Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ СОЦІАЛЬНОЇ МЕРЕЖІ «SOCIAL CLUB»

Виконав: студент 4 курсу групи 2КН-20Б
спеціальності 122 Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Лесюк М.М.
(прізвище та ініціали)

Керівник: к.т.н., доцент
Озеранський В.С.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: Барабан М. В.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

ницький національний технічний університет
ультет інтелектуальних інформаційних технологій та автоматизації
едра комп'ютерних наук
нь вищої освіти перший (бакалаврський)
узь знань – 12 Інформаційні технології
ціальність – 122 Комп'ютерні науки
ітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«07» 03 2024 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Лесюку Михайлу Михайловичу

ема роботи: Програмний модуль соціальної мережі "Social Club".

ерівник роботи: Озеранський Володимир Сергійович, к.т.н., доцент,
затверджені наказом вищого навчального закладу "11" 03 2024 року № 80

трок подання студентом роботи 27 05 2024 р

ихідні дані до роботи :
база даних типу MySQL;
кількість колекцій в базі даних - не менше 2 шт;
кількість полів для використання в колекції - не менше 3шт;
Мова програмування – об'єктно-орієнтована з можливістю маніпулювання ими і управління базами даних, сервер яких підтримує HTTPS протокол.

Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно робити):
- Аналіз підходів до класифікації та характеристики соціальних мереж.
- Сучасні засоби реалізації програмних модулів.
- Аналіз відомих програмних рішень для реалізації веб-інтерфейсу
- Розробку структури програмного модуля соціальної мережі "Social Club".
- Розробку алгоритму функціонування системи, налагодження зв'язків між її модулями, реалізувати актуальну схему бази даних системи.
- Сформулювати об'єкт, предмет, мету та задачі подальшого дослідження

ерелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Принципові схеми функцонування технологій що лежать в основі

соціальної мережі;
Діаграми принципових схем роботи з використаннм патерну Unit of Work;
Загальний вигляд інтерфейсу користувача соціальної мережі.

7. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Озеранський В.С., к.т.н., доцент		

8. Дата видачі завдання 07 03 2024р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви робіт	Термін виконання		Примітка
		початок	закінчення	
1	аналіз предметної області	06.05.2024	08.05.2024	
2	постановка задачі	09.05.2024	12.05.2024	
3	аналіз архітектурних рішень	13.05.2024	20.05.2024	
4	розробка клієнтської та серверної частини	21.05.2024	28.05.2024	
5	тестування	29.05.2024	30.05.2024	
6	аналіз результатів тестування	31.05.2024	01.06.2024	
7	обгрунтування вибору інструментів технічної реалізації	02.06.2024	03.06.2024	
8	розробка інструкції користувача	04.06.2024	4.06.2024	
9	Оформлення БКР до захисту	05.06.2024	06.06.2024	

Студент _____ **Лесюк М. М.**
(підпис) (прізвище та ініціали

Керівник проекту (роботи) _____ **Озеранс**
(підпис) (прізвище та ініціал

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 86 сторінок формату А4, на яких є 14 рисунків, список використаних джерел містить 25 найменувань.

Бакалаврська дипломна робота присвячена створенню веб ресурсу соціальної мережі «Social Club». В даній бакалаврській роботі були розроблені та реалізовані модулі бази даних, інтерфейсу користувача, а також спроектована архітектура додатку, що забезпечує ефективну комунікацію між цими модулями з використанням мов програмування C# для розробки серверної частини додатку та JavaScript, каскадних таблиць стилів CSS та мови розмітки гіпертексту HTML для реалізації клієнтської частини, забезпечуючи інтерактивність та динамічність вебсторінок.

Ключові слова: база даних, соціальна мережа, модуль, архітектура, користувач.

ABSTRACT

The bachelor's degree thesis consists of 86 pages in A4 format, including 14 figures. The reference list contains 25 citations.

The bachelor's degree thesis is dedicated to the development of a web resource for a social network called «Social Club». In this bachelor's thesis, database and user interface modules were developed and implemented, along with the design of an application architecture that facilitates efficient communication between these modules. The programming languages used for the server-side development of the application were C# for the server-side development of the application and JavaScript, CSS for cascading style sheets, and HTML for the client-side implementation, providing interactivity and dynamism to web pages.

Keywords: database, social network, module, architecture, user.

ЗМІСТ

ВСТУП

Актуальність досліджень. Споживачі соціальних мереж зростають вимогливішими щодо зручності та ефективності використання платформи. Вони очікують швидкого та легкого доступу до функцій соціальної мережі, інтуїтивно зрозумілого інтерфейсу та можливості налаштування особистих налаштувань. Тому постійне дослідження та вдосконалення інтерфейсу користувача є необхідним для забезпечення зручного та задовільного досвіду використання соціальної мережі. Завдяки зростанню популярності соціальних мереж, обсяг та складність даних, які зберігаються в базі даних, також збільшуються. Це вимагає постійного вдосконалення бази даних, оптимізації запитів та забезпечення швидкого доступу до інформації. Ефективна робота з базою даних дозволить забезпечити високу продуктивність соціальної мережі та задоволення користувачів. Технологічний прогрес швидко рухається вперед, і нові інструменти та методики з'являються на ринку. Використання останніх досягнень в галузі інтерфейсу користувача та бази даних може значно поліпшити функціональність соціальної мережі та забезпечити конкурентні переваги. Користувачі активно взаємодіють з соціальними мережами, залишають коментарі, публікують власні матеріали, взаємодіють з іншими користувачами тощо. Врахування та аналіз цих взаємодій може допомогти вдосконалити інтерфейс користувача та забезпечити зручнішу і більш персоналізовану взаємодію між користувачами. З урахуванням постійного зростання популярності соціальних мереж і вимог користувачів, дослідження та вдосконалення інтерфейсу користувача та бази даних соціальної мережі "Social Club" є актуальним завданням. Забезпечення зручного, ефективного та задовільного досвіду використання соціальної мережі вимагає постійного вдосконалення технологій, оптимізації бази даних та використання новітніх розробок у галузі інтерфейсу користувача. Дослідження у цій області може

принести значну користь як для користувачів соціальної мережі " Social Club ", так і для розвитку соціальних мереж в цілому.

Метою дослідження є розширення функціональних можливостей соціальної мережі за допомогою розробки програмного модуля інтерфейсу користувача і база даних, що дозволить користувачам ефективно здійснювати обмін інформацією.

