — 1000 підписників. Інакше про поведінку вашої аудиторії ви зможете дізнатись тільки через зовнішні сторонні сервіси;
- немає можливості «кастомізувати» візуалізацію даних у необхідному для вас форматі;
- функціоналу по експорту статистики являється відсутнім.
- вбудована статистична інформація не має врахування кількості кліків. Активність аудиторії може відображатися лише за показниками репостів і передплат.

1.2. Аналіз UI/UX

Важливим етапом в розробці будь-якого застосунку є розробка правильного UI/UX, тобто задача в тому, що є необхідним чітко і конкретно підійти до розробки привабливого і в той же час інтуїтивно зрозумілого інтерфейсу, який буде виділяти ту чи іншу систему з поміж інших.

UX має ціль, яка направлена на те, щоб допомогати користувачу у досягнені бажаних результатів і мати здатність залишати у нього позитивні враження від роботи. Якщо розбирати його завдання, то вони є такими:

- мати змогу дати користувачу можливість відчути, що продукт є унікальним;

- дати користувачу бажання до вивчення продукту і користування його контентом;
- дати користувачу чіткі переконння в тому що продукт вартий того, щоб його спробувати;
- залишити людину користувачем цього продукту.

Як показують проведені раніше дослідження, якщо інвестувати в дизайн, то є можливість значно збільшити можливий прибуток. Якщо прислуховуватись до оцінок незалежної аналітичної компанії в сфері інформаційних технологій Forrester Research [4], то можна зрозуміти, що один вкладений долар в дизайн, може приносити 2 - 100 доларів прибутку.

Отже можна константувати, що UX дійсно зпроможний допомогти бізнесу в таких речах, як:

- збільшення конверсії;
- підвищення продаж;
- підвищення лояльність користувачів.

Також можна зазначити, що ретельною роботою з UX є змога покращити зрозумілість продукту для користувача, а також заощаджувати на його обслуговуванні, прийшовши до зниження:

- витрат на підтримку;
- невдоволення користувачів;
- витрат на залучення користувачів.

У деякихвипадках, ваш продукт може виявитись незручним для використання користувачами. Причинамо тому може стати розробка без урахування таких аспектів, як: конкретні бізнес-процеси компанії, потреби користувачів. Порадою є оминання розробки незручного ІТ-продукту, що знешкодить можливі ризики в конкурентній боротьбі.

В таких випадках виходом є проведення аудитів — на яких слід проаналізувати, як ІТ-продукт, яким ви володієте може виконувати або вже

виконує поставлені завдання і наскільки UX і UI є відповідним до вимог ринку. Таким чином, за допомогою проведення аудиту по UX, ви можете виявити основні проблеми, які є в продукті, а це є першим кроком на шляху до їх усунення.

Висновок до розділу

Провівши ретельний аналіз вбудованих статистик переглянутих СМ мають проблеми. Виявлені проблеми здатні вплинути на user experience, а також не мають достатнього функціоналу, що в результаті може зашкодити досягненню поставлених йому цілей по просуванню бізнесу.

РОЗДІЛ 2. ВИМОГИ І ПРОЕКТУВАННЯ СИСТЕМИ

Для того, щоб визначити вимоги до системи, необхідно провести їх аналіз. Під час аналізу вимог, ми визначаємо потреби та вимоги які постають перед системою.

Аналіз вимог має 3 єтапи:

- виявлення вимог;
- аналіз вимог;
- запис вимог.

Під час виявлення вимог, ми займаємося збором вимог. Під час аналізу виявляємо недоліки вимог і способи їх виправлення. Потім вимоги документуються. Аналіз вимог загалом є довгим та важким процесом, що вимагає використання часу. Нові системи змінюють середовище і відношення між людьми, тому важливо розпізнати всі зацікавлені сторони, взяти до уваги всі їхні потреби, і переконатись що вони розуміють наслідки які приносить нова система. Аналітики можуть використати кілька методів щоб отримати від споживача вимоги. Історично це включає проведення інтерв'ю, чи фокус-груп (яку в цьому контексті частіше називають як майстерня вимог) і створення списків вимог. До сучасніших підходів відносять прототипування, та прецеденти. За потреби аналітик використає комбінацію цих методів щоб встановити точні вимоги зацікавлених сторін, так щоб система відповідала бізнес-потребам [5].

У майбутньому цей додаток буде розширюватись у напрямку оптимізації та розширення програмних функцій.

Розроблюваний додаток буде унікальним. Єдиною загрозою для такого програмного продукту може стати фінансування.

У даному розділі будуть описаними такі вимоги:

- вимоги користувача;
- функціональні вимоги;
- нефункціональні вимоги;
- системні вимоги;
- вимоги до апаратного забезпечення.

2.1. Вимоги користувача

- Реєстрація користувача
- Авторизація користувача
- Перегляд основного функціоналу сайту
- Скористатись функціоналом сайту
- Зареєстрований користувач матиме можливість:
    - обрати СМ, яка цікавить юзера;
    - заповнювати інформацію про свої акаунти різних СМ;
    - виконувати дослідження акаунта СМ;
    - переглянути сторінку з графіками по статистиці акаунту СМ;
    - переглянути можливі поради для поетапного розвитку акаунту СМ;
    - переглядати історію пошукових запитів;
    - повторювати запити, користуючись історією;

Діаграма використання софту сображена на рисунку 2.1.

Більш детальним описом варіантів використання системи можна побачити за допомогою схеми, що має назву - вербальні специфікації прецедентів. За допомогою вербальних специфікацій прецендентів можна передбачити опис певних сценаріїв використання ПЗ у словесній формі. У наведених специфікаціях можливим є опис альтернативних варіантів

розвитку подій і умови, які мають виконуватись, щоб спровокувати їх появу.

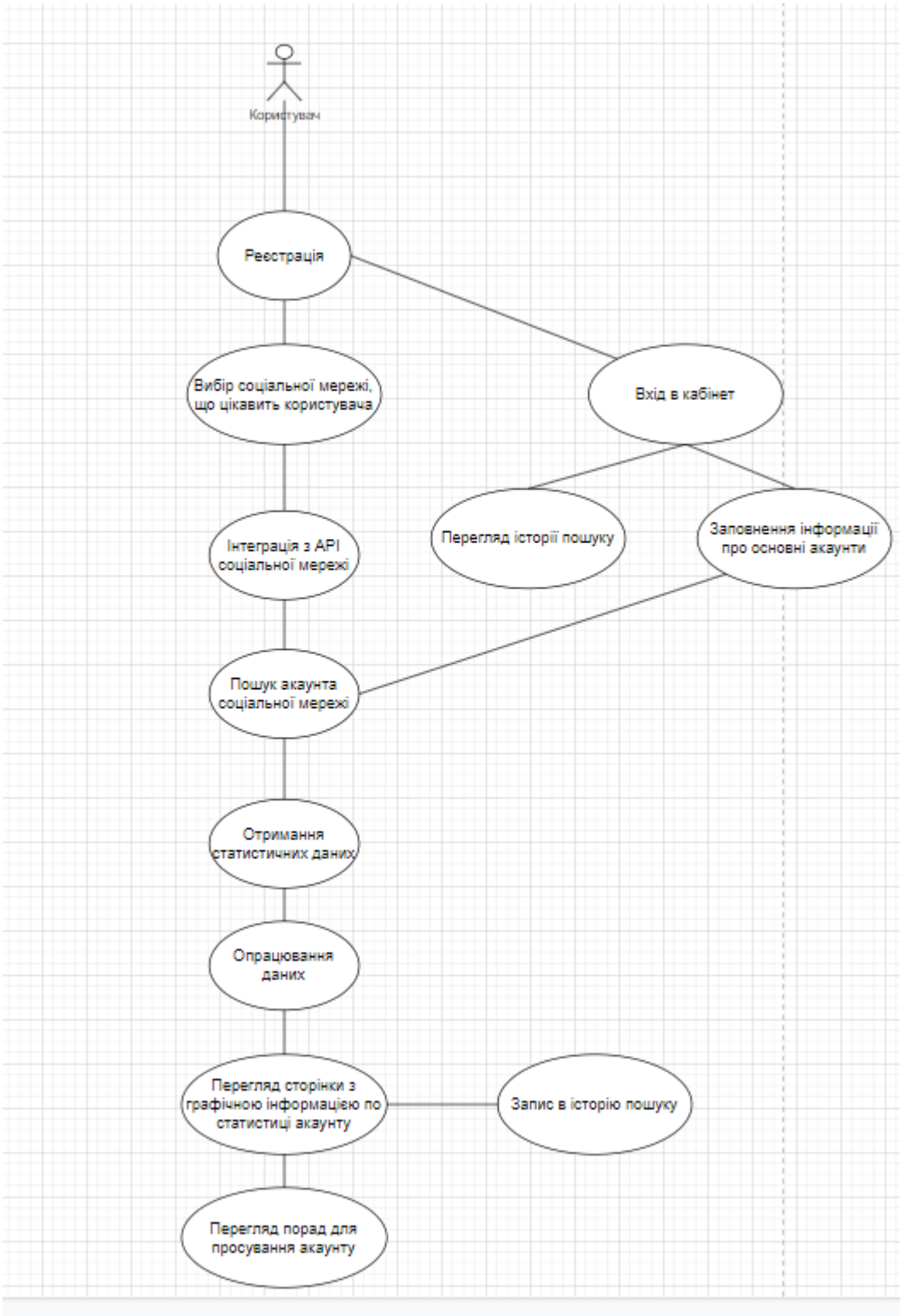


Рис. 2.1. Діаграма використання

2.1.1. Вербальна специфікація

1. Дослідження акаунту СМ.

Короткий опис: Дослідження акаунту СМ, для отримання графіків зі статистичною інформацією.

Суб'єкт - Користувач.

Передумови:

1) Користувач обирає СМ, яка його цікавить і натискає на кнопку «Пошук».

Основний потік:

1) Користувач проводить реєстрацію на сайті.
2) Користувач натискає на кнопку «Пошук».
3) Користувач обирає СМ, яка його цікавить.
4) Користувач заповнює інформацію, що є необхідною для пошуку та отримує результат дослідження СМ.

Альтернативні потоки:

А1. Вказаний акаунт не знайдено. Якщо при здійсненні пошуку, акаунт був не знайденим, за тими даними, що вводив користувач, то юзеру буде повертатись модальне вікно з повідомленням, що вказаний вами акаунт не знайдено.

Постумови:

Користувач отримує доступ до інформації про статистику акаунту СМ у вигляді графіків.

2. Перегляд історії пошуку статистики акаунта.

Короткий опис: Перегляд історії попередніх пошукових запитів по дослідженню статистики акаунтів СМ.

Суб'єкт - Користувач.

Передумови:

1) Користувач перейшов на сторінку історії пошукових запитів, натиснувши кнопку «Історія пошуку» і отримав доступ до історії пошукових запитів по дослідженню акаунта СМ.

Основний потік:

1) Користувач провів реєстрацію або авторизацію у кабінет.

2) Користувач перейшов на сторінку дослідження акаунтів СМ і зробив дослідження.

3) Користувач перейшов до історії пошуку в акаунті та отримав історію попередніх пошукових запитів.

Альтернативні потоки:

А1. Історія вашиз пошукових запитів пуста. Якщо історія пошукових запитів є пустою, то юзеру повертається модальне вікно з повідомленням про те, що історії пошукових запитів на даний момент пуста.

Постумови:

Користувач отримує доступ до інформації про попередні пошукові запити статистики акаунту СМу вигляді графіків.

4. Заповнення інформації про основні аккаунти СМ.

Короткий опис: Заповнення необхідних даних про основні акаунти СМ юзера, для подальшого дослідження.

Суб'єкт - Користувач.

Передумови:

1) Користувач провів авторизацію у кабінет.

Основний потік:

1) Користувач обрав СМ, яка його цікавить.

2) Користувач заповнив необхідні дані, що потребує форма для дослідження та отримує результат дослідження у вигляді графіка.

3) Після вдалого дослідження акаунту СМ, результати записуються в історію пошукових запитів на дослідження певної СМ.

Альтернативні потоки:

А1. Акаунт за вашим запитом не знайдено. Якщо пошукова операція не знайшла акаунт, за даними, що заповнив користувач, то юзеру буде повертатись модальне вікно з повідомленням, що вказаний вами акаунт не знайдено.

Постумови:

Користувач отримує доступ до інформації про статистику акаунту СМ у вигляді графіків.

2.2. Функціональні вимоги

1. В системі має бути функція реєстрація/авторизація юзерів.
2. В системі має бути імплементація інтеграції з API СМ, за допомогою яких буде реалізовано побудова статистичних графіків.
3. В системі має бути особистий кабінет юзера. На сторінці особистого кабінету має бути імплементовано:

- Можливість перейти на головну сторінку.
- Можливість перейти скористатися сторінкою “Історія пошуку”.
- Можливість відкрити сторінку “Особисті дані”.
- Можливість перейти на сторінку дослідження акаунтів СМ.

В системі має бути імплементовано механізм запису історії пошукових запитів на дослідження статистики акаунту СМ.

2.3. Нефункціональні вимоги

1. **Вимоги до інтерфейсу** (Interface Requirements)
    1. Зручність
    2. Зрозумілість
    3. Легкість у використанні
    4. Мінімалізм, що приємно для перегляду

5. Стійкість
6. Адаптивність
7. Швидкодія

2. **Апаратні та програмні вимоги** (Hardware/Software Requirements) —
   1. Кроссплатформенність – завдяки технології .Net Core
   2. Адаптивність – завдяки Bootstrap і Media Query
3. **Бізнес вимоги**
   1. Приваблення уваги користувачів до сайту;
   2. Зручність і простота дизайну сайту;
   3. Інформацію подавати простою і доступною мовою;
   4. Зручний інтерфейс.

2.4. Системні вимоги

2.4.1. Вимоги до надійності системи

Стійке функціонування ПЗ має бути забезпечене шляхом певних речей:

1) контроль коректності та повноти даних – тобто всі дані, що вводить користувач, мають перевіряютись на формальну коректність;

2) логування дій, що робитькористувачі;

3) синхронізація системи в разі поламки з останнім зафіксованим стабільним станом;

4) можливості створень резервних копій інформаційної БД.

2.4.2. Вимоги до захисту інформації від несанкціонованого доступу

Захист інформації від всякого виду несанкціонованого доступу до неї повинен забезпечуватися такими засобами ПЗ, як авторизація юзерів, контроль та протоколювання дій авторизованих користувачів.

Дані, необхідні для функціонування ПЗ та особисті дані юзерів, повинні зберігатися на сервері ПЗ у зашифрованому вигляді.

2.5. Вимоги до апаратного забезпечення

ПЗ розрахований на роботу при такому наборі технічних засобів:

Апаратна конфігурація сервера:

- 4-ядерний процесор;
- оперативна пам’ять має бути від 10 Гб;
- вільний дисковий простір має бути від 8 Гб.

Вимоги до клієнта:

- Наявність браузера

2.6. Опис технологій для реалізації системи

Back end:

• **C#** - на цей момент мова програмування C # одна з найпотужніших.Вона швидко розвивається і є однією з самих затребуваних мов в IT-галузі. На даний момент на ньому пишуться найрізноманітніші програми: від невеликих десктопних програмок до великих веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів, чим ми і скористуємося.

• **SQL** - SQL Server – це одна з найбільш популярних систем управління базами даних (СКБД) в світі і це досить виправдано. Дана СУБД підходить для самих різних проектів: від невеликих додатків до великих високонавантажених проектів, що стане нам в нагоді.

Front end:

• **JavaScript + jQuery** – JS мультіпарадігменна мова програмування для логіки на стороні клієнта. Підтримує об'єктно-орієнтований, імперативний і функціональний стилі. Є реалізацією стандарту ECMAScript.

JavaScript зазвичай використовується як вбудований мова для програмного доступу до об'єктів додатків і керуванням функцій для логіки, анімацій чи маніпулювання стилями розмітки. jQuery - набір функцій JavaScript, що фокусується на взаємодії JavaScript і HTML, що сильно скорочує і полегшує прописування коду. Бібліотека jQuery допомагає легко отримувати доступ до будь-якого елементу DOM, звертатися до атрибутів і вмісту елементів DOM, маніпулювати ними і змінювати їх. Також бібліотека jQuery надає зручний API для роботи з AJAX, що дає нам можливість спілкуватися і взаємодіяти с серверною частиною програми [6].