Об'єктом дослідження є процеси функціонування інтерфейсу користувача і бази даних соціальної мережі « Social Club ».

Предметом дослідження є алгоритми та програмне забезпечення модуля інтерфейсу користувача і база даних соціальної мережі «Social Club». **Задачі дослідження:**

1. Аналіз існуючих рішень.
2. Обґрунтування методу розв'язання задачі.
3. Проєктування програмних модулів бази даних та інтерфейсу користувача соціальної мережі «Social Club».
4. Програмна реалізація модулів бази даних та інтерфейсу користувача соціальної мережі «Social Club».
5. Тестування програмних модулів бази даних та інтерфейсу користувача соціальної мережі «Social Club».
6. Розробка інструкції користувача.

Апробація роботи.

Результати роботи були апробовані на LII науково-технічній конференції підрозділів ВНТУ (НТКП ВНТУ-2023) [1].

Публікації: електронні тези доповідей: «Інструменти розробки інтерфейсу користувача та бази даних соціальної мережі» на LII науково-технічній конференції підрозділів ВНТУ (НТКП ВНТУ-2023) [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз підходів до класифікації та характеристики соціальних мереж

Соціальні мережі стали необхідними інструментами для спілкування та обміну інформацією в онлайн-середовищі. Вони вирішують декілька важливих завдань:

− По-перше, соціальні мережі забезпечують зручний спосіб спілкування та взаємодії між користувачами, незалежно від їхнього географічного розташування. Люди можуть легко знаходити друзів, родичів, колег і спільноти, що дозволяє їм підтримувати соціальні зв’язки та обмінюватися інформацією.

– По-друге, соціальні мережі надають платформу для обміну різноманітним вмістом, таким як фотографії, відео, повідомлення, статуси тощо. Це дозволяє користувачам ділитися своїми думками, ідеями, враженнями та знаннями з іншими.

- По-третє, соціальні мережі дають можливість знайти нових друзів, колег, партнерів і співробітників. Вони сприяють спільноті спільних інтересів і допомагають вам знайти людей, які поділяють ваші цінності та цілі. Статистика показує вагу та вплив соціальних мереж у повсякденному житті людей. Наприклад, статистика показує, що кількість користувачів соціальних мереж значно зросла. Наприклад, станом на вересень 2021 року Facebook має понад 2,8 мільярда активних користувачів щомісяця.

Коефіцієнт впливу (Ci) — це показник, який вимірює активність користувачів і взаємодію з опублікованим контентом у соціальних мережах.

Формула для розрахунку фактора впливу така:

$$Ci = (Cl + Cc + Cr) / Cp * 100,$$

де *Ci* – це коефіцієнт впливу; *Cl* – це коефіцієнт вподобань; *Cc* – це коефіцієнт коментарів; *Cr* – це коефіцієнт репостів; *Cp* – це коефіцієнт публікацій.

Ця метрика показує рівень активності глядачів і взаємодії з контентом.

Середній час, який користувачі проводять у соціальних мережах, є важливим показником популярності та впливу цих платформ. Дані показують, що середня тривалість сеансів соціальних мереж у 2020 році становила приблизно 2 години на день [2].

Соціальні мережі мають значний вплив на економіку через рекламу, маркетинг, впливають на купівлю товарів і послуг. За оцінками, світові витрати на рекламу в соціальних мережах у 2020 році перевищили 84 мільярди доларів. Ці статистичні дані та формули забезпечують кількісну оцінку та вимірювання впливу соціальних мереж на суспільство, вони доповнюють аналіз і показують такі важливі аспекти, як популярність соціальних мереж, взаємодія та соціальний вплив.

Дослідження соціальних мереж доводять, що вони є широко поширеними та мають значний вплив на сучасне суспільство.

Класифікація соціальних мереж за призначенням, типом зв’язку, фокусом і типом взаємодії може допомогти краще зрозуміти їх різноманітність і характеристики.

Одна з основних класифікацій базується на призначенні соціальних мереж.

Професійні мережі, такі як LinkedIn, призначені для об’єднання професіоналів і надання можливості публічно ділитися своєю професійною інформацією. Такі популярні соціальні мережі, як Facebook і Twitter, створюють платформи для спілкування та обміну інформацією в різних сферах життя.

Інша класифікація заснована на типі підключення до соціальної мережі.

Горизонтальні соціальні мережі засновані на спільних інтересах і об'єднують користувачів з однаковими захопленнями та діяльністю. З іншого боку, вертикальні соціальні мережі створюють взаємодію між користувачами на різних ієрархічних рівнях, таких як студенти та викладачі університетів або професійні спільноти.

Соціальні мережі також можна класифікувати за їх спрямованістю.

Соціальні мережі зв’язку, такі як WhatsApp і Viber, дозволяють користувачам спілкуватися та обмінюватися повідомленнями.

З іншого боку, соціальні контент-мережі, такі як YouTube і Instagram [3], спрямовані на спільне створення та обмін контентом, таким як фотографії та відео.

Не менш важлива класифікація соціальних мереж залежить від форми взаємодії. Мережі мікроблогів, такі як Twitter, дозволяють користувачам публікувати короткі повідомлення та оновлення статусу. Фото та відеомережі, такі як Instagram і TikTok, дозволяють легко ділитися та розповсюджувати фотографії та відео з іншими.

Здобуті в ході дослідження знання про соціальні мережі та їхню класифікацію мають важливе значення для подальшого розвитку цих платформ. Вони допомагають розробникам створювати нові інноваційні рішення та покращувати вже існуючі соціальні мережі. Крім того, користувачам вони надають можливість обирати підходящі платформи, що відповідають їхнім потребам і інтересам у спілкуванні та обміні інформацією з оточуючим світом. Зрозуміння різноманітності соціальних мереж і їхніх характеристик сприяє покращенню якості взаємодії та спілкування в цифровому середовищі.

Соціальна мережа «Social Club» відповідає декільком категоріям класифікації соціальних мереж, які були наведені вище. За форматом взаємодії, « Social Club» можна віднести до фото- та відео-мереж, оскільки її основна функція полягає у спільному

створенні, обміні та поширенні фотографій та відео. Також, з точки зору призначення, «Social Club» може вважатись соціальною мережею загального призначення, оскільки вона надає можливість користувачам спілкуватись та обмінюватись інформацією в різних сферах життя.

1.2 Огляд існуючих соціальних мереж

В період швидкого розвитку технологій передачі даних через мережу Інтернет, для розробників виникають безліч можливостей створення проектів з

різними видами архітектур. Користувач повинен отримувати максимальний відгук від додатків і не чекати на довгі затримки під час обміну інформації між компонентами додатка.

X (Twitter) - Twitter відомий зараз як X, був створений у 2006 році і швидко став популярним завдяки своїй платформі для мікроблогів, де користувачі можуть публікувати короткі повідомлення (твіти) довжиною до 280 символів. Платформа використовується для обміну новинами, думками, фотографіями, відео та лінками. Вона стала важливим інструментом для миттєвого поширення інформації та взаємодії з великою аудиторією завдяки своєму підходу до розміщення інформації на ресурсі.

Розглянемо технічні особливості реалізації платформи:

- Архітектура та технології:
- Мова програмування: Scala, Java, Ruby, C++
- Фреймворки: Finagle
- Бази даних: MySQL, Manhattan, Redis
- Кешування: Memcached, Redis
- Сервіси: Мікросервісна архітектура
- Хостинг: Власні центри обробки даних, AWS
- Безпека:
- Шифрування: HTTPS, шифрування даних у стані спокою
- Аутентифікація: Двофакторна аутентифікація (2FA)
- Моніторинг: Виявлення та запобігання зловмисним діям

TikTok – соціальна мережа для створення та перегляду коротких відео, яка була запущена у 2016 році компанією ByteDance. Платформа стала надзвичайно популярною серед молоді завдяки своїм можливостям для створення відео з

використанням музики, ефектів та фільтрів. TikTok відомий своїми вірусними трендами та швидким поширенням контенту.

Розглянемо технічні особливості реалізації платформи:

- Архітектура та технології:
- Мова програмування: Java, Python, Go
- Фреймворки: Flask, власні розробки
- Бази даних: MySQL, Redis, Apache Cassandra
- Кешування: Redis
- Сервіси: Мікросервісна архітектура
- Хостинг: AWS, Google Cloud Platform (GCP)
- Безпека:
- Шифрування: HTTPS, шифрування даних у стані спокою
- Аутентифікація: Двофакторна аутентифікація (2FA)
- Приватність: Контроль за налаштуваннями приватності, моніторинг відповідності стандартам конфіденційності

Facebook – продукт, створений у 2004 році Марком Цукербергом, став однією з найбільших соціальних мереж у світі. Платформа дозволяє користувачам спілкуватися, обмінюватися фотографіями та відео, створювати групи за інтересами, а також використовувати додаткові функції, такі як Marketplace для купівлі та продажу товарів.

Розглянемо технічні особливості реалізації платформи:

- Архітектура та технології:
- Мова програмування: PHP (Hack), JavaScript, C++, Python
- Фреймворки: React, HHVM
- Бази даних: MySQL, HBase, RocksDB
- Кешування: Memcached, TAO

- Сервіси: Мікросервісна архітектура
- Хостинг: Власні центри обробки даних
- Безпека:
- Шифрування: HTTPS, шифрування даних у стані спокою
- Аутентифікація: Двофакторна аутентифікація (2FA)
- Приватність: Детальні налаштування приватності, відповідність GDPR для користувачів з ЄС

1.3 Аналіз переваг та недоліків існуючих соціальних мереж

Соціальні мережі стали невід'ємною частиною нашого повсякденного життя, надаючи платформу для спілкування, обміну інформацією та творчого самовираження. Кожна з основних соціальних мереж має свої унікальні особливості, які приваблюють різні групи користувачів. Ось загальний аналіз переваг та недоліків основних соціальних мереж, таких як Twitter (X), TikTok, Facebook[2] та інших:

Переваги соціальних мереж:

1 Масштабована аудиторія:

- Соціальні мережі забезпечують доступ до великої аудиторії по всьому світу. Вони дозволяють користувачам взаємодіяти з іншими людьми незалежно від географічного розташування.

2 Швидке поширення інформації:

- Інформація може розповсюджуватися миттєво, що особливо корисно для новин, маркетингових кампаній та громадських ініціатив.

3 Творчі можливості:

- Платформи, такі як TikTok та Instagram, надають інструменти для створення і редагування контенту, що стимулює творчість та самовираження.

4 Підтримка спільнот:

- Соціальні мережі дозволяють створювати спільноти за інтересами, де люди можуть обмінюватися досвідом, знаннями та підтримувати один одного.

5 Аналітичні інструменти:

- Платформи надають потужні інструменти аналітики, які допомагають бізнесам та контент-мейкерам краще розуміти свою аудиторію та оптимізувати свою діяльність.

6 Можливості для бізнесу:

- Соціальні мережі надають платформи для реклами, просування брендів та продажу товарів і послуг. Facebook Marketplace, Instagram Shopping та інші сервіси дозволяють бізнесам безпосередньо взаємодіяти зі споживачами.

Недоліки соціальних мереж:

1 Проблеми з приватністю:

- Соціальні мережі часто стикаються з критикою щодо захисту даних користувачів. Скандали, пов’язані з витоками даних та неправомірним використанням інформації, є серйозними проблемами.

2 Поширення дезінформації:

- Платформи можуть бути використані для поширення фейкових новин та дезінформації, що може мати негативний вплив на суспільство.

3 Кібербулінг та тролінг:

- Відсутність ефективної модерації може призводити до поширення образливого контенту, кібербулінгу та тролінгу.

4 Залежність та вплив на психічне здоров'я:

- Надмірне використання соціальних мереж може призводити до залежності та негативно впливати на психічне здоров'я користувачів, включаючи тривожність, депресію та проблеми з самооцінкою.

5 Рекламна надмірність:

- Велика кількість реклами може дратувати користувачів і знижувати якість користувацького досвіду.

6 Зловживання та шахрайство:

- Соціальні мережі можуть бути використані для шахрайства, зловживань та інших незаконних дій, таких як фішинг та інші види кіберзлочинності.

Отже, під час створення соціальної мережі потрібно ретельно зважувати всі ризики, передбачувати створення правил спільноти та важелів впливу на недобросовіних користувачів.

1.4 Концепція створення соціальної мережі

Створення соціальної мережі є складним і багатогранним процесом, що вимагає детального планування та стратегічного підходу. Першим кроком у розробці соціальної мережі є визначення її цільової аудиторії та місії. Важливо зрозуміти, ком будуть користувачі платформи, які їхні інтереси та потреби. Визначення місії мережі допоможе сформулювати її основну мету, яка може варіюватися від професійного нетворкінгу до обміну творчим контентом чи підтримки спільнот за інтересами.