Верстка:

• **HTML** - стандартизована мова розмітки документів у Всесвітній павутині і є саме тою частиною, яку бачить користувач. Більшість веб-сторінок містять опис розмітки на мові HTML, що не дивно, бо вона є стандартом [9].

• **CSS** - формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки, а саме виступає можливістю стилізувати документ у всесвітній павутині.

Адаптивна верстка:

• **Bootstrap 5** - це безкоштовна колекція стилізованих і адаптивних об’єктів з відкритим вихідним кодом коду CSS і JavaScript / jQuery, використовувана для створення сайтів з динамічним макетом і веб-додатків.

Середовища разробки:

• *MS Visual Studio/Code*

• *MSSQL*

Технології:

• **.NET Core** - це модульна крос-платформа для розробки програмного забезпечення з відкритим вихідним кодом і простим використанням. Сумісна з такими операційними системами як Windows, MacOS, Linux, Android, iOS, Tizen

• **WebApi** - представляє спосіб і можливість побудови програми ASP.NET, який спеціально заточений для роботи в стилі REST (Representation State Transfer або "передача стану вистави"),

• **ASP.NET Core MVC** - Фреймворк ASP.NET Core MVC працює поверх платформи ASP.NET Core, і призначений для того, щоб спростити створення програми за видом Model-View-Controller. Але ми можемо і не використовувати MVC, а застосовувати чистий ASP.NET Core і на ньому цілком вибудовувати логіку програми, але не будемо [7].

• **Entity Framework Core** - являє собою об'єктно-орієнтовану, легковажну і расширювану технологію від компанії Microsoft для доступу до даних, що знаходяться в базі. EF Core є зручним ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти)[8].

2.7. Опис основних компонентів

ПЗ буде мати в собі 8 компонентів:

- Сервіс реєстрації та авторизації/аутентифікації
- Сервіс будування статистичних графіків.
- Сервіс інтеграцій з API СМ.
- Сервіс для роботи з історією пошукових запитів на дослідження.
- Сервіс для роботи з кабінетом юзера.
- Сервіс для роботи з домлідженнями акаунтів СМ.
- БД.
- Інтерфейс для користувачів.

1. Сервіс реєстраціЇ та авторизаціЇ/аутентифікації

Цей сервіс створюється для роботи з функціонал по реєстрації та авторизації/аутентифікації юзера. Механізм буде реалізований за допомогою вбудованої в ASP.NET Core систему ASP.NET Identity.

2. Сервіс будування статистичних графіків

Цей сервіс відповідає за роботу з статистичними даними і побудовою по ним графіків.

Основні функції:

- Робота з сервісом по інтеграції з API СМ
- Робота з отриманими даними
- Побудова графіків

3. Сервіс інтеграцій з API СМ

Цей сервіс несе відповідальність за інтеграцію з API СМ (отримання даних про акаунт необхідної СМ).

Основні функції:

- «Спілкування» з API СМ
- Отримання необхідних даних

4. Сервіс для роботи з історією пошукових запитів на дослідження

Цей сервіс несе відповідальність за збереження пошукових запитів по дослідженню СМ.

Основні функції:

- Збереження пошукових запитів по дослідженню СМ користувача
- Відображення історії пошукових запитів по дослідженню СМ на сторінці особистого кабінету

5. Сервіс по роботі з кабінетом юзера

Цей сервіс несе відповідальність за функціонал, що є в кабінеті користувача.

Основні функції:

- Відображення історії пошуку по дослідженню СМ
- Пошук по збереженим налаштуванням з минулих досліджень основних акаунтів СМ користувача

6. Сервіс для роботи з домлідженнями акаунтів СМ

Цей сервіс несе відповідальність за збереження статистичних даних з дослідження СМ у БД

Основні функції:

- Збереження даних

Для детальнішого опису компонентів ПЗ, що були перечислені і описані вижче, розглянемо діаграму послідовності (рис.2.2).

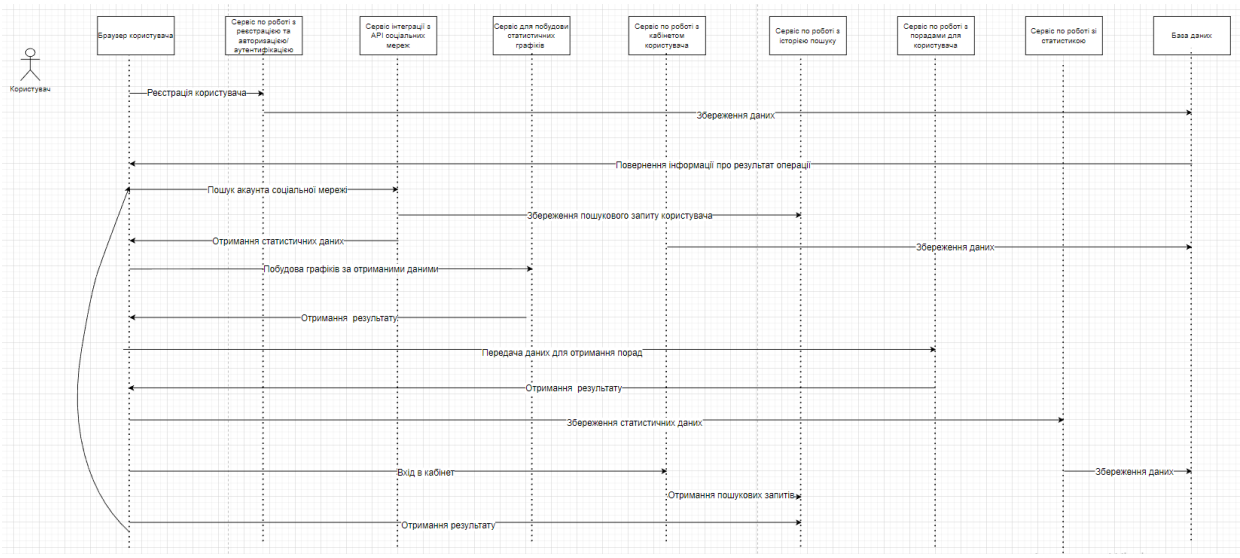


Рис.2.2. Діаграма послідовності

2.8. Архітектура проекту

Система буде розроблена з використанням трирівневої архітектури.

Шар - це багаторазова частина коду, яка виконує певну функцію. У середовищі .NET шар зазвичай встановлюється як проект, що представляє цю специфічну функцію. Цей конкретний шар відповідає за роботу з іншими шарами для досягнення певної мети. Спершу коротко розглянемо останню ситуацію.

Шар даних. DAL містить методи, які допомагають бізнес-рівню зв’язувати дані та виконувати необхідні дії, як для повернення даних, так і для обробки даних.

Бізнес рівень

BAL містить бізнес-логіку, перевірки чи обчислення, що стосуються даних.

Незважаючи на те, що веб-сайт може спілкуватися безпосередньо з рівнем доступу до даних, він, як правило, проходить інший рівень, який називається Бізнес-рівень. Бізнес-рівень є життєво важливим, оскільки він перевіряє умови введення перед викликом методу з рівня даних. Це гарантує правильність введення даних перед тим, як продовжувати, і часто може також забезпечити правильність виводу. Така перевірка вхідних даних називається діловими правилами, тобто правилами, які бізнес-рівень використовує для «судження» щодо даних.

Презентаційний шар

Шар презентації містить такі сторінки, як форми .aspx або Windows Forms, де дані представляються користувачеві або вводиться в користувача. Веб-сайт ASP.NET або програма Windows Forms (інтерфейс користувача для проекту) називається рівень презентації. Презентаційний шар - це найважливіший шар просто тому, що його бачать і використовують усі. Навіть при добре структурованому рівні бізнесу та даних, якщо рівень презентації розроблений погано, це дає користувачам поганий огляд системи.

Переваги трирівневої архітектури

Основною характеристикою архітектури хосту є те, що програма та бази даних розміщені на одному і тому ж хост-комп’ютері, і користувач взаємодіє з хостом, використовуючи недружній тупий термінал. Ця архітектура не підтримує розподілених обчислень (хост-програми не можуть підключитися до бази даних стратегічно пов'язаного партнера). Деякі менеджери вважають, що розробка хост-додатків займає занадто багато часу і це дорого. Ці недоліки, отже, призвели до архітектури клієнт-сервер.

Клієнт-серверна архітектура - це дворівнева архітектура, оскільки клієнт не розрізняє рівень презентації та рівень бізнесу. Зростаючі вимоги до елементів керування графічним інтерфейсом викликали труднощі в управлінні сумішшю вихідного коду з графічного інтерфейсу та бізнес-логіки (код спагетті). Крім того, архітектура клієнтського сервера недостатньо підтримує управління змінами. Припустимо, що уряд збільшує ставку податку на розваги з 4% до 8%, тоді у випадку Клієнт-Сервер нам потрібно надіслати оновлення кожному клієнту, і вони повинні синхронно оновлюватись у визначений час, інакше ми можемо зберігати недійсні або неправильна інформація.

Архітектура клієнт-сервер також є тягарем для мережевого трафіку та ресурсів. Припустимо, що близько п’ятисот клієнтів працюють на сервері даних. Тоді ми матимемо п’ять сотень з’єднань ODBC та кілька наборів записів, які потрібно транспортувати із сервера до клієнтів (оскільки бізнес-рівень залишається на стороні клієнта). Той факт, що клієнт-сервер не має ніяких засобів кешування, як в ASP.NET, спричиняє додатковий трафік у мережі. Зазвичай сервер має краще обладнання, ніж клієнт, тому він здатний обчислювати алгоритми швидше, ніж клієнт, тому цей факт також є додатковим аргументом на користь 3-рівневої архітектури.

Ця категоризація програми полегшує функцію для багаторазового використання, і стає занадто легко знаходити функції, написані раніше. Якщо програміст хоче робити подальші оновлення в програмі, він може легко зрозуміти попередній написаний код і може легко його оновити.

Але для системи на цій стадії більш вигідно буде поєднати всі 3 рівні в один, а при доповненнях і розширенні програми, вона буде розподілятися на три.

2.8.1. Реалізація згідно архітектури

1. Рівень представлення

Цей рівень містить в собі:

- контролери,
- представлення
- моделі для цих представлень (рисунки 2.3-2.8).

Як приклад наведено метод для реєстрацію юзера — Register, який міститься в контролері - AccountController, сторінка для реєстрації користувача, вью модель - RegisterViewModel, та js-файл chart.js, що несе відповідальність за побудову графіків

Сторінка реєстрації - користувач заповнює у форму такі дані, як: свою електроннну пошту і вигаданий пароль для реєстрації на продукті.

```
<div class="limiter">
    <div class="container-login100">
        <div class="wrap-login100 p-t-85 p-b-20">
            <form class="login100-form validate-form">
                <span class="login100-form-avatar">
                    <img src="~/images/images.png" alt="AVATAR">
                </span>

                <div class="wrap-input100 validate-input m-t-85 m-b-35" data-validate="Enter email">
                    <input class="input100" type="text" name="email" id="email">
                    <span class="focus-input100" data-placeholder="Електронна адреса"></span>
                </div>

                <div class="wrap-input100 validate-input m-b-50" data-validate="Enter password">
                    <input class="input100" type="password" name="pass" id="password">
                    <span class="focus-input100" data-placeholder="Пароль"></span>
                </div>

                <div class="container-login100-form-btn">
                    <button class="login100-form-btn" id="submit">
                        Створити обліковий запис
                    </button>
                </div>

                <ul class="login-more p-t-190">
                    <li>
                        <span class="txt1">
                            Уже маєте обліковий запис?
                        </span>

                        <a href="/login" class="txt2">
                            Увійти
                        </a>
                    </li>
                </ul>
            </form>
        </div>
    </div>
```

Рис.2.3. Сторінка реєстрації

Дані, що були заповнені у формі, потім передаються в реалізований для цього метод - Register, який знаходиться у відповідному для користувача контролері - AccountController.

```
<script>
    $("#submit").click(function () {
        var email = $("#email").val();
        var password = $("#password").val();
        $.ajax({
            async: false,
            url: '/register',
            method: 'post',
            data: { Email: email, Password: password }
        }).done(function () {
            window.location.assign('/test');
        });

        return false;
    });
```

Рис.2.4. Функція, яка відправляє дані, які ввів користувач при реєстрації

Дані, що були введені згодом конвертуються в об’єкт класу RegisterViewModel за допомогою внутрішнього функціоналу, що представляє ASP.NET Core.

```
namespace Diplom.ViewModels
{
    ссылка: 1
    public class RegisterViewModel
    {
        [Required]
        Ссылок: 2
        public string Email { get; set; }
        [Required]
        ссылка: 1
        public string Password { get; set; }
    }
}
```

Рис.2.5. Модель представлення для реєстрації

Метод Register при звернені до нього, отримує дані у вигляді об’єкту класу, що був для реалізований - RegisterViewModel і проводить валідацію, згодомістворює об’єкт класу, що був реалізрвний для запису користувачів -

User і за допомогою функціоналу, що представляє ASP.NET Core Identity записує ці дані у БД.

```
[HttpPost("register")]
Ссылок: 0
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (!ModelState.IsValid)
        return BadRequest();

    User user = new User { Email = model.Email, UserName = model.Email };

    var result = await _userManager.CreateAsync(user, model.Password);
    if (result.Succeeded)
    {
        await _signInManager.SignInAsync(user, false);
        return Ok();
    }

    return BadRequest();
}
```

Рис.2.6. Метод контролера, який відповідає за реєстрацію користувача

JS-файл chart.js містить в собі логіку для побудови графіків за статистичними даними.

На рисунку 2.7 показана реалізація логіки побудови графіка юзерів. Спочатку від сервісу інтеграції з API CM ми отримуємо необхідні нам статистичні дані, а саме: набір об’єктів «ключ-значення» - де ключ = дата, а значення = кількість підписників у даний день. Всі інші графіки формуються за схожою логікою.

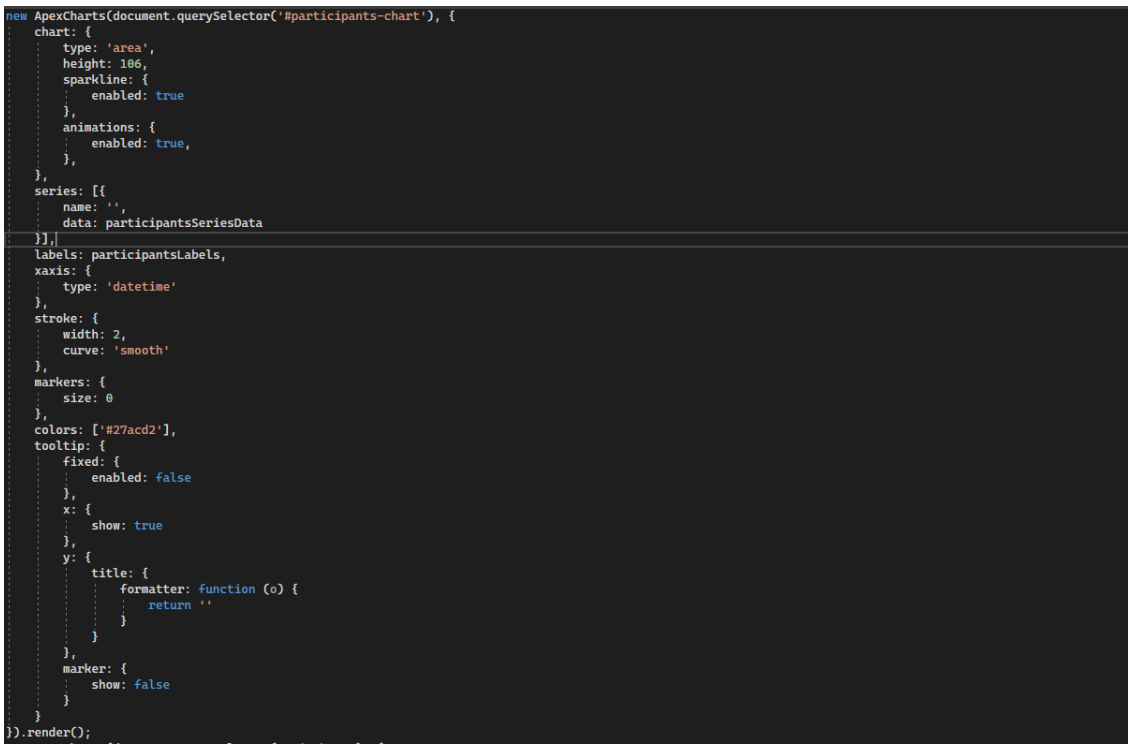


Рис.2.7. Логіка побудови графіка користувачів

На рисунку 2.8. показана реалізація логіки для отримання даних про кількість підписників за певний період (в даному прикладі за місяць).