Наступним етапом є планування архітектури та технічної реалізації. Вибір технологічного стеку є критично важливим, оскільки від цього залежить продуктивність, масштабованість та зручність підтримки соціальної мережі. Можна розглянути використання популярних мов програмування та фреймворків, таких як

React або Angular для фронтенду та Node.js або .Net для бекенду. Крім того, вибір відповідної бази даних (наприклад, MySQL або MongoDB) та серверної інфраструктури (включаючи хмарні сервіси як AWS, Google Cloud чи Azure) впливатиме на загальну ефективність роботи мережі.

Реалізація функціональних можливостей є наступним ключовим етапом. Основні функції включають реєстрацію та аутентифікацію користувачів, створення та редагування профілів, можливість додавання друзів та підписок, а також взаємодію з контентом через пости, коментарі, чати та групи.

Безпека та приватність є критично важливими аспектами соціальної мережі. Забезпечення безпеки даних користувачів через шифрування (HTTPS для передачі даних та шифрування даних у стані спокою) є обов'язковим. Двофакторна аутентифікація (2FA) додасть додатковий рівень безпеки.[3]

Управління та розвиток платформи є постійним процесом. Можливість підтримки користувачів для вирішення технічних проблем та збору зворотного зв'язку є критично важливим фактором. Постійні оновлення, які включають нові функції, покращення безпеки та виправлення помилок, є необхідними для підтримки зацікавленості користувачів та забезпечення стабільної роботи мережі. Використання аналітики для прийняття обґрунтованих рішень щодо розвитку платформи допоможе краще розуміти потреби користувачів та оптимізувати функціональність мережі.

Отже, створення успішної соціальної мережі вимагає комплексного підходу, що включає глибоке розуміння потреб цільової аудиторії, вибір відповідної технологічної архітектури, реалізацію функціональних можливостей, забезпечення безпеки та приватності.

1.5 Сучасні засоби реалізації програмних модулів

У даному розділі ми детальніше розглянемо різні варіанти реалізації програмного модуля «База даних» для нашої соціальної мережі «Social Club». Порівняємо реляційні бази даних (РБД) та нереляційні бази даних (NoSQL), а також розкриємо деякі цікаві особливості та застосування кожного типу.

Розрізнення між різними варіантами структур баз даних в SQL та NoSQL полягає в організації та зберіганні даних. Давайте розглянемо кожен тип окремо [4].

Бази даних типу SQL мають два основних підтипи: реляційні та нереляційні бази даних. Реляційні бази даних є найпоширенішим типом баз даних, вони зберігають дані у вигляді таблиць, де кожен рядок представляє запис, а кожний стовпчик - поле або атрибут (RDBMS). Реляційна модель дозволяє встановлювати взаємозв'язки між таблицями за допомогою ключів, що дозволяє виконувати складні запити та забезпечує цілісність даних. Також є можливість оптимізовувати CRUD операції з цим типом баз даних за допомогою додавання індексів, контролю рівнів ізоляції транзакцій до бази даних. З іншого боку, нереляційні бази даних спеціалізуються на обробці великих обсягів даних та їх аналізі. Вони оптимізовані для швидкого виконання агрегатних запитів та операцій OLAP (Online Analytical Processing), що дозволяє ефективно працювати з великомасштабними аналітичними завданнями. Самі ж бази даних зазвичай зберігають дані в оптимізованому текстовому форматі (наприклад JSON), який не має чіткої структури чи явних взаємозв’язків між атрибутами.

Бази даних типу NoSQL пропонують більш гнучкі моделі зберігання та організації даних. Вони включають такі типи баз даних:

1. Документ-орієнтовані. Наприклад, MongoDB, Couchbase, та деякі інші. Вони зберігають дані у вигляді JSON-подібних документів.

2. Ключ-значення. Наприклад, Redis, Riak, і Amazon DynamoDB. Дані зберігаються у вигляді пар ключ-значення, де ключ унікальний і використовується для швидкого доступу до значення.
3. Стовпчасті. Наприклад, Apache Cassandra та HBase. Дані зберігаються у вигляді стовпців з допомогою рядків ключів.
4. Графові. Наприклад, Neo4j, Amazon Neptune. Вони використовують графову модель для зберігання даних та зв'язків між ними.

Приклади наведених вище структур наведені на рисунку 1.1.

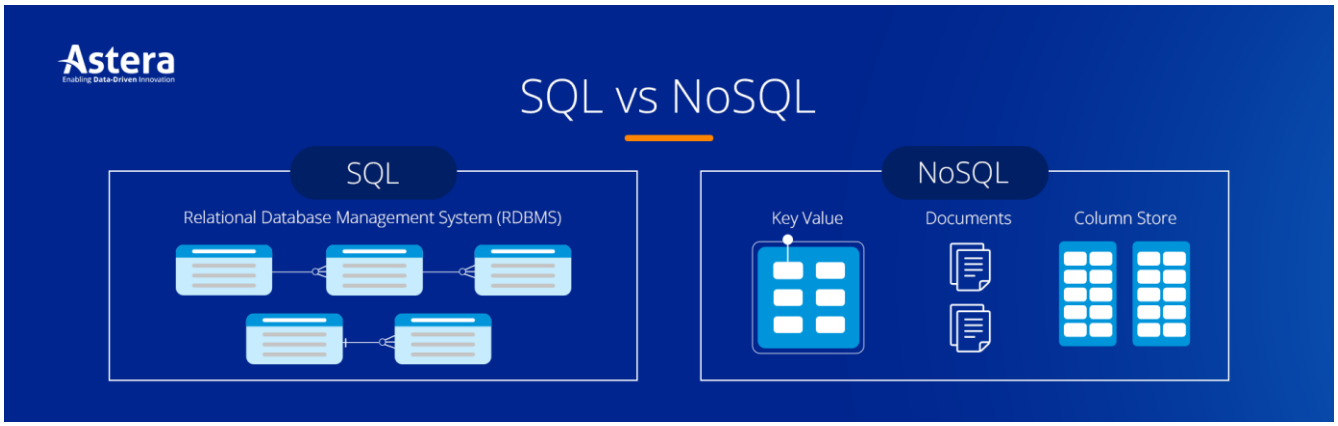


Рисунок 1.1 – Структурні схеми баз даних типу SQL та NoSQL

Також, як реляційні, так і не реляційні бази даних, мають певні переваги та недоліки.

Реляційні бази даних (РБД) використовують таблиці для організації та зберігання даних. Наприклад, MySQL та PostgreSQL є популярними РБД, які забезпечують структурованість даних та гарантують цілісність. Вони надають можливість виконувати складні операції над даними, такі як з'єднання таблиць, фільтрація та сортування, що важливо для ефективного управління даними у соціальній мережі. Реляційні бази даних також забезпечують можливість

використовувати мову структурованих запитів SQL, що спрощує роботу з даними та дозволяє виконувати складні запити.

Нереляційні бази даних (NoSQL) використовуються для зберігання та обробки великих обсягів даних з високою швидкодією. Наприклад, MongoDB та Cassandra є популярними NoSQL базами даних. Вони забезпечують гнучкість схеми даних, дозволяючи змінювати її гнучко без необхідності відповідного оновлення бази даних. Також вони мають високу горизонтальну масштабованість, що дозволяє розподіляти дані на кластери серверів для ефективної обробки великих обсягів даних. Нереляційні бази даних є особливо корисними у випадках, коли необхідно зберігати невпорядковані або змінюючіся дані, наприклад, при роботі з потоковими даними або інформацією з сенсорів.

Порівнюючи обидва варіанти, ми бачимо, що реляційні бази даних надають структурованість, цілісність та широкий спектр операцій, що є необхідним для успішної реалізації бази даних додатка. MySQL або PostgreSQL можуть бути відмінними виборами для наших потреб у зберіганні та управлінні даними. Вони також забезпечують надійність та довіреність, що є важливими аспектами для соціальної мережі. Це можна наглядніше побачити в таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця SQL та NoSQL баз даних

Перевага	SQL	NoSQL
Структурованість даних	+	-
Гарантія цілісності	+	-
Складні запити	+	-
Гнучка схема даних	+	-
Горизонтальна масштабованість	-	+
Прості запити	-	+

Висока швидкодія	-	+
Широка спільнота користувачів	-	+

Проте, враховуючи розширення соціальної мережі «Social Club» та потенціал зростання обсягів даних, ми можемо розглянути використання нереляційних баз даних, таких як MongoDB або Cassandra. Їх горизонтальна масштабованість дозволить ефективно працювати з великими обсягами інформації та забезпечить високу швидкодію обробки запитів. Це особливо важливо, коли у соціальній мережі наявні мільйони користувачів із значною активністю.

Також, варто розглянути різні системи керування базами даних (СКБД) і порівняти їх переваги та недоліки для розробки соціальної мережі. У таблиці 1.2 представлені деякі популярні СКБД та їх характеристики [5].

Таблиця 1.2 – Порівняльна таблиця СКБД

СКБД	Висока надійність	Розширені можливості управління даними	Підтримка широкого спектру інструментів розробника	Зручне адміністрування бази даних	Високі вимоги до ресурсів сервера	Вартість ліцензування	Обмежена підтримка розподіленої обробки даних
Microsoft SQL Server	+	+	+	+	+	+	-
MySQL	+	-	-	+	-	+	
PostgreSQL	+	+	-	-	-	-	+
Oracle	+	+	+	+	+	-	+
MongoDB	-	-	-	-	+	+	+

Отже, виходячи з інформації, що міститься у таблиці 1.2, можемо зробити такі висновки:

− Microsoft SQL Server має значні переваги для розробки соціальної мережі «Social Club». Він має високу надійність, розширені можливості управління даними, підтримку широкого спектру розробних інструментів, зручне адміністрування бази даних та обмежену підтримку розподіленої обробки даних [6].

− Система MySQL підходить для проектів з слабкішими

вимогами до управління даними та обмеженими ресурсами сервера, але він не має розширених можливостей управління даними та обмежується підтримкою широкого спектру розробних інструментів.

− Система PostgreSQL має високу надійність, але обмежена у розширених можливостях управління даними та не підтримує широкий спектр розробних інструментів. Він також має обмежену підтримку розподіленої обробки даних;

− Система Oracle має всі переваги, що необхідні для розробки соціальної мережі «Social Club», включаючи високу надійність, розширені можливості управління даними, підтримку широкого спектру розробних інструментів та обмежену підтримку розподіленої обробки даних. Проте, вартість ліцензування Oracle може бути вищою в порівнянні з іншими СКБД;

− Система MongoDB, хоча не має багатьох переваг для розробки соціальної мережі «Social Club», він має високі вимоги до ресурсів сервера, зручне адміністрування бази даних та підтримку розподіленої обробки даних. Втім, його висока надійність та розширені можливості управління даними відсутні, що може ускладнити розробку соціальної мережі [7].

Після порівняння цих варіантів, було вирішено обрати реляційну базу даних Microsoft SQL Server для нашого модуля «База даних» у соціальній мережі «Social Club». Реляційні бази даних надають структурованість, цілісність та широкий спектр операцій, що є необхідним для успішної реалізації нашого модуля.

Таким чином, використання реляційної бази даних в нашому програмному модулі забезпечить надійну та ефективну роботу нашої соціальної мережі «Social Club».

1.6 Аналіз відомих програмних рішень для реалізації веб-інтерфейсу

У цьому розділі буде проведено аналіз декількох програмних рішень, які можна використовувати для створення веб-інтерфейсу. Розглянемо три популярні фреймворки: AngularJS, Vue.js та ReactJS, і визначимо, який обрати для реалізації графічного інтерфейсу платформи[11].

Почнемо з AngularJS. Цей фреймворк відомий своєю здатністю створювати динамічні односторінкові додатки. Завдяки своїй масштабованості, AngularJS дозволяє легко розширювати проект та додавати нові функції. Додатковою перевагою є велика спільнота розробників, яка забезпечує наявність різноманітних ресурсів і підтримку. Проте, для нових розробників AngularJS може бути трохи складним у вивченні, через його комплексний підхід до створення веб-додатків.

Наступним фреймворком для аналізу є Vue.js. Його особливістю є простота використання та швидкодія. Порівняно легкий API та проста структура роблять Vue.js привабливим для швидкого початку розробки. Крім того, Vue.js може легко інтегруватись з існуючими проектами, що забезпечує його гнучкість. Варто відзначити, що у порівнянні з ReactJS, Vue.js може мати меншу спільноту

розробників, обмежену екосистему розширень та застарілі технічні рішення, що можуть бути не актуальними для сучасних завдань.