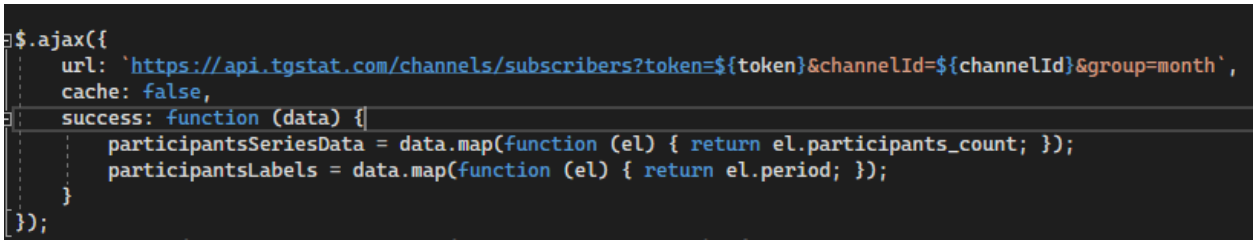


Рис.2.8. Отримання статистичних даних про підписників акаунту

2. Рівень бізнес-логіки

Цей рівень містить в собі сервіси, які несуть відповідальність за обробку даних з рівня представлення. Вяк приклад наведено клас - SearchHistoryService, який несе відповідальність за логіку для функціоналу історії пошуку (рисунки 2.9-2.10).

Даний клас проводить реалізацію інтерфейса - ISearchHistoryService, а в середині себе містить CRUD операції:

- додавання,
- оновлення,
- видалення,
- отримування даних про історію пошуку користувача.

В якості параметрів, методи цього класу отримують - DTO SearchHistoryDto (містить інформацію про пошуковий запит по дослідженню СМ користувача) або ідентифікатор історії пошукових запитів в БД.

```
Ссылок: 1
public class SearchHistoryService : ISearchHistoryService
{
    private readonly IUnitOfWork<SearchHistory, int> _unitOfWork;
    private readonly IMapper _mapper;

    Ссылок: 0
    public SearchHistoryService(SqlUnitOfWork<SearchHistory, int> unitOfWork, IMapper mapper)
    {
        _unitOfWork = unitOfWork;
        _mapper = mapper;
    }

    Ссылок: 1
    public async Task AddAsync(SearchHistoryDto entity)
    {
        var model = _mapper.Map<SearchHistory>(entity);
        _ = _unitOfWork.Repository.Add(model);
        await _unitOfWork.SaveChangesAsync();
    }

    Ссылок: 1
    public async Task DeleteAsync(int id)
    {
        _unitOfWork.Repository.Remove(id);
        await _unitOfWork.SaveChangesAsync();
    }
```

Рис.2.9. Конструктор класу і методи додавання і видалення

```
Ссылок: 1
public async Task<IEnumerable<SearchHistoryDto>> GetAllAsync()
{
    var models = await _unitOfWork.Repository.GetAll().ToListAsync();
    return _mapper.Map<List<SearchHistoryDto>>(models);
}

Ссылок: 1
public async Task<SearchHistoryDto> GetByIdAsync(int id)
{
    var task = await _unitOfWork.Repository.GetByIdAsync(id);
    return _mapper.Map<SearchHistoryDto>(task);
}

Ссылок: 1
public async Task UpdateAsync(SearchHistoryDto entity)
{
    var task = _mapper.Map<SearchHistory>(entity);
    _unitOfWork.Repository.Update(task);
    await _unitOfWork.SaveChangesAsync();
}
```

Рис.2.10. Методи оновлення і отримання даних

Ще в наведеному класі використовується mapper для того, щоб зробити проекції моделі БД на модель (DTO) історії пошукових запитів. Для mapper в проекті використовується NuGet-пакет AutoMapper. Прикладом роботи маперу наведений рисунку 2.11.

```
Ссылок: 2
public class EntitiesAutoMapperProfile : Profile
{
    Ссылок: 0
    public EntitiesAutoMapperProfile()
    {
        CreateMap<Account, AccountDto>();
        CreateMap<AccountInfo, AccountInfoDto>()
            .ForMember(x => x.SocialNetwork, y => y.MapFrom(src => (ESocialNetwork)src.SocialNetwork))
        CreateMap<Advice, AdviceDto>()
            .ForMember(x => x.AdviceType, y => y.MapFrom(src => (EAdviceType)src.AdviceType));
        CreateMap<SearchHistory, SearchHistoryDto>()
            .ForMember(x => x.SocialNetwork, y => y.MapFrom(src => (ESocialNetwork)src.SocialNetwork))
        CreateMap<User, UserDto>();
    }
}
```

Рис.2.11. Мапер

3. Рівень доступу до даних

Рівень доступу даних містить в собі:

- моделі даних
- контекст (рисунки 2.12 - 2.14).

Як приклад моделі наведено клас - SearchHistory, що представляє собою історію пошукових запитів, і клас Account, що містить в собі дані про юзера, його історію пошукових запитів, і вказані акаунти СМ, а в якості контексту – клас, що має назву - MyDbContext.

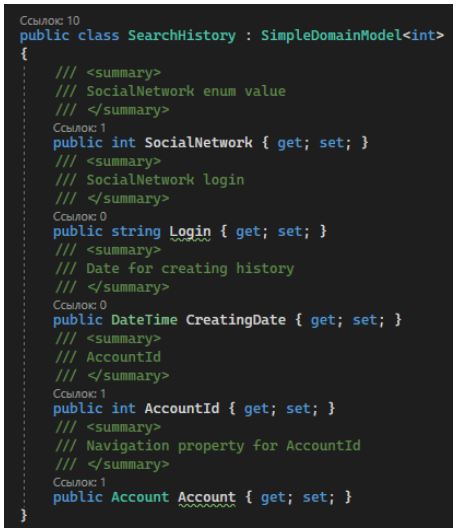


Рис.2.12. Модель історії пошуку

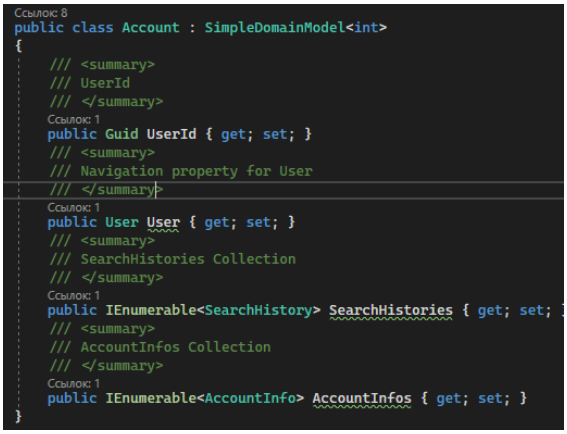
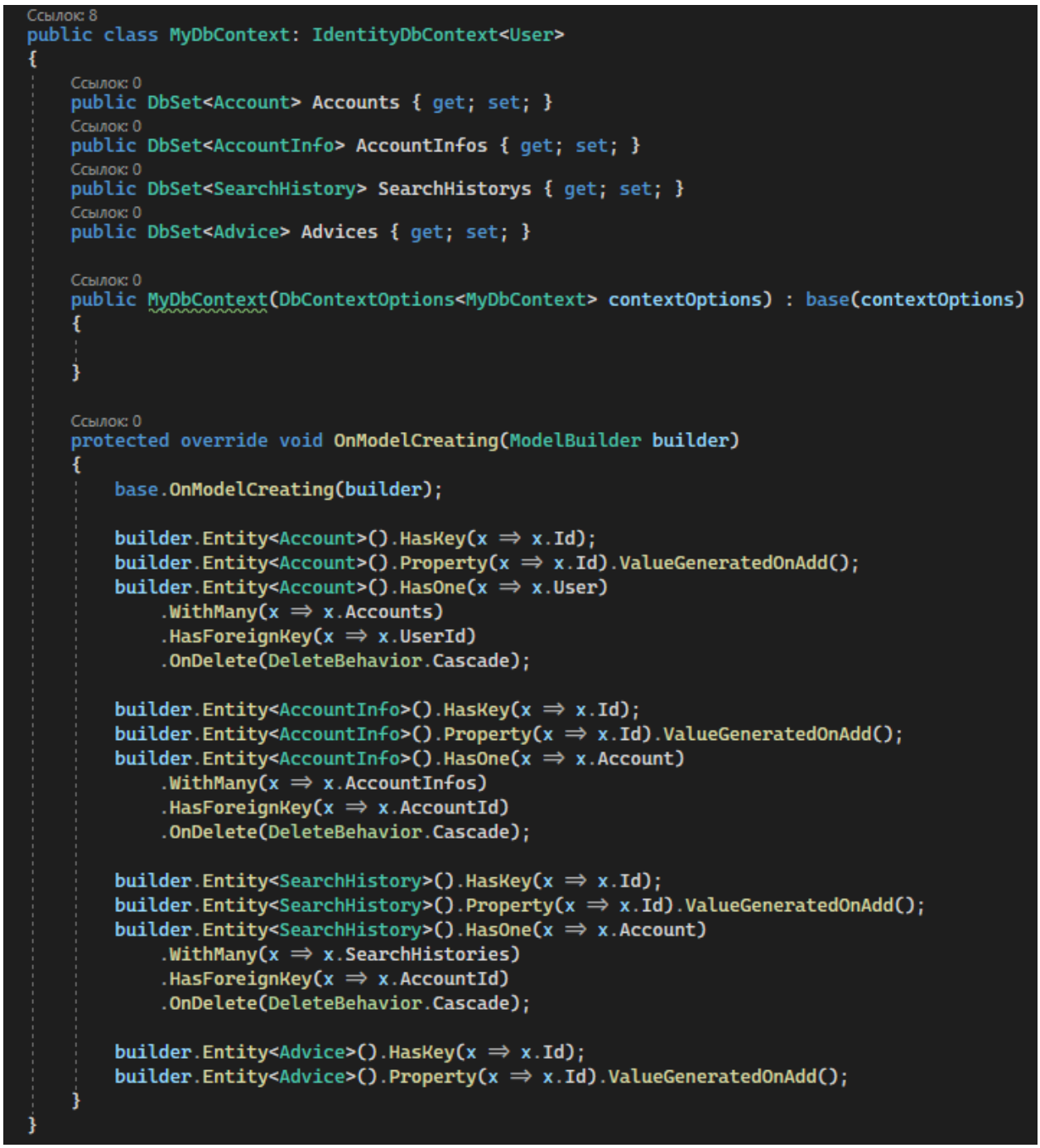


Рис.2.13. Модель проміжної сутності між User і TypeOfHuman



```
Ссылок: 8
public class MyDbContext: IdentityDbContext<User>
{
    Ссылок: 0
    public DbSet<Account> Accounts { get; set; }
    Ссылок: 0
    public DbSet<AccountInfo> AccountInfos { get; set; }
    Ссылок: 0
    public DbSet<SearchHistory> SearchHistorys { get; set; }
    Ссылок: 0
    public DbSet<Advice> Advices { get; set; }

    Ссылок: 0
    public MyDbContext(DbContextOptions<MyDbContext> contextOptions) : base(contextOptions)
    {

    }

    Ссылок: 0
    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.Entity<Account>().HasKey(x => x.Id);
        builder.Entity<Account>().Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Entity<Account>().HasOne(x => x.User)
            .WithMany(x => x.Accounts)
            .HasForeignKey(x => x.UserId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<AccountInfo>().HasKey(x => x.Id);
        builder.Entity<AccountInfo>().Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Entity<AccountInfo>().HasOne(x => x.Account)
            .WithMany(x => x.AccountInfos)
            .HasForeignKey(x => x.AccountId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<SearchHistory>().HasKey(x => x.Id);
        builder.Entity<SearchHistory>().Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Entity<SearchHistory>().HasOne(x => x.Account)
            .WithMany(x => x.SearchHistories)
            .HasForeignKey(x => x.AccountId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<Advice>().HasKey(x => x.Id);
        builder.Entity<Advice>().Property(x => x.Id).ValueGeneratedOnAdd();
    }
}
```

Рис.2.14. Процес створення таблиць і налаштування конекшенів між таблицями за допомогою механізму, що має назву - Fluent Api, який надає Entity Framework Core

Для системи аутентифікації/авторизації в ПЗ використовується ASP.NET Core Identity. Цза допомогою цієї системи користувач можеробити такі функції: створення облікових записів, аутентифікація, керування

обліковими записами або використовувати для входу облікові записи зовнішніх провайдерів. Щоб користуватись цією системою, контекст БД має наслідуватися від батьківського класу - IdentityDbContext і підключити її в Program.cs або Startup.cs (рис.2.15.).

```
builder.Services.AddEntityFrameworkNpgsql()
            .AddDbContext<DbContext, MyDbContext>(optionsAction ⇒ optionsAction.UseNpgsql(
builder.Services.AddIdentity<User, IdentityRole>().AddEntityFrameworkStores<MyDbContext>();
```

Рис.2.15. Підключення Identity до проекту

4. ER-модель бази даних

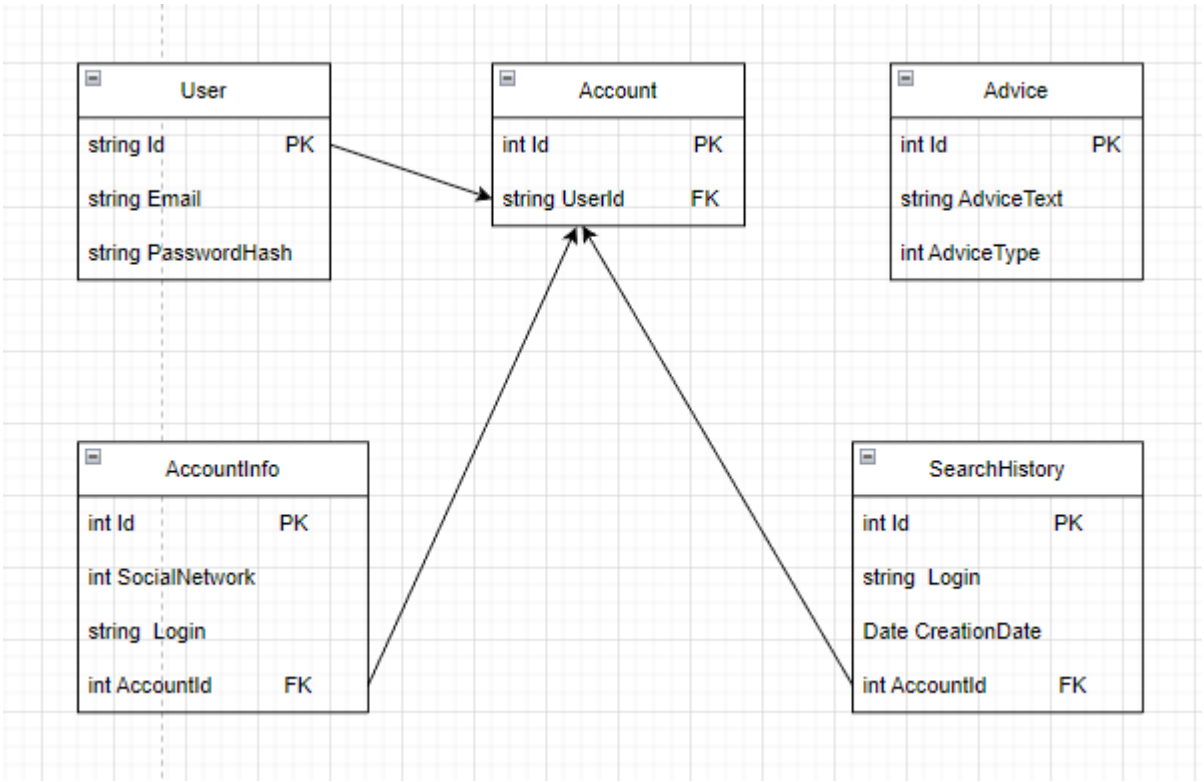


Рис.2.16. Модель бази даних

Висновок до розділу

В розглянутомі вище розділі ми проаналізували вимоги до системи, що проектується, а також розглянули етапи проектування ПЗ. Для формалізації вимог було використано:

- вербальний опис,
- діаграми використання.

Було розглянуто такі вимоги:

- Вимоги користувача;
- Функціональні вимоги;
- Нефункціональні вимоги;
- Системні вимоги:
  - Вимоги до надійності системи;
  - Вимоги до захисту.
- Вимоги до апаратного забезпечення.