Також розглянемо ReactJS, який є одним з найпопулярніших фреймворків у сфері веб-розробки. Він відомий своїм компонентним підходом, що спрощує створення перевикористовуваних компонентів та розширення функціональності. ReactJS також славиться високою швидкодією завдяки використанню віртуального DOM та ефективного алгоритму оновлення. Крім того, наявність великої спільноти розробників ReactJS забезпечує наявність багатьох ресурсів, документації та підтримки. Значною перевагою ReactJS є його здатність легко інтегруватись з іншими технологіями і бібліотеками, можливість самостійного компонування частин бібілотеки залежно від бізнес вимог, що дає більшу гнучкість у розробці.

Для зручного порівняльного аналізу наведених вище фреймворків, варто звернутися до таблиці

Таблиця 1.4 – Порівняльна таблиця фреймворків, для реалізації вебінтерфейсу

Переваги	AngularJS	Vue.js	ReactJS
Легкість використання	-	+	+
Швидкодія	+	+	+
Розширюваність	+	+	+
Велика спільнота розробників	+	-	+
Інтеграція з існуючими проектами	-	+	+
Компонентний підхід	-	+	+

Гнучкість у розробці	-	-	+

Після аналізу цих фреймворків можна зробити висновок, що ReactJS є одним з найкращих варіантів для реалізації веб-інтерфейсу у нашому проекті. Його швидкодія, розширюваність, наявність великої спільноти розробників та підтримки роблять його потужним інструментом для створення веб-інтерфейсу, який найкраще підходить у випадку створення соціальної мережі.

1.7 Висновок до розділу 1

Отже, у першому розділі дипломної роботи проведено аналіз предметної області, порівняно різні типи баз даних та інтерфейси користувача для програмного модулю соціальної мережі «Social Club». Виявлено, що реляційні бази даних, зокрема MySQL, мають переваги у зберіганні структурованих даних та забезпеченні консистентності. Веб-інтерфейс було вибрано як оптимальний варіант для забезпечення універсального та зручного доступу до соціальної мережі «Social Club». Рекомендовано використовувати реляційну базу даних (наприклад, MySQL або PostgreSQL) для зберігання структурованих даних і розробити вебінтерфейс, щоб забезпечити зручний доступ для користувачів з різних пристроїв.

2 РОЗРОБКА ТА ПРОЄКТУВАННЯ ПРОГРАМНИХ МОДУЛІВ СОЦІАЛЬНОЇ МЕРЕЖІ

2.1 Розробка структури бази даних соціальної мережі «Social Club»

Розробку структури бази даних варто почати з розробки універсального відношення [7]. В універсальне відношення потрібно включити атрибути, що описують наступні сутності: Chat (Чат), MessageImages (Зображення повідомлень), Statuses (Статуси), User (Користувач), Message (Повідомлення), MessageImage (Зображення повідомлення), Post (Повідомлення), Subscribers (Підписки), UserChat (Чат користувача), Comments (Коментарі), Likes (Лайки) та CommentLikes (Лайки коментарів).

Типи сутностей:

- Chat (Чат) – стрижнева;
- MessageImages (Зображення повідомлень) – характеристична, бо є атрибутом сутності Message (Повідомлення);
- User (Користувач) – стрижнева;
- Message (Повідомлення) – стрижнева;
- Post (Дописи) – стрижнева;
- Subscribers (Підписки) – стрижнева;
- Comments (Коментарі) - характеристична, бо є атрибутом сутності Post (Дописи);
- Likes (Лайки) - характеристична, бо є атрибутом сутності Post (Дописи);

− CommentLikes (Лайки коментарів) - характеристична, бо є атрибутом сутності Comments (Коментарі);

Сутності описуються відповідно до вимог подання категорій бази даних: −

Chat (Чат) (ChatId, CreationDate, LastSeen);

− MessageImages (Зображення повідомлень) (MessageImagesId, Image)

− User (Користувач) (UserId, Email, Name, Dob, CreationDate, LastSeen, PhoneNumber, UserName, Password, ProfilePic, BioProfile, EmailVerified)

− Message (Повідомлення) (MessageId, ChatId, UserId, MessageText, SendDate, StatusId, MessageImagesId);

− Post (Дописи) (PostId, StatusId, Description, PublishDate, Image, UserId, Thumbnail);

− Subscribers (Підписки) (SubscriptionId, FollowerId, FollowingId, FollowDate);

− UserChat (Чат користувача) (UserChatId, ChatId, UserId);

− Comments (Коментарі) (CommentId, UserId, PostId, CommentText, CommentDate, EditingDate, StatusId);

− Likes (Лайки) (UserId, PostId, LikeTime, LikeId);

− CommentLikes (Лайки коментарів) (UserId, CommentId, LikeTime, LikeId).

Отже, універсальне відношення для даної бази даних буде мати наступний вигляд:

R (ChatId, CreationDate, LastSeen, MessageImagesId, Image, UserId, Email, Name, Dob, CreationDate, LastSeen, PhoneNumber, UserName, Password, ProfilePic, BioProfile, EmailVerified, MessageId, ChatId, UserId, MessageText, SendDate, StatusId, MessageImagesId, PostId, StatusId, Description, PublishDate, Image, UserId, Thumbnail, SubscriptionId, FollowerId, FollowingId, FollowDate, UserChatId, ChatId, UserId,

CommentId, UserId, PostId, CommentText, CommentDate, EditingDate, StatusId, UserId, PostId, LikeTime, LikeId, UserId, CommentId, LikeTime, LikeId.).

Ступінь універсального відношення – 50. Наступним кроком буде розробка ER-моделі предметної області «Соціальна мережа».

Характеристику зв'язків предметної області «Соціальна мережа» представлено в таблиці 2.1.