РОЗДІЛ 3. ПРЕДСТАВЛЕННЯ СИСТЕМИ

3.1. Опис класів.

Розроблена система містить в собі багато видів класів, а саме:

- Model
- Context
- Interface
- Service
- Controller
- Repository
- Startup
- UnitOfWork
- Mapper
- Enum
- Configuration file
- ViewModel
- View

Model. Модель являє собою однин з ключових компонентів паттерна, що має назву — MVC. Ключовим завданням моделі є:

- опис структури,
- логіки даних.

Моделі в проектах описують сутності, а також їх структури, саме тому всі суті, що будуть використовувані в додатку, зазвичай виділяють в окремі моделі. Кількість моделей в проекті напряму залежить від завдань і предметної області. У проекті як приклад наведена модель, що представляє аккаунт юзера (рис. 3.1).

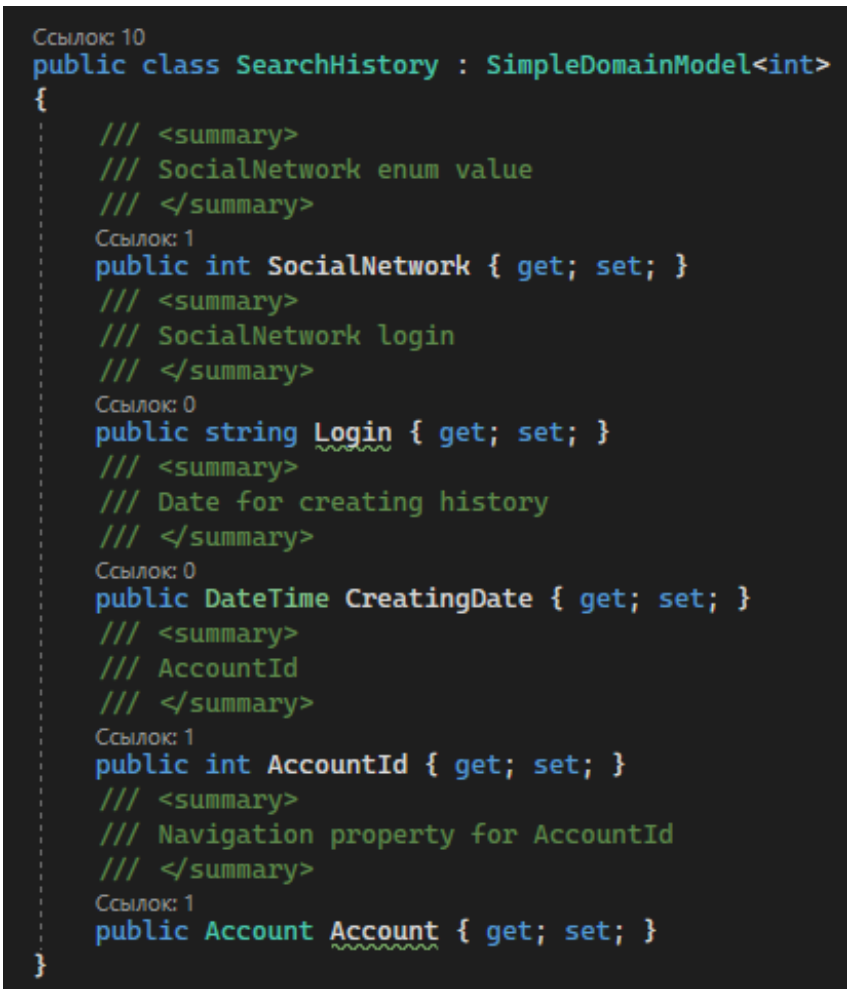


Рис .3.1. Приклад моделі з проекту

Context. Клас контексту визначає контекст даних і використовується для взаємодії з БД. Набір об'єктів у класі контексту даних представляє клас, що має назву - DbSet <T>. Всі створені моделі є прямим відображенням таблиці в БД (рис. 3.2). Крім того, для налаштування підключення ми повинні перевизначити метод, що має назву - OnConfiguring. Параметр класу DbContextOptionsBuilder, що передається за допомогою методу, що має назву - UseSqlServer дозволяє налаштувати рядок підключення для з'єднання з сервером БД. Ми визначаємо, що як сервер буде використано - localdb, який призначений спеціально для розробки.

```
Ссылок: 8
public class MyDbContext: IdentityDbContext<User>
{
    Ссылок: 0
    public DbSet<Account> Accounts { get; set; }
    Ссылок: 0
    public DbSet<AccountInfo> AccountInfos { get; set; }
    Ссылок: 0
    public DbSet<SearchHistory> SearchHistorys { get; set; }
    Ссылок: 0
    public DbSet<Advice> Advices { get; set; }

    Ссылок: 0
    public MyDbContext(DbContextOptions<MyDbContext> contextOptions) : base(contextOptions)
    {

    }

    Ссылок: 0
    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.Entity<Account>().HasKey(x => x.Id);
        builder.Entity<Account>().Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Entity<Account>().HasOne(x => x.User)
            .WithMany(x => x.Accounts)
            .HasForeignKey(x => x.UserId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<AccountInfo>().HasKey(x => x.Id);
        builder.Entity<AccountInfo>().Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Entity<AccountInfo>().HasOne(x => x.Account)
            .WithMany(x => x.AccountInfos)
            .HasForeignKey(x => x.AccountId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<SearchHistory>().HasKey(x => x.Id);
        builder.Entity<SearchHistory>().Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Entity<SearchHistory>().HasOne(x => x.Account)
            .WithMany(x => x.SearchHistories)
            .HasForeignKey(x => x.AccountId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.Entity<Advice>().HasKey(x => x.Id);
        builder.Entity<Advice>().Property(x => x.Id).ValueGeneratedOnAdd();
    }
}
```

Рис.3.2 Приклад контексту з проекту

Interface. Інтерфейс є типом посилання, що може визначати певний функціонал:

- набір методів,
- властивості без реалізації.

Потім класи, які застосовують даний інтерфейс, мають його реалізувати. В цьому інтерфейсі є визначеними методи для:

- отримання колекції,
- створення,
- оновлення,
- видалення об’єкту,

а потім цей інтерфейс використовується у сервісах, як Dependency Injection, щоб абстрагуватись від сервісів, які його реалізують, посуті це є апкастом (рис. 3.3).

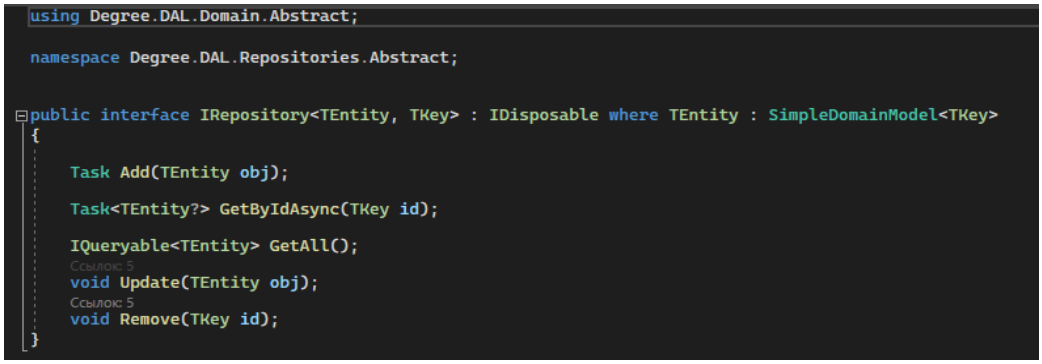


Рис.3.3 Приклад інтерфейсу з проекту

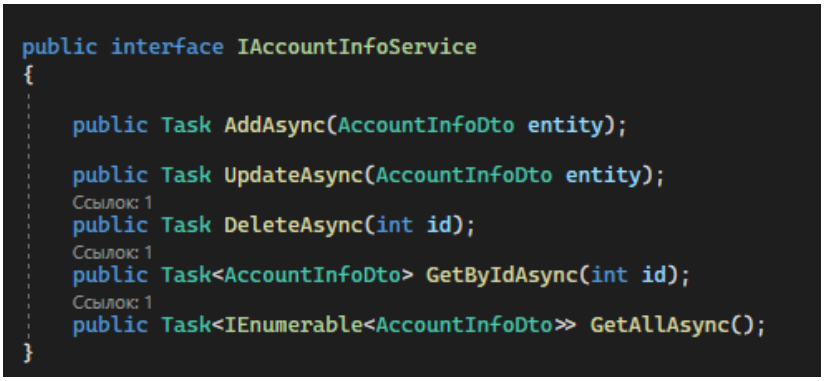
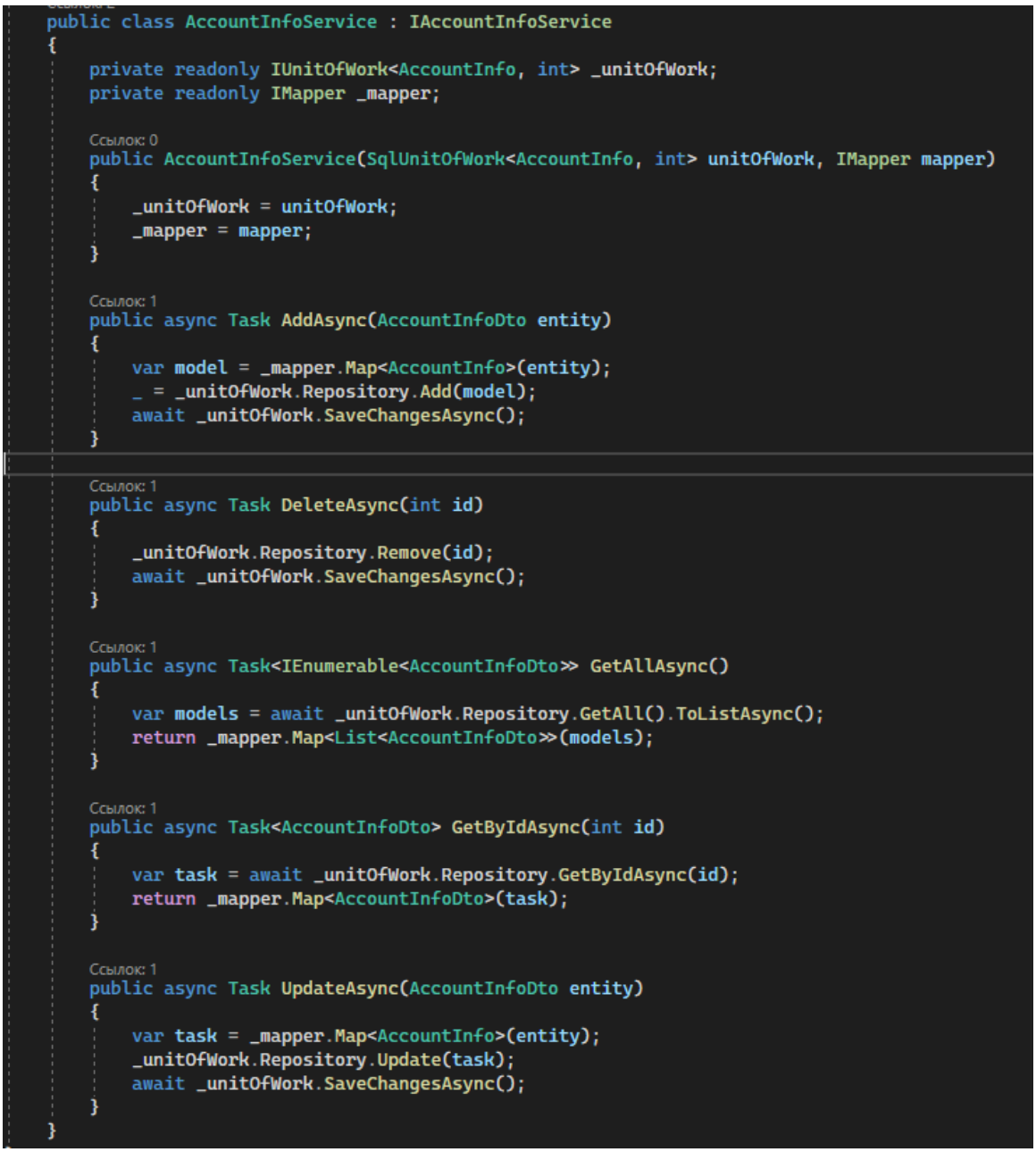


Рис.3.4 Приклад інтерфейсу з проекту

Service. Сервіси реалізують логіку для роботи з даними, а саме CRUD операції, та ін. Використовуючи сервіси, ми можемо проводити різного виду маніпуляції з даними:

- створення,
- оновлення,
- видалення,
- зчитування записів,
- роботи зі списками,
- фільтрація даних,
- сортування даних,
- хешування даних,
- обробка колекцій,
- більш складні логічні операції.

Також в сервісах часто використовують методи, які створені для сортування даних по тим чи іншим певним критеріям за допомогою інструменту Linq, що являється мовою запитів до джерел даних. Ще одним не менш важливим фактом є те, що в розробленому сервісі (рис. 3.5), відбувається реалізуація інтерфейсу, в якому попередньо було визначено певний базовий функціонал, який необхідний саме для цього виду сервісів. Таким чином, цей зв’язок, що ми використали interface — service, звісно ж робить нашу програму більш гнучкою і правильної зі сторони підходів і основ об’єктно орієнтованих мов програмування.



```
public class AccountInfoService : IAccountInfoService
{
    private readonly IUnitOfWork<AccountInfo, int> _unitOfWork;
    private readonly IMapper _mapper;

    Ссылок: 0
    public AccountInfoService(SqlUnitOfWork<AccountInfo, int> unitOfWork, IMapper mapper)
    {
        _unitOfWork = unitOfWork;
        _mapper = mapper;
    }

    Ссылок: 1
    public async Task AddAsync(AccountInfoDto entity)
    {
        var model = _mapper.Map<AccountInfo>(entity);
        _ = _unitOfWork.Repository.Add(model);
        await _unitOfWork.SaveChangesAsync();
    }

    Ссылок: 1
    public async Task DeleteAsync(int id)
    {
        _unitOfWork.Repository.Remove(id);
        await _unitOfWork.SaveChangesAsync();
    }

    Ссылок: 1
    public async Task<IEnumerable<AccountInfoDto>> GetAllAsync()
    {
        var models = await _unitOfWork.Repository.GetAll().ToListAsync();
        return _mapper.Map<List<AccountInfoDto>>(models);
    }

    Ссылок: 1
    public async Task<AccountInfoDto> GetByIdAsync(int id)
    {
        var task = await _unitOfWork.Repository.GetByIdAsync(id);
        return _mapper.Map<AccountInfoDto>(task);
    }

    Ссылок: 1
    public async Task UpdateAsync(AccountInfoDto entity)
    {
        var task = _mapper.Map<AccountInfo>(entity);
        _unitOfWork.Repository.Update(task);
        await _unitOfWork.SaveChangesAsync();
    }
}
```

Рис. 3.5. Приклад одного із сервісів проекту

Controller. У архітектурі ASP.NET Core MVC, що в нас представлено, центральною ланкою є контролер. Коли створюється певний запит до якогось контролера підключається система маршрутизації, що займається підбиранням потрібного контролера для обробки надійшовшого для нього запиту. Контроллер після отримання запиту обробляє данні, що прийшли в запиті і посилає результат через ту чи іншу відповідь, в залежності від результату обробки. Контролер у ASP.NET Core MVC є ні чи іншим, як

звичайний класс c#, який наслідується від батьківського класу, що має назву - Microsoft.AspNetCore.Mvc.Controller. Зазвичай, при створенні проекту, в ньому міститься лише один контролер, що має назву - HomeController. Для обробки запитів зазвичай використовуються Api-контролери. Дані контролери обробляють запити HTTP:

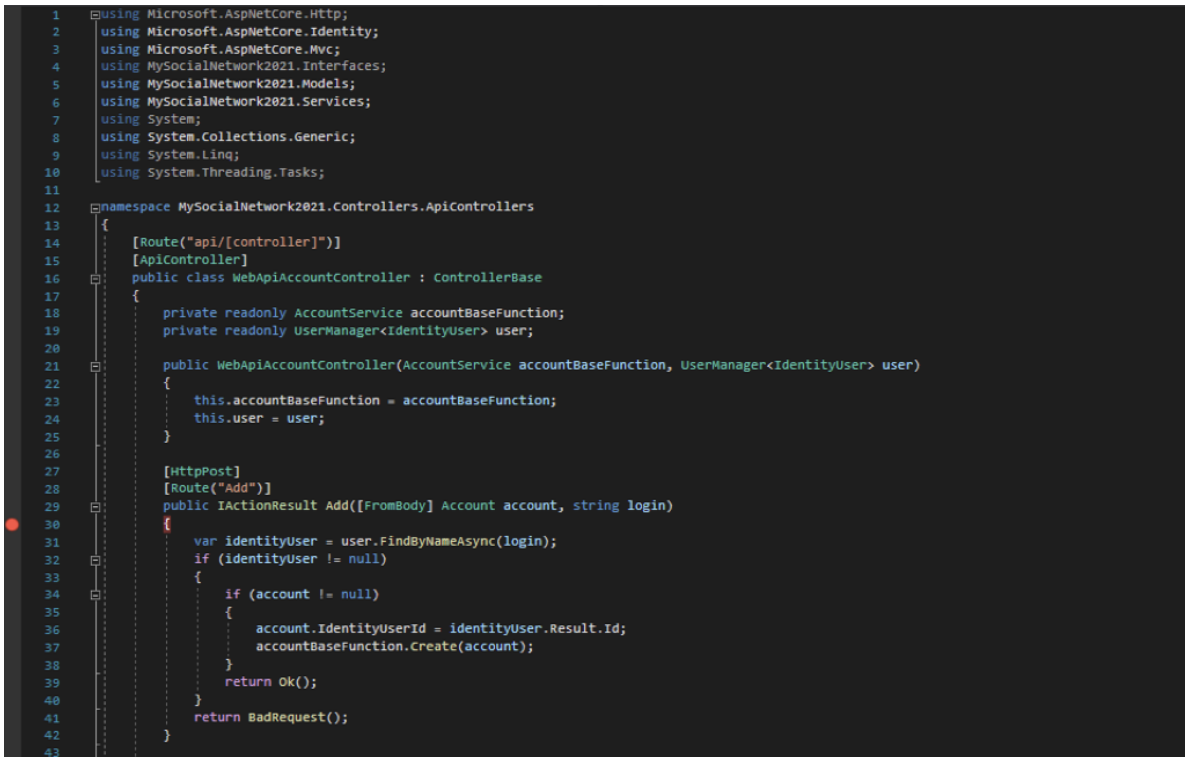
- Get,
- Post,
- Put,
- Delete,
- Patch,
- Head,
- Options.

Зазвичай при використанні таких контролерів, нам надається можливість контролювати стан обробки запиту на сервері за допомогою статусних кодів:

- 200- вказує на успішне виконання.
- 201- вказує на успішне створення об’єкту.
- 204- вказує що, запит пройшов успішно.
- 400- вказує що, відбулася помилка при виконанні.
- 401- вказує на те, що користувач не авторизований.
- 403- звказує на заборонений доступ.
- 404- вказує на те, що ресурс не був знайдений.

В ПЗ, що спроектовано, бкло зроблено відповідний метод для кожного типу запиту. Отже, коли було розподілено весь функціонал між різними контролерам, було чітко реалізувано паттерн Solid, а саме *(The Single*

Responsibility Principle), що говорить про те, що кожному контролеру було надано особисту відповідальність.



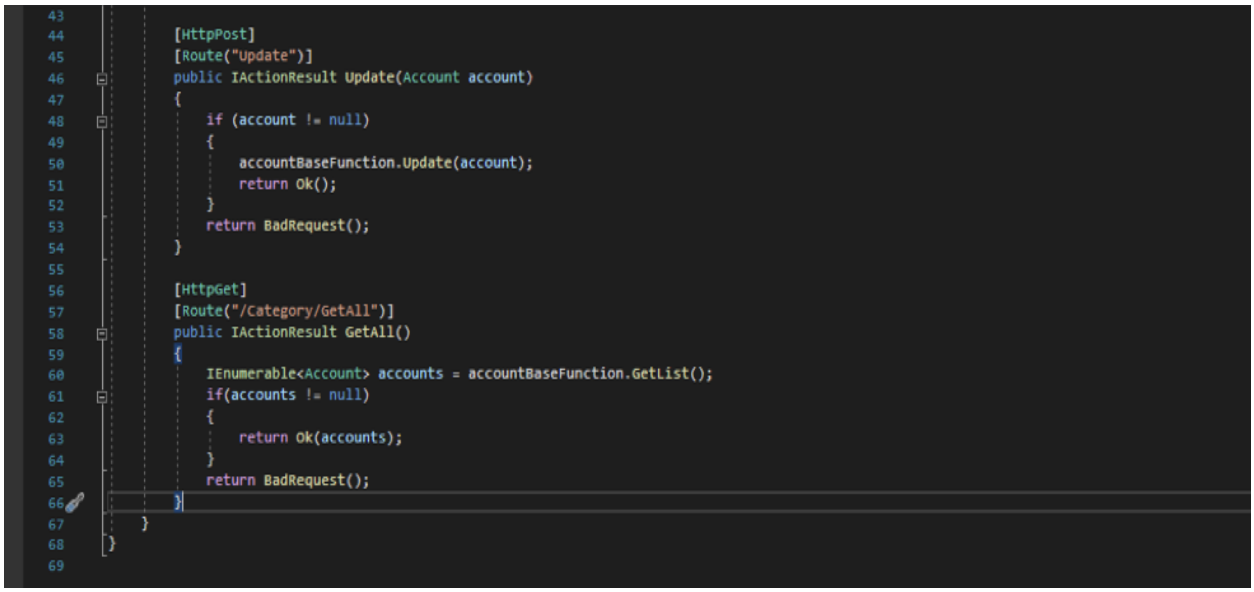
```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using MySocialNetwork2021.Interfaces;
using MySocialNetwork2021.Models;
using MySocialNetwork2021.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace MySocialNetwork2021.Controllers.ApiControllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class WebApiAccountController : ControllerBase
    {
        private readonly AccountService accountBaseFunction;
        private readonly UserManager<IdentityUser> user;

        public WebApiAccountController(AccountService accountBaseFunction, UserManager<IdentityUser> user)
        {
            this.accountBaseFunction = accountBaseFunction;
            this.user = user;
        }

        [HttpPost]
        [Route("Add")]
        public IActionResult Add([FromBody] Account account, string login)
        {
            var identityUser = user.FindByNameAsync(login);
            if (identityUser != null)
            {
                if (account != null)
                {
                    account.IdentityUserId = identityUser.Result.Id;
                    accountBaseFunction.Create(account);
                }
                return Ok();
            }
            return BadRequest();
        }
```

Рис. 3.6. Приклад контролера з проекту



```
        [HttpPost]
        [Route("Update")]
        public IActionResult Update(Account account)
        {
            if (account != null)
            {
                accountBaseFunction.Update(account);
                return Ok();
            }
            return BadRequest();
        }

        [HttpGet]
        [Route("/Category/GetAll")]
        public IActionResult GetAll()
        {
            IEnumerable<Account> accounts = accountBaseFunction.GetList();
            if(accounts != null)
            {
                return Ok(accounts);
            }
            return BadRequest();
        }
    }
}
```

Рис.3.7 Приклад контролеру з проекту

Repository. Репозиторій дає можливість абстрагуватися від конкретних підключень до БД. Наприклад, в нас є одне підключення до БД, що має назву - MS SQL Server, однак, якщо ми захочемо змінити підключення в якийсь момент з MS SQL, то ми робимо окремий репозиторій для того підключення і в UnitOfWork його підставляємо По стандарту, навіть у невеликому додатку, нам доведеться зробити велику кількість змін при тих чи інших обставинах. Або під час роботи ПЗ ми хочемо використати два різні підключення і знову ж з цм не виникне ні яких проблем. Тому, проаналізувавши все, можна чітко сказати, що репозиторій додає програмі гнучкість.

```
public class SqlRepository<TEntity, TKey> : IRepository<TEntity, TKey> where TEntity : SimpleDomainModel<TKey>
{
    protected readonly MyDbContext _context;
    protected readonly DbSet<TEntity> DbSet;

    Ссылок: 0
    public SqlRepository(MyDbContext context)
    {
        _context = context;
        DbSet = _context.Set<TEntity>();
    }

    Ссылок: 0
    public void Dispose()
    {
        _context.Dispose();
        GC.SuppressFinalize(this);
    }

    Ссылок: 5
    public virtual Task Add(TEntity model)
    {
        DbSet.Add(model);
        return Task.CompletedTask;
    }

    Ссылок: 5
    public async Task<TEntity?> GetByIdAsync(TKey id) => await DbSet.FindAsync(id);

    Ссылок: 5
    public IQueryable<TEntity> GetAll() => DbSet.AsQueryable();

    Ссылок: 5
    public virtual void Update(TEntity model)
    {
        DbSet.Update(model);
    }

    Ссылок: 5
    public virtual void Remove(TKey id)
    {
        DbSet.Remove(DbSet.Find(id));
    }
}
```

Рис.3.8 Приклад репозиторію з проекту

Startup. Клас Startup є вхідною точкою до програми ASP.NET Core. Цей клас здійснює конфігурацію програми, налаштовує сервіси, які програма буде використовувати, встановлює компоненти для обробки запиту або middleware.

Клас Startup повинен визначати метод Configure(), а також опціонально в Startup можна визначити конструктор класу та метод ConfigureServices().

При запуску програми спочатку спрацьовує конструктор, потім метод ConfigureServices() та в кінці метод Configure(). Ці методи викликаються середовищем виконання ASP.NET.конвеєра. Спочатку дані запиту отримує перший компонент конвеєрі. Після обробки він передає дані HTTP-запиту другому компоненту тощо. Ці компоненти конвеєра, які відповідають за обробку запиту, називаються middleware. В ASP.NET Core для підключення компонентів middleware використовується метод Configure із класу Startup.

Компонент middleware може або передати запит далі наступному в конвеєрі компоненту, або виконати обробку та закінчити роботу конвеєра. Також компонент middleware у конвеєрі може виконувати обробку запиту як до, так і після наступного у конвеєрі компонента.

Необов'язковий метод ConfigureServices реєструє сервіси, що використовуються додатком. Як параметр він приймає об'єкт IServiceCollection, який представляє колекцію сервісів у додатку. За допомогою методів розширення цього об'єкта проводиться конфігурація програми для використання сервісів.

Метод Configure встановлює, як програма оброблятиме запит. Цей метод є обов'язковим. Для встановлення компонентів, які обробляють запит, використовуються методи об'єкта IApplicationBuilder. Об'єкт IApplicationBuilder є обов'язковим параметром для Configure.

Крім того, метод нерідко приймає ще один необов'язковий параметр - об'єкт IWebHostEnvironment, який дозволяє отримати інформацію про середовище, в якому запускається програма, та взаємодіяти з нею.

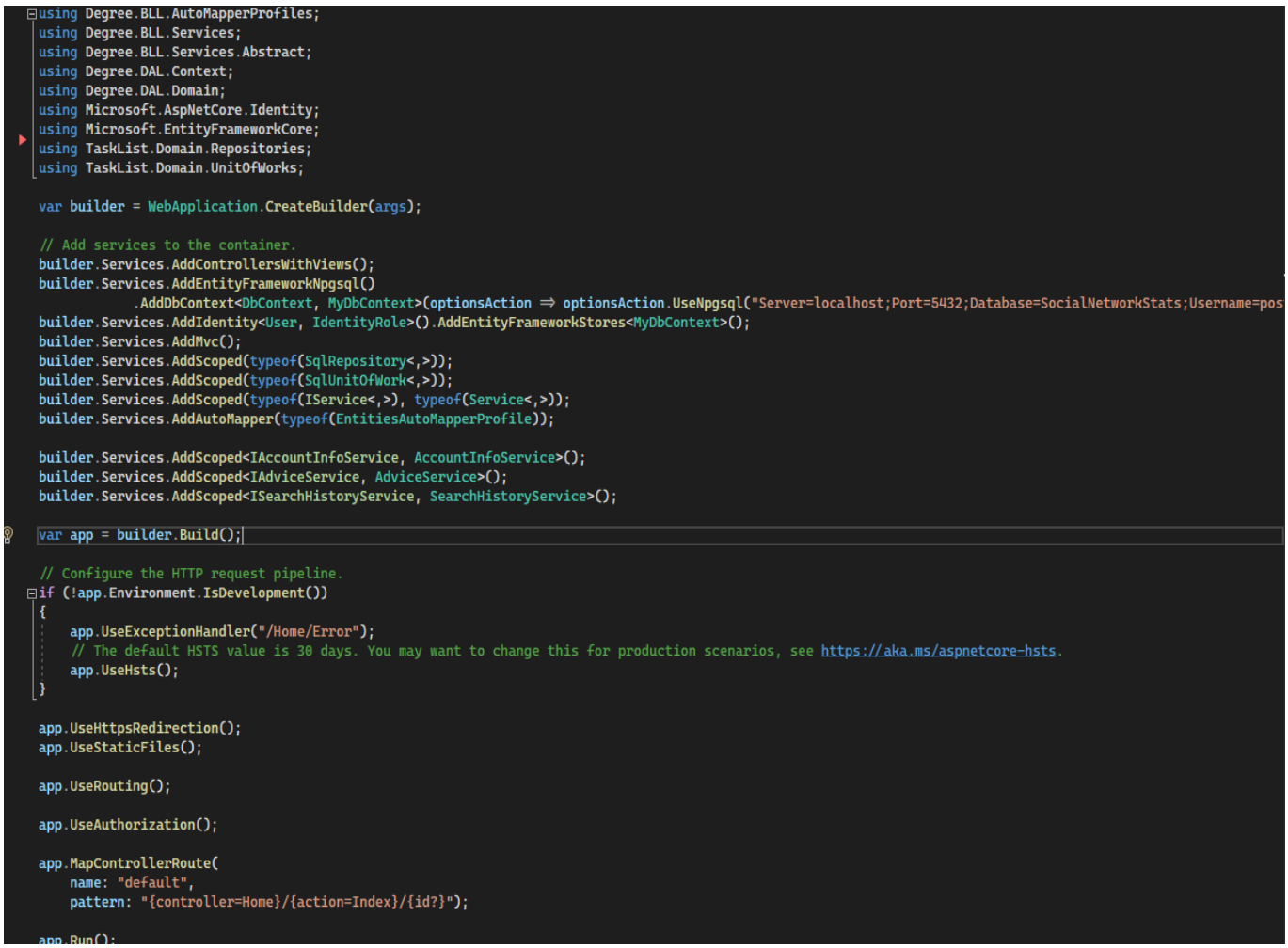


Рис.3.9 Приклад Sturtup з проекту

UnitOfWork. В нашому ПЗ використовується шаблон, що має назву - Repository, яка слугує для інкапсуляції логіки роботи з БД. Часто при написані пром, доводиться оперувати великою кількістю сутностей та моделей. А для того, щоб управляти ними створюється відповідно велика кількість репозиторіїв. Unit of Work допомагає спрощувати роботу з різними репозиторіями і дає можливість бути впевненими, що всі репозиторії будуть використовувати той самий DbContext.

Ще одним плюсом використання шаблону Репозиторій і Unit of Work є створення правильної структури для розгортання програми та впровадження в неї DI практик.

```
public interface IUnitOfWork<TEntity, TKey> : IDisposable where TEntity : SimpleDomainModel<TKey>
{
    IRepository<TEntity, TKey> Repository { get; set; }
    Task<bool> SaveChangesAsync();
}
```

Рис.3.10 Приклад інтерфейсу IUnitOfWork з проекту

```
public class SqlUnitOfWork<TEntity, TKey> : IDisposable, IUnitOfWork<TEntity, TKey> where TEntity : SimpleDomainModel<TKey>
{
    private readonly MyDbContext _context;

    public IRepository<TEntity, TKey> Repository { get; set; }

    public SqlUnitOfWork(MyDbContext context, SqlRepository<TEntity, TKey> repository)
    {
        _context = context;
        Repository = repository;
    }

    public Task<bool> SaveChangesAsync()
    {
        var res = _context.SaveChanges();
        return Task.FromResult(res > 0);
    }

    public void Dispose()
    {
        _context.Dispose();
        GC.SuppressFinalize(this);
    }

}
```

Рис.3.11 Приклад UnitOfWork з проекту

Mapper. Mapper слугує для того, щоб робити проекції моделей БД на моделі (DTO). Для mapper в проекті використовується NuGet-пакет AutoMapper.