Таблиця 2.1 – Характеристика зв'язків предметної області «Соціальна мережа»

Ім'я сутності 1	Ім'я сутності 2	Тип зв'язку	Ім'я зв'язку	Клас належності
User	Chat	1:Б	Має	Обов'язковий
User	Message	1: Б	Надсилає	Обов'язковий
User	Post	1:Б	Публікує	Обов'язковий
Subscribers	User	Б:1	Слідкує	Обов'язковий
User	Comments	1:Б	Пише	Обов'язковий
User	Likes	1:Б	Ставить	Обов'язковий
User	CommentLikes	1:Б	Ставить	Обов'язковий
Chat	Message	1:Б	Містить	Обов'язковий
Message	MessageImages	1:1	Містить	Необов'язковий
Post	Comments	1:Б	Містить	Необов'язковий
Post	Likes	1:Б	Містить	Необов'язковий
Comments	CommentLikes	1:Б	Містить	Необов'язковий

Представлення ER-моделі для бази даних «Соціальна мережа» зображено на рисунку 2.1

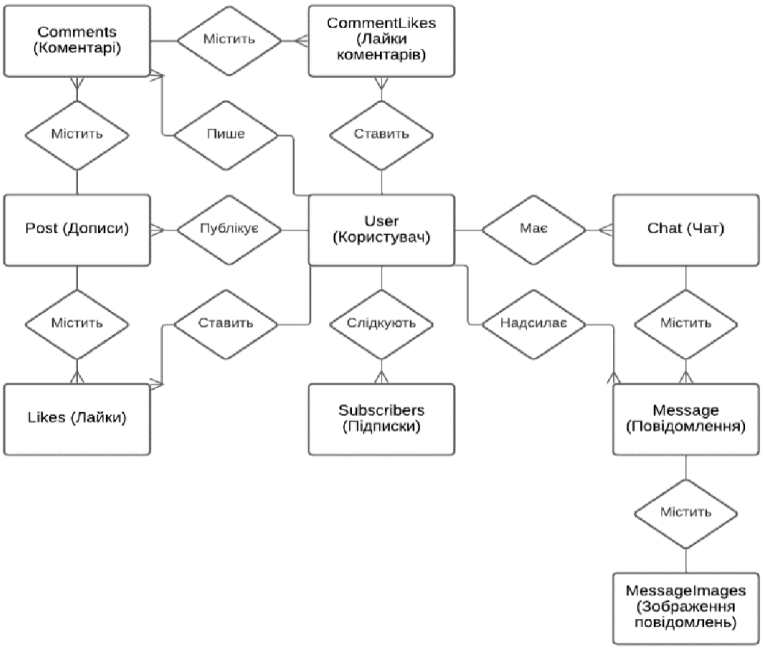


Рисунок 2.1 – Схема ER-моделі бази даних соціальної мережі

Для ідентифікації та забезпечення унікальності даних було створено такі основні ключі:

Основний ключ (PrimaryKey) для кожної сутності:

- Chat: ChatId;
- MessageImages: MessageImagesId;
- User: UserId;
- Message: MessageId;
- Post: PostId;
- Subscribers: SubscriptionId;
- Comments: CommentId;
- Likes: LikeId;
- CommentLikes: CommentLikeId.

Для забезпечення взаємозв'язків між сутностями було використано наступні зв’язки:

- Chat (Чат) зв'язаний з сутністю User (Користувач) через зв'язок UserChat (Чат користувача), де зберігаються дані про учасників чату;

− Chat (Чат) зв'язаний з сутністю Message (Повідомлення) через зв'язок, де кожне повідомлення має вказане відповідний ChatId;

− Message (Повідомлення) зв'язаний з сутністю User (Користувач) через зв’язок UserId, який вказує на автора повідомлення;

− Message (Повідомлення) зв'язаний з сутністю Statuses (Статуси) через зв’язок StatusId, який вказує на статус повідомлення;

− Message (Повідомлення) зв'язаний з сутністю MessageImages (Зображення повідомлень) через зв'язок MessageImagesId, який вказує на зображення повідомлення;

− MessageImages (Зображення повідомлень) зв'язаний з сутністю MessageImage (Зображення повідомлення) через зв'язок MessageImagesId, який вказує на зображення, пов'язане з групою зображень повідомлення;

− User (Користувач) зв'язаний з сутністю Subscribers (Підписки) через зв'язок FollowerId і FollowingId, де зберігаються дані про підписку користувачів;

− User (Користувач) зв'язаний з сутністю Post (Повідомлення) через зв'язок UserId, який вказує на автора поста;

− Comments (Коментарі) зв'язаний з сутністю User (Користувач) через зв'язок UserId, який вказує на автора коментаря;

− Comments (Коментарі) зв'язаний з сутністю Post (Повідомлення) через зв'язок PostId, який вказує на повідомлення, до якого прикріплений коментар;

− Likes (Лайки) зв'язаний з сутністю User (Користувач) через зв'язок UserId, який вказує на користувача, що поставив лайк;

− Likes (Лайки) зв'язаний з сутністю Comments (Коментарі) через зв'язок CommentId, який вказує на коментар, до якого поставлений лайк.

Ці зв'язки дозволять встановлювати залежності між різними сутностями в базі даних, що забезпечить зручний доступ та організацію інформації у даному проекті [12].

Провівши аналіз усіх вищеописаних відношень, можна зробити висновок, що всі вони нормалізовані до третьої нормальної форми, а отже процес розробки структури бази даних можна вважати завершеним.

2.2 Розробка структури інтерфейсу користувача соціальної мережі «Social Club»

У даному розділі описано процес розробки структури інтерфейсу користувача для соціальної мережі «Social Club». Зосереджуючись на основних сторінках, таких як головна, профіль користувача та чат, були визначені елементи та їх взаємодія [13].

Головна сторінка є центральним місцем, де користувач переглядати контент (в нашому випадку – пости), який продукують користувачі з його мережі контактів. Користувач може коментувати пости, виражати свої враження та взаємодіяти з контентом. На головній сторінці також розташовані функції пошуку користувачів та навігаційні кнопки для переходу між іншими сторінками.

Сторінка профілю дозволяє користувачеві переглядати власний профіль та інформацію про себе. На сторінці профілю користувач може переглядати свої власні публікації, а також список своїх підписників. Користувач має можливість редагувати свій профіль, додавати або змінювати фотографії та встановлювати налаштування приватності.

Чат є важливою функцією соціальної мережі, яка дозволяє користувачам комунікувати один з одним. На сторінці чату користувач може обмінюватися повідомленнями зі своїм колом контактів, надсилати повідомлення користувачам, які не є в колі його контактів. Чат також може містити додаткові функції, такі як можливість створення групових чатів або відправлення медіафайлів.

Користувач може переглядати сторінки інших користувачів, де вони представляють свій профіль та публікації. На сторінці іншого користувача можна переглядати його публікації у вигляді сітки, що складається з маленьких фотографій публікацій користувача (thumbnails), натискання на які переводить нас на сторінку обраного допису, де ми можемо писати коментарі та взаємодіяти з ними.