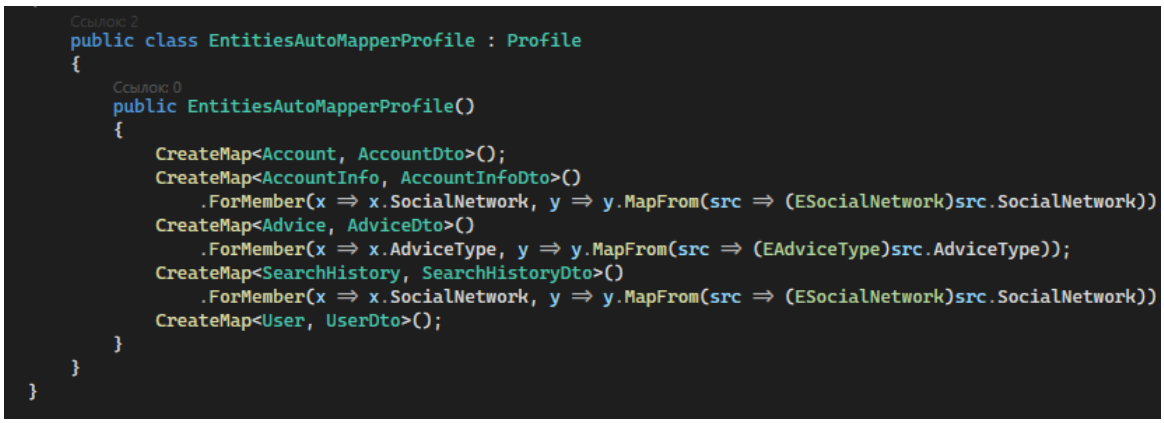


Рис.3.12 Приклад Mapper з проекту

Enum. Enum представляє набір констант, що логічно пов'язані. З кожним новим перехуванням фактично визначається новий тип даних. За допомогою визначення нових типів даних ми можемо визначати:

- змінні,
- константи,
- параметри методів і т.д.

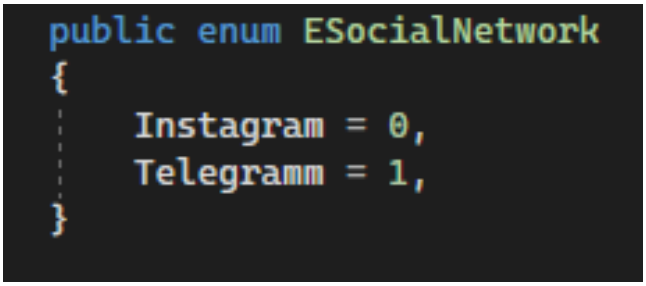


Рис.3.13 Приклад Enum з проекту

Configuration file. При створенні програми, автоматично створюється конфігурація програми у вигляді об'єкта IConfiguration. Тому всю необхідну конфігурацію програми ми завжди можемо отримати як і будь-який інший сервіс через механізм застосування залежностей.

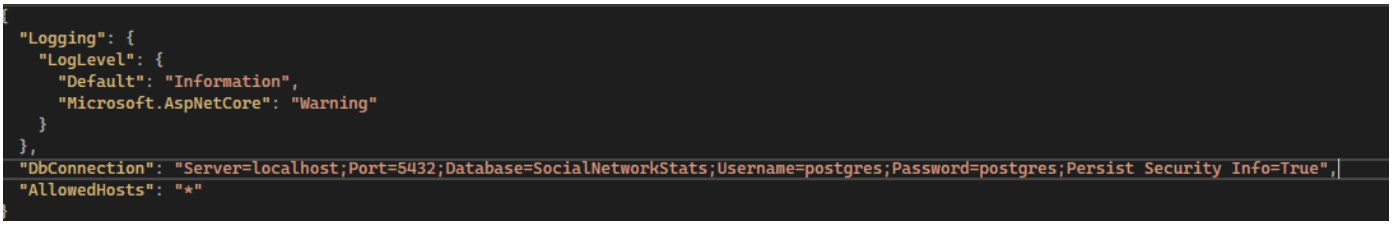


Рис.3.14 Приклад файлу конфігурацій з проекту

ViewModel. Модель представлення - це модель, яка створена для того, щоб передавати дані на сторінку або зі сторінки, для отримання даних з View.

```
public class HistoryViewModel
{
    Ссылок: 0
    public int SocialNetwork { get; set; }

    Ссылок: 0
    public string Login { get; set; }

    Ссылок: 0
    public DateTime CreatingDate { get; set; }

    Ссылок: 0
    public int AccountId { get; set; }

    Ссылок: 0
    public int AccountInfoId { get; set; }

}
```

Рис.3.15 Приклад моделі представлення з проекту

View. Коли користувач звертається до веб-додатку, він отримує веб-сторінку з даними. У MVC для цього використовуються уявлення, що повністю визначають зовнішній вигляд ПЗ, тому на їх основі формується веб-сторінка. В ASP.NET MVC Core таке поняття, як уявлення — є файлом, що

має розширення — cshtml. Цей файл містить код інтерфейсу який при компіляції транслюється для відображення користувацького інтерфейсу у тому вигляді, який йому задавали.

```
@{
    ViewData["Title"] = "Home Page";
}

<!DOCTYPE html>
<html lang="en">

<head>

    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">

    <title>Dashboard</title>

    <!-- Custom fonts for this template-->
    <link href="vendor/fontawesome-free/css/all.min.css" rel="stylesheet" type="text/css">
    <link
        href="https://fonts.googleapis.com/css?family=Nunito:200,200i,300,300i,400,400i,600,600i,700,700i,800,800i,900,900i"
        rel="stylesheet">

    <!-- Custom styles for this template-->
    <link href="css/sb-admin-2.min.css" rel="stylesheet">

</head>

<body id="page-top">

    <!-- Page Wrapper -->
    <div id="wrapper">

        <!-- Sidebar -->
        <ul class="navbar-nav bg-gradient-primary sidebar sidebar-dark accordion" id="accordionSidebar">

            <!-- Sidebar - Brand -->
            <a class="sidebar-brand d-flex align-items-center justify-content-center" href="index.html">
                <div class="sidebar-brand-icon rotate-n-15">
                    <i class="fas fa-laugh-wink"></i>
                </div>
                <div class="sidebar-brand-text mx-3">SB Admin <sup>2</sup></div>
            </a>

            <!-- Divider -->
            <hr class="sidebar-divider my-0">

            <!-- Nav Item - Dashboard -->
            <li class="nav-item active">
                <a class="nav-link" href="index.html">
```

Рис.3.16 Приклад представлення з проекту

3.2. Опис інтерфейсу та реалізації функціоналу

Процес реєстрації і входу в профіль користувача

Функціонал реєстрації та вхідо у профіль користувача було реалізовано використовуючи ASP.NET Core Identity.

Сама схема реєстрація користувача є дефолтною: «заповнюєш форму необхідними даними + натискаєш кнопку для підтвердження реєстрації = успішна реєстрація».

Сама схема аутентифікації є ще простішою: «ваповнюєш форму необхідними даними + натискаєш кнопку підтвердження входу в систему = аутентифікований користувач».

В схемі, коли необхідно змінити пароль, якщо його забули, все є ще простішим: «вводиш свою пошту у форму + натискаєш кнопку підтвердження зміни паролю = отримуєш новий сформований пароль на пошті, що була вказана».

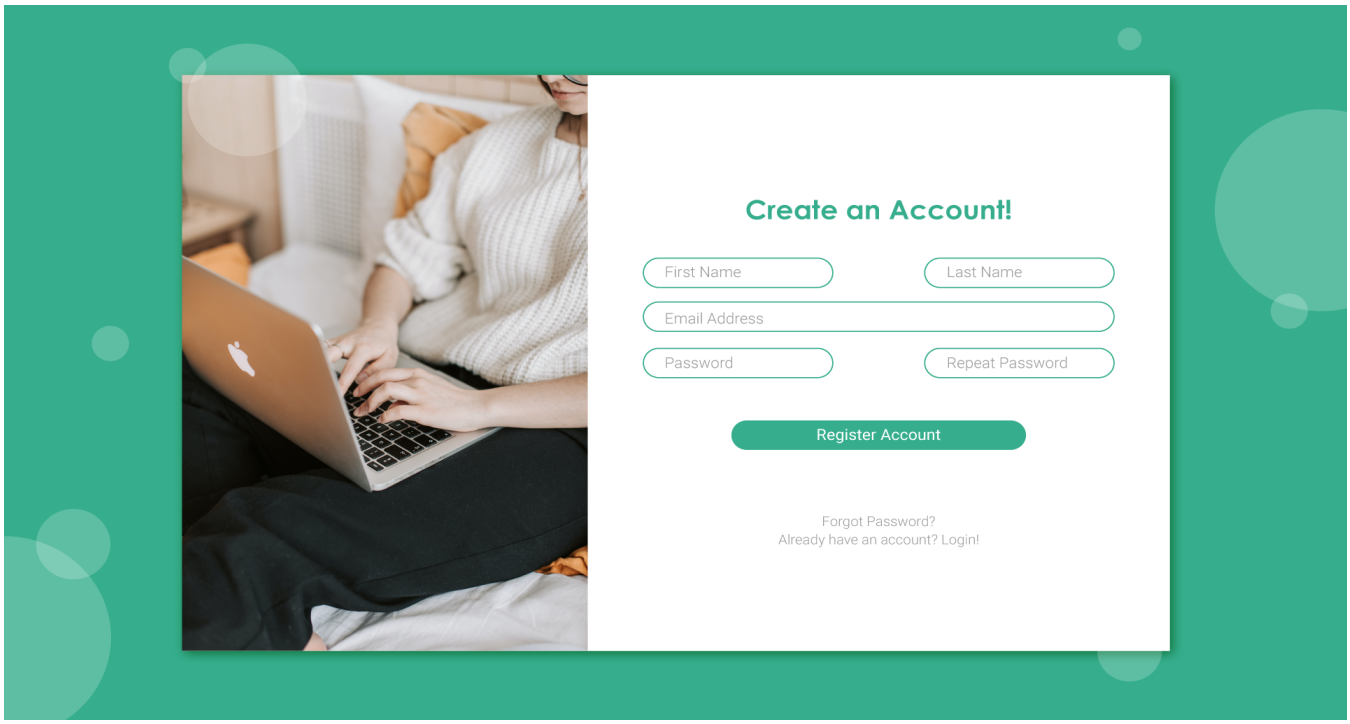


Рис. 3.17. Сторінка реєстрації

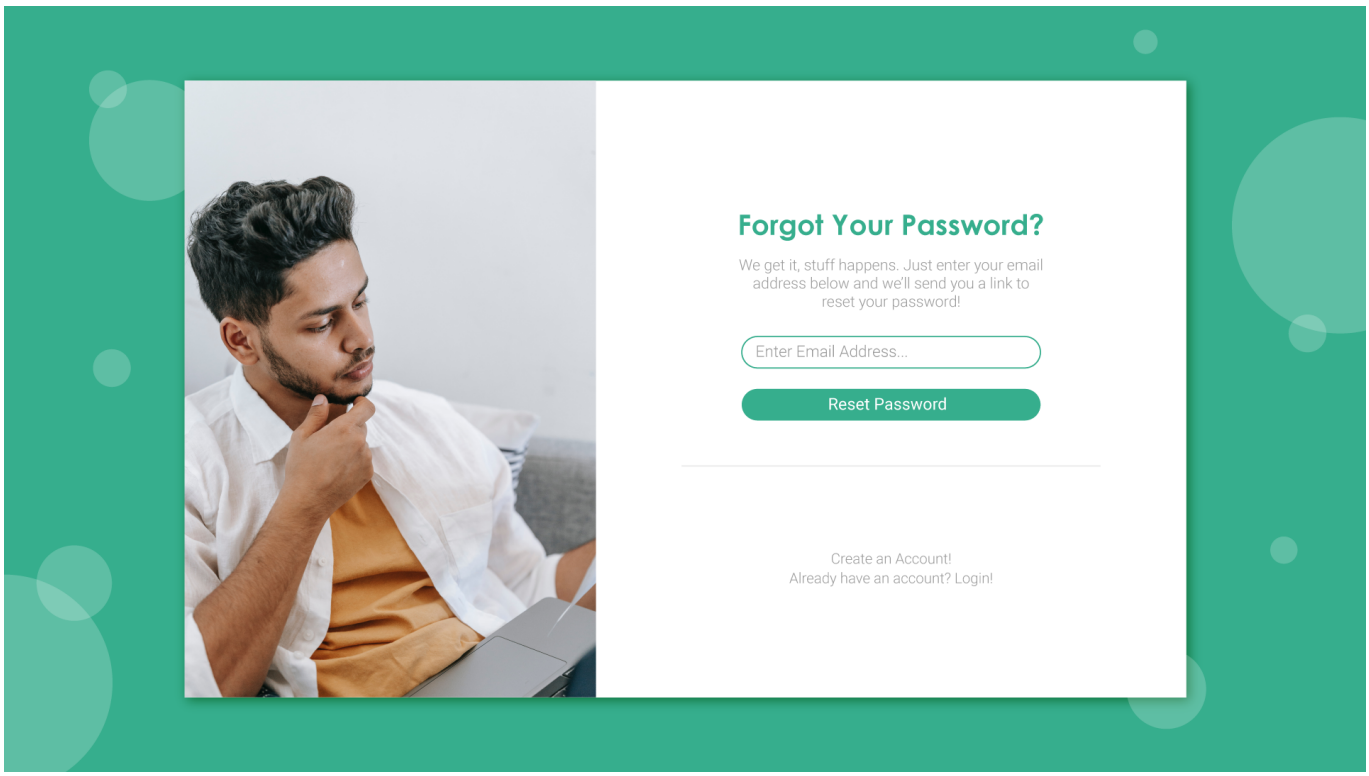


Рис.3.18 Сторінка зміни паролю

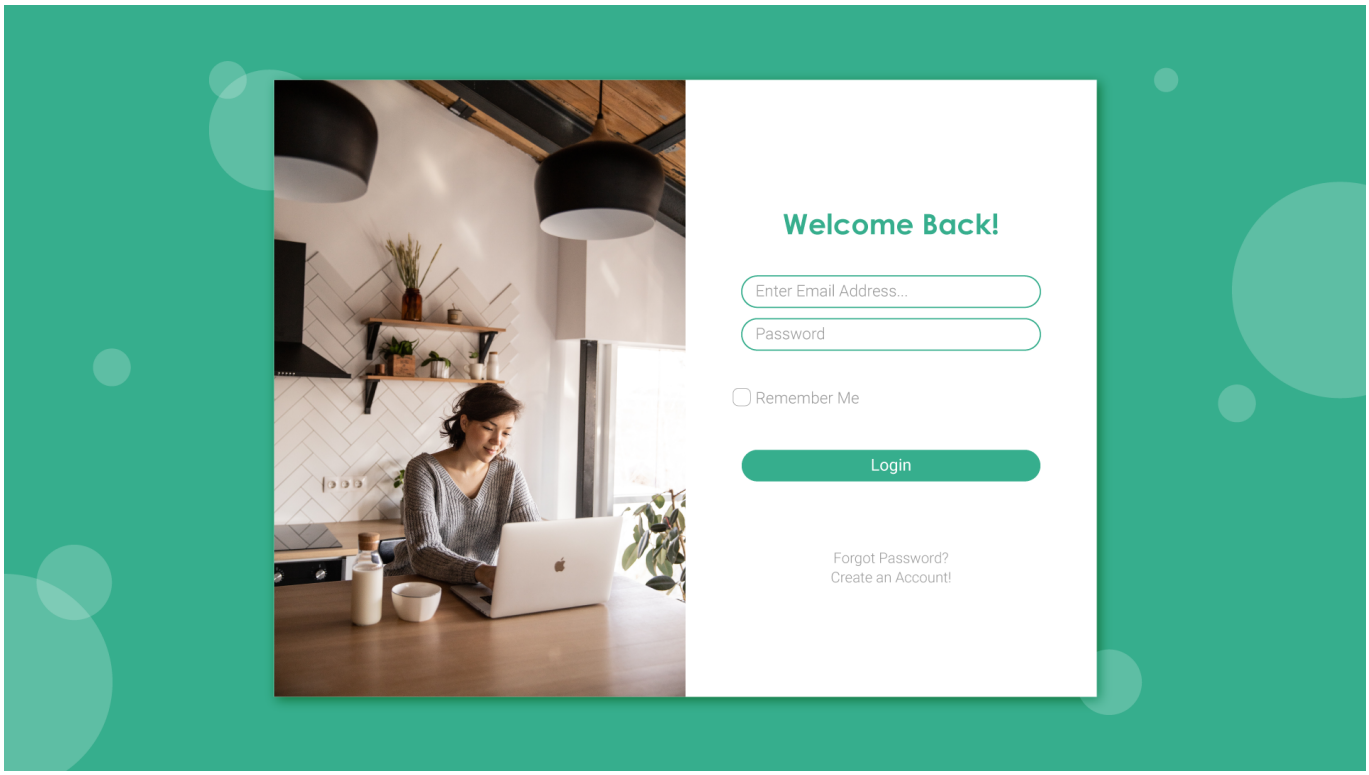


Рис.3.19 Сторінка авторизації

Головна сторінка системи

Головна сторінка системи охоплює такі можливості:

- Можна продивитись ознайомитись з основним функціоналом
- Можна зробити дослідження своєї СМ без реєстрації
- Можна провести авторизацію або реєстрацію

Можливість зробити дослідження акаунту СМ без реєстрації була зроблена спеціально для людей, в яких є необхідність терміново провести дослідження або провести дослідження один тестовий раз і в них нема бажання для цього проводити реєстрацію.

Дослідження СМ з реєстрацією має свої переваги:

- Кожне проведене дослідження СМ записується в БД
- На сторінці акаунту є можливість заповнити дані про свої СМ і використати їх для дослідження

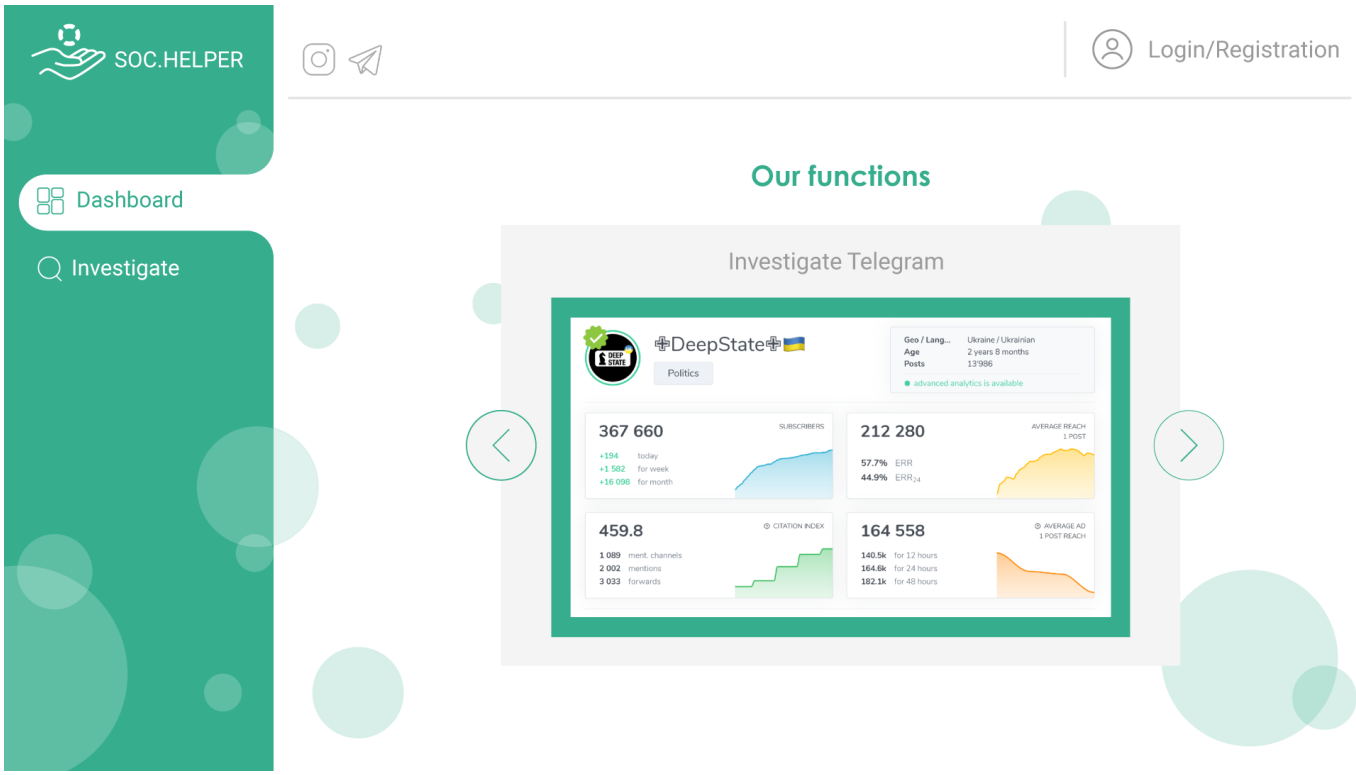


Рис.3.20 Головна сторінка

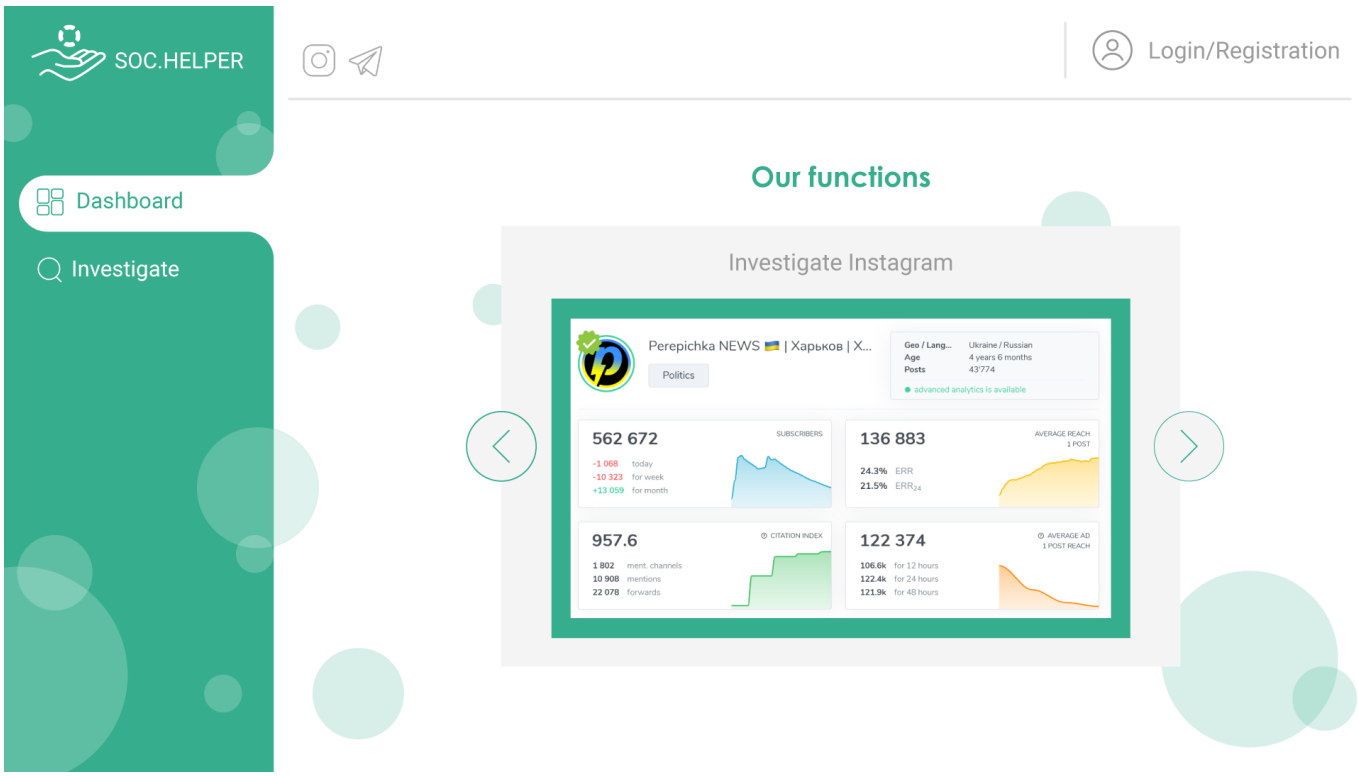


Рис.3.21 Головна сторінка

Сторінка аккаунту користувача

Сторінка акаунту надає можливість використовувати функціонал, який покращить роботу з дослідженням СМ користувача:

- є можливість заповнити дані про свої СМі використовувати їх під час дослідження;
- є можливість здійснити перехід на сторінку Досліджень СМ і там провести необхідні дослідження з своїми СМ;
- є можливість здійснити перехід на сторінку Історій і переглянути історію пошукових запитів, що були здійнені при дослідженні СМ;
- є можливість доповнити дані аккаунту.

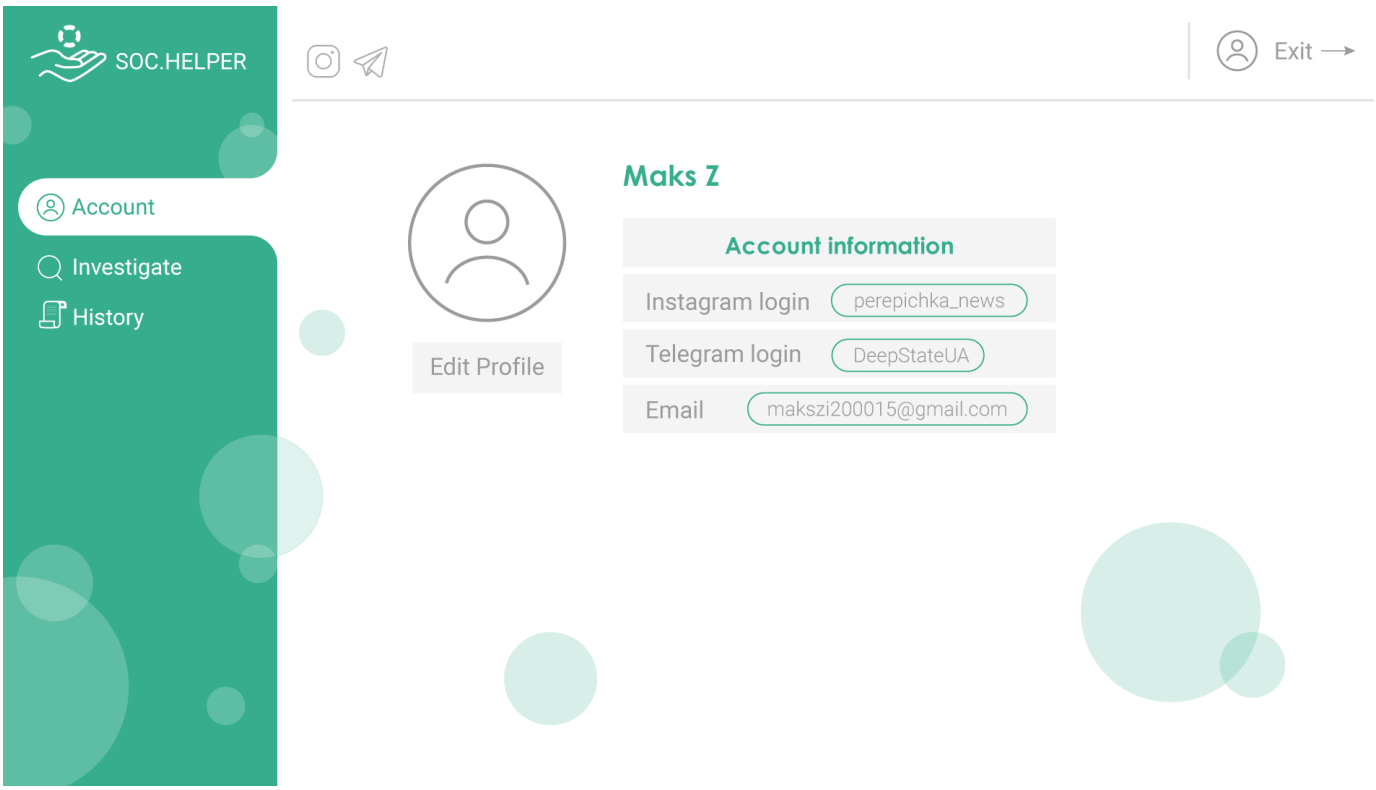


Рис.3.22 Сторінка аккаунта

Сторінка історій пошукових запитів

Сторінка історій пошукових запитів надає такий функціонал:

- є можливість переглянути історію пошукових запитів в яких записуються такі дані:
    1. назва СМ;
    2. логін для потрібної СМ;
    3. дата створення і запису пошукового запиту.
- є можливість відтворити запит по дослідженню СМ по тим даним, що були записані в історію і зробити нове дослідження СМ з актуальною інформацією;
- є можливість здійснити перехід на сторінку досліджень;
- є можливість здійснити перехыд на сторінку акаунту.

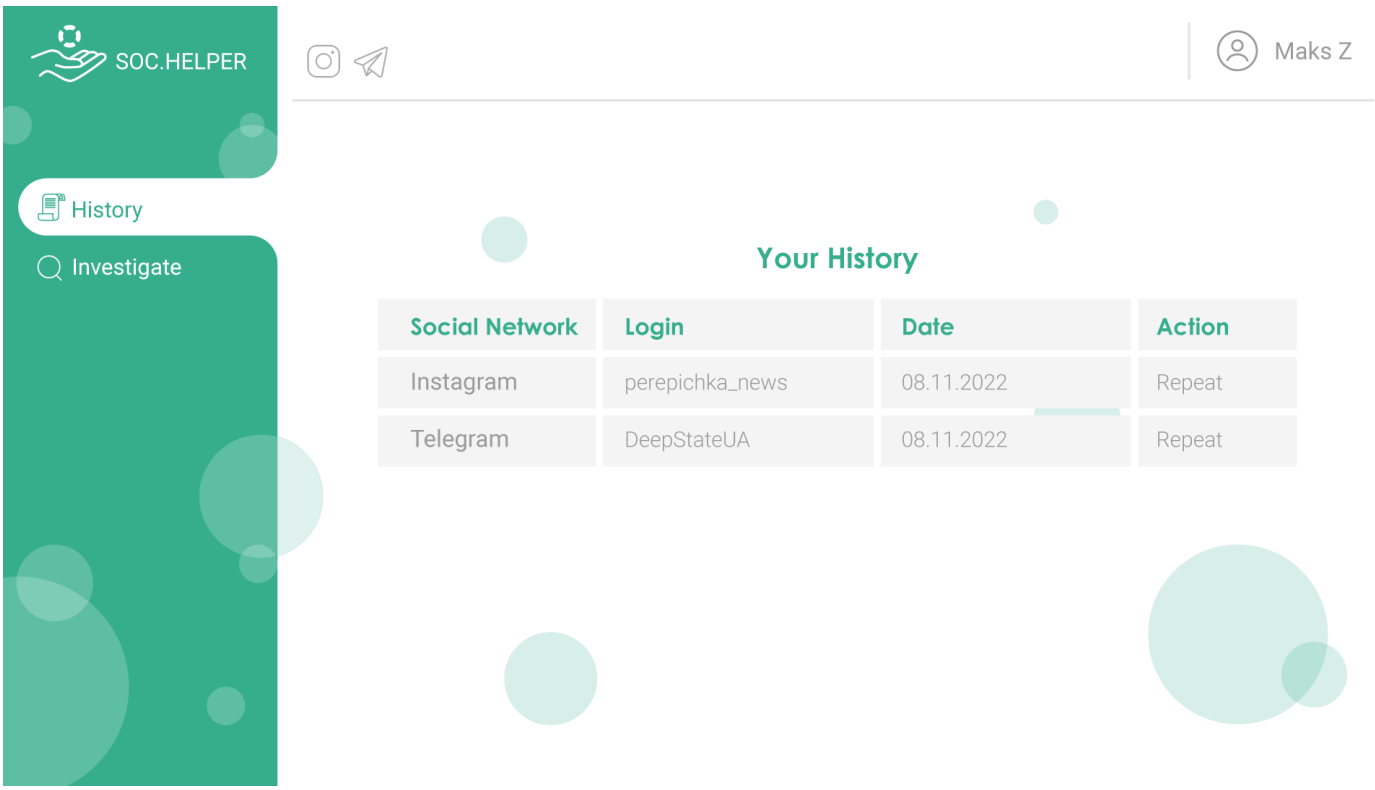


Рис.3.23 Сторінка сторії пошукових запитів

Сторінка досліджень СМ

Сторінка досліджень СМ надає нам можливість скористатись функціоналом дослідження каналу необхідної нам СМ. Дослідження можливо здійнити як зареєстрованим користувачам, так і ні.

Процес дослідження СМ користувача:

- Користувач заповнює дані у форму, а саме:
    1. назва СМ;
    2. логін для СМ.
- Натискає кнопку для підтвердження дослідження і отримує статистичну інформацію по необхідному акаунту із заповнених користувачем даних про СМ.

Дані які ми отримуємо після проведення дослідження:

- Назва акаунту;
- Геолокація, де канал зареєстровано;
- Мову, якою користується акаунт;
- Тематика, що була обрана для акаунту;
- Вік існування каналу;
- Кількість викладених за весь час постів;
- Загальна кількість користувчів, що підписались;
- Кількість користувчів, що відписались за сьогодні;
- Кількість користувчів, що відписались за тиждень;
- Кількість користувчів, що відписались за місяць;
- Показник авторитетності акаунту або індекс цитування, що розраховується на основі згадок каналу в будь якій формі;
- Середній охват аудиторії для однієї публікації;
- Загальний коефіцієнт залученості або Engagement Rate;
- Добовий коефіцієнт залученості або Engagement Rate24;
- Загальний середній охват реклами;
- Середнє рекламне охоплення однієї публікації за 12 годин;
- Середнє рекламне охоплення однієї публікації за 24 годин;
- Середнє рекламне охоплення однієї публікації за 48 годин.

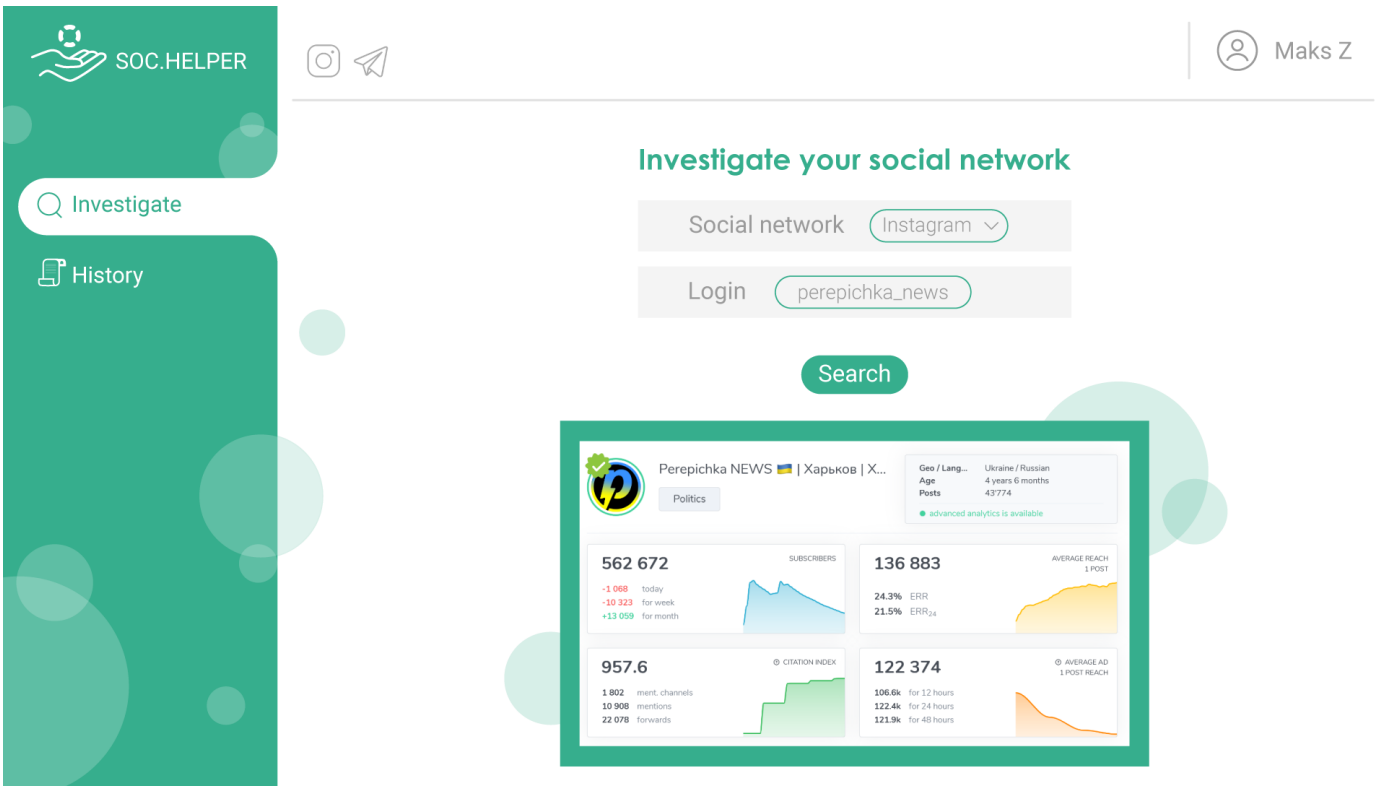


Рис.3.24 Сторінка дослідження Instagram з акаунту

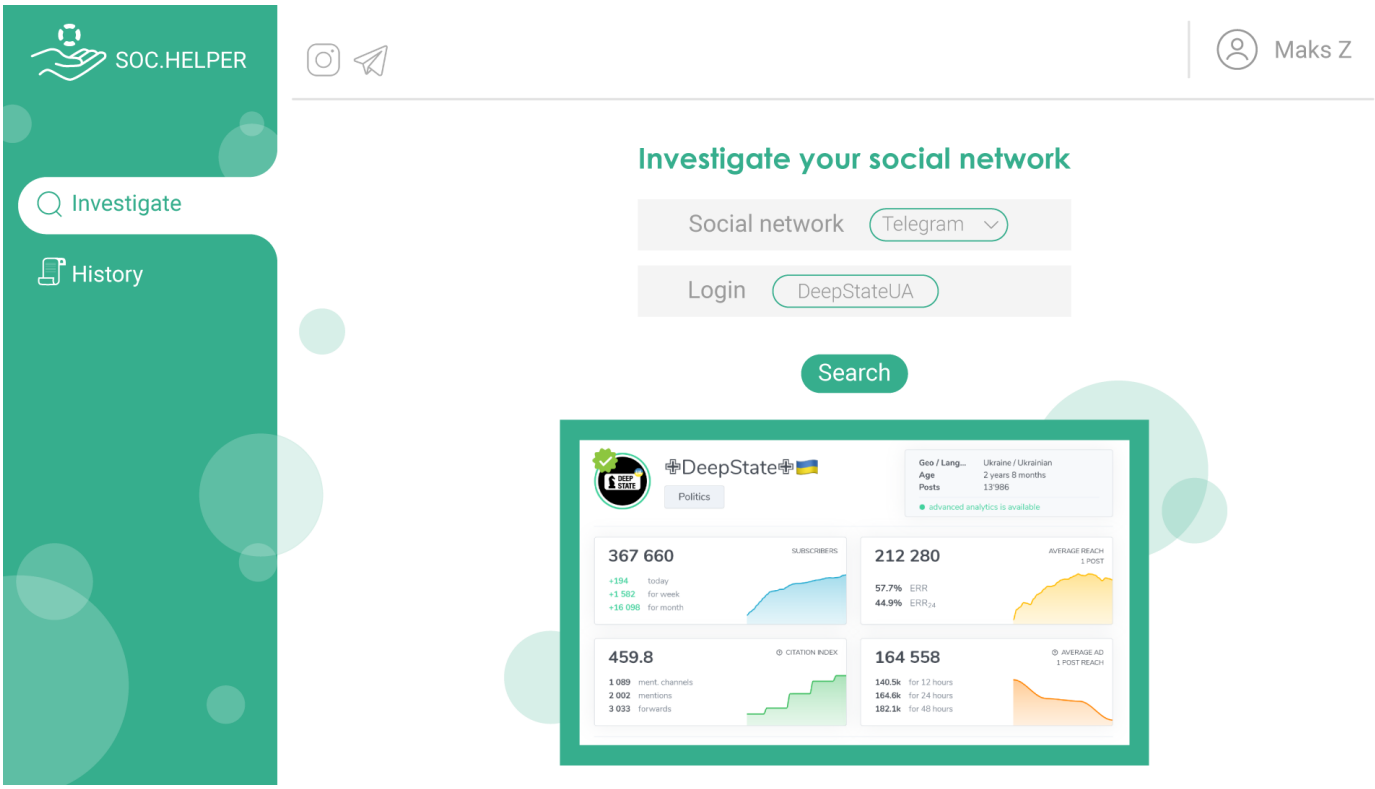


Рис.3.25 Сторінка дослідження Telegram з акаунту

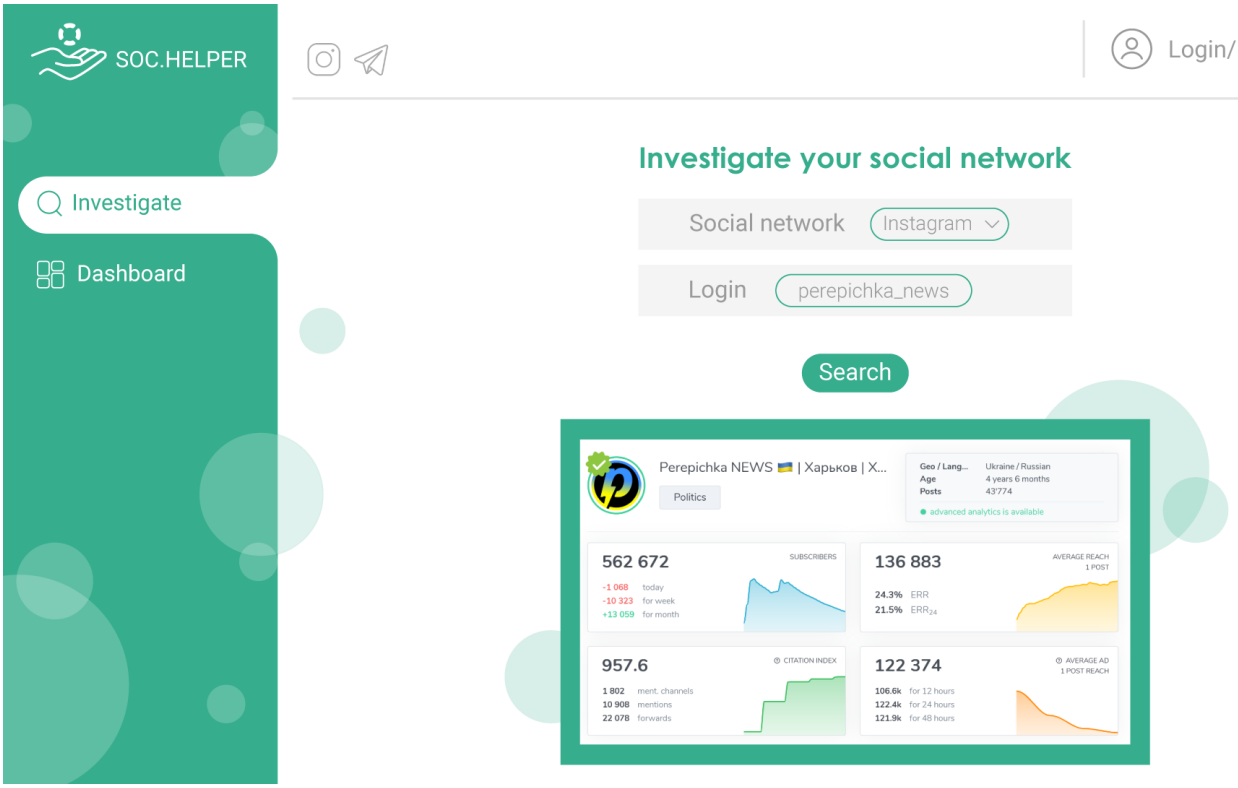


Рис.3.26 Сторінка дослідження Instagram без реєстрації

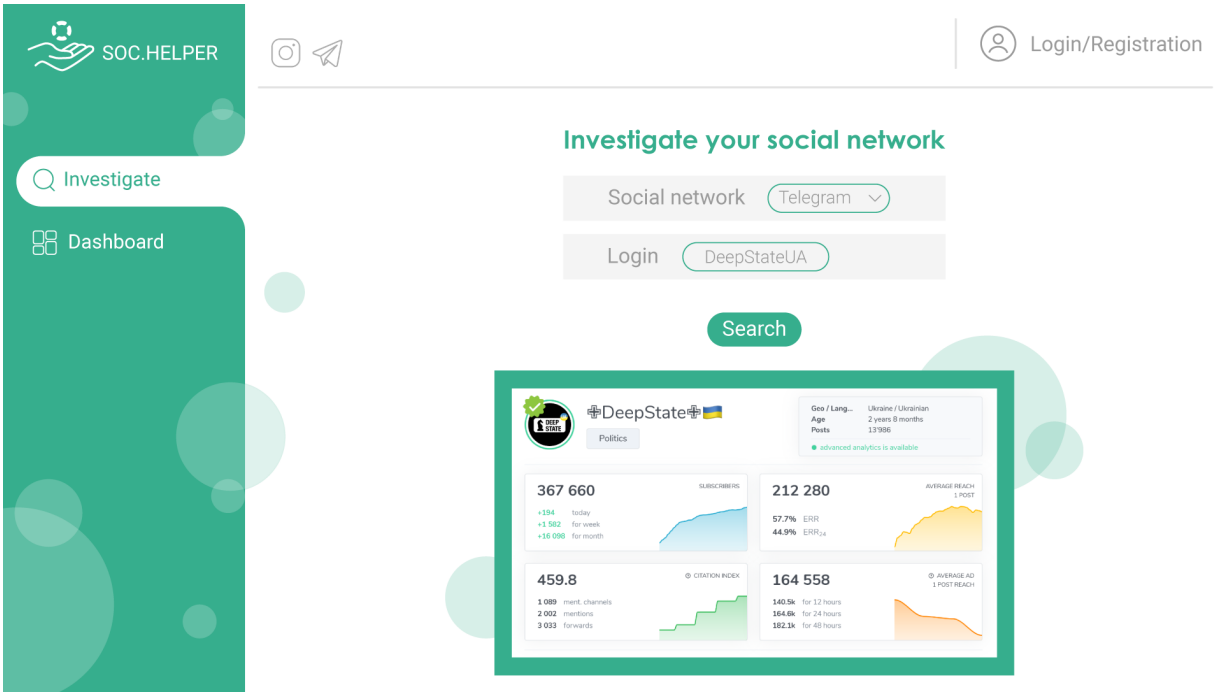


Рис.3.27 Сторінка дослідження Telegram без реєстрації

Висновок до розділу

Цей розділ був присвячений аналіз класів, що використовуються в даному ПЗ. Також було докладно представлено і розглянуто інтерфейс програми, продемонстровано всі його можливості. Аналізуючи розроблене ПЗ ми ознайомились з функціоналом даної системи. Після опису цього розділу було виявлено, що ПЗ повністю дотримує вимоги, що були описані раніше.

ВИСНОВКИ

У даній кваліфікаційній роботі булио детально розглянуто проблеми недостатньої кількості додатків, які могли б надати допомогу у розвитку і покращенні ведення СМ і їх аналізу.

У процесі проектування ПЗ було виконано такі задачі:

1. Аналіз існуючих аналізаторів соціальних мереж.
2. Визначення вимог до ПЗ.
3. Розроблено ПЗ.

Щоб вирішити виявлені аналізом проблеми, було вирішено реалізувати додаток дослідження сторінок СМ.

Було проведено аналіз предметної області та розглянуто існуючі додатки.

В межах кваліфікаційної роботи було розроблено - «Додаток дослідження сторінок соціальних мереж».

Додаток має можливість вдосконалюватися й розширюватись у майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **CreativeSmm** [Електронний ресурс] – Режим доступу до ресурсу: https://creativesmm.com.ua/iak-vesty-storinku-u-sotsmerezhakh-iaku-chytaiut/
2. **Статистика в Інстаграмі: як подивитися статистику в Instagram? - Wezom** [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/statistika-instagram
3. **Встроенная статистика для каналов Telegram: 10 графиков** [Електронний ресурс] – Режим доступу до ресурсу: https://onlypult.com/ru/blog/vstroennaya-statistika-telegram
4. **The Rise Of Online Learning** [Електронний ресурс] – Режим доступу до ресурсу: https://www.forbes.com/sites/ilkerkoksal/2020/05/02/the-rise-of-online-learning/?sh=79c9208472f3
5. ASP.NET Core Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialsteacher.com/core/aspnet-core-introduction
6. Entity Framework Core Documentation And Tutorials | Learn Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://www.learnentityframeworkcore.com/