Сторінка дописів відображає перелік публікацій, де користувач може переглядати та взаємодіяти з ними. Користувач може коментувати пости, ставити вподобання, а також поділитися постом на своїй сторінці

Під час розробки структури інтерфейсу користувача було враховано зручність використання, а також прагнення забезпечити зручну навігацію та взаємодію для кожного користувача соціальної мережі «Social Club».

2.3 Розробка алгоритму взаємодії програмних модулів соціальної мережі

Технологія ASP.Net використовує модель запит-відповідь для обробки веб-запитів [14, 15]. Під час надсилання запиту на сервер, ASP.Net виконує наступні етапи:

Контейнер Middleware є потужним механізмом, який дозволяє розширювати та налаштовувати функціональність додатку в ASP.Net Core з допомогою

додаткових шарів програмного забезпечення, які обробляють запити перед та після основного обробника запиту.

Маршрутизація (Routing): Визначає, який контролер та дія будуть виконуватися на основі URL та налаштувань маршрутів.

Створення контексту (Context Creation): Створює контекст, що містить інформацію про запит, таку як URL, параметри та заголовки.

Створення контролера (Controller Creation): Створює екземпляр контролера для обробки запиту та може мати залежності від сервісів або репозиторіїв.

Виконання дії (Action Execution): Викликає відповідну дію контролера, яка обробляє запит та виконує необхідну логіку.

Фільтрація (Filtering): Застосовує фільтри перед, під час або після виконання дії, які забезпечують додаткову логіку, таку як аутентифікація, авторизація, кешування або логування.

Представлення (View): Відправляє дані представлення на клієнт. Це може бути статична HTML-сторінка або динамічно згенероване представлення з використанням шаблонів та моделей даних.

Відповідь (Response): Формує відповідь на запит, яка може бути у форматі JSON, XML, файлу або редиректування на іншу сторінку. Життєвий цикл запиту зображено на рисунку 2.6.

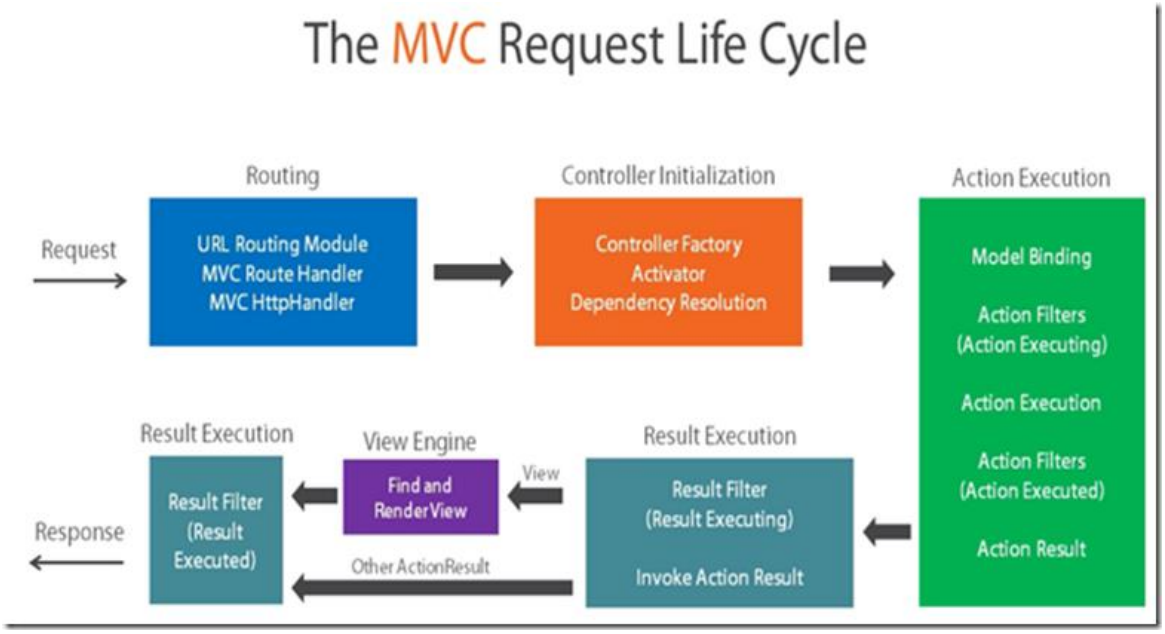


Рисунок 2.6 – Принципова схема життєвого циклу запиту в ASP.Net Core

При розробці middleware pipeline було використано наступні middleware компоненти, такі як ExceptionMiddleware, CorsMiddleware, ResponseCachingMiddleware, AuthenticationMiddleware та AuthorizationMiddleware, що додаються та налаштовуються для обробки відповідних аспектів запиту. Крім того, також використовуються Middleware для роботи з статичними файлами, маршрутизації та WebSocket комунікації з використанням SignalR.

Далі розглянемо патерни, що були використані при створенні додатку. Repository та Unit of Work є популярними підходами в проектуванні та реалізації архітектури додатків. Вони допомагають забезпечити розділення відповідальностей та підвищують простоту, підтримуваність та тестируваність коду [16].

Repository (репозиторій) - це компонент, який забезпечує доступ до даних та виконання операцій збереження, оновлення, видалення та отримання об'єктів даних. Кожен репозиторій відповідає за доступ до даних одного конкретного типу об'єктів, що взаємодіють з визначеною частиною логіки додатку. Наприклад,

UserRepository відповідає за доступ до даних користувачів, PostRepository - за доступ до даних постів тощо. Репозиторії зазвичай містять методи, такі як GetById, GetAll, Post, Update, Delete та інші, які виконують CRUD операції над об'єктами даних.

Unit of Work (одиниця роботи) – це концепція, що об'єднує кілька репозиторіїв та визначає контекст виконання декількох операцій з даними як єдину транзакцію. Вона забезпечує атомарність та консистентність змін, здійснюваних над даними в рамках однієї операції. Клас Unit of Work зазвичай містить методи SaveChanges або Commit, які виконують збереження змін в базу даних після виконання групи операцій.

Використання репозиторіїв та Unit of Work у контролерах дозволяє розділити логіку доступу до даних від логіки обробки запитів, що робить код більш структурованим та зручним для розширення.

Наглядний приклад архітектури додатків без використання репозиторіїв, з їх використанням, та використанням комбінації repository та UnitOfWork зображено на рисунку 2.7.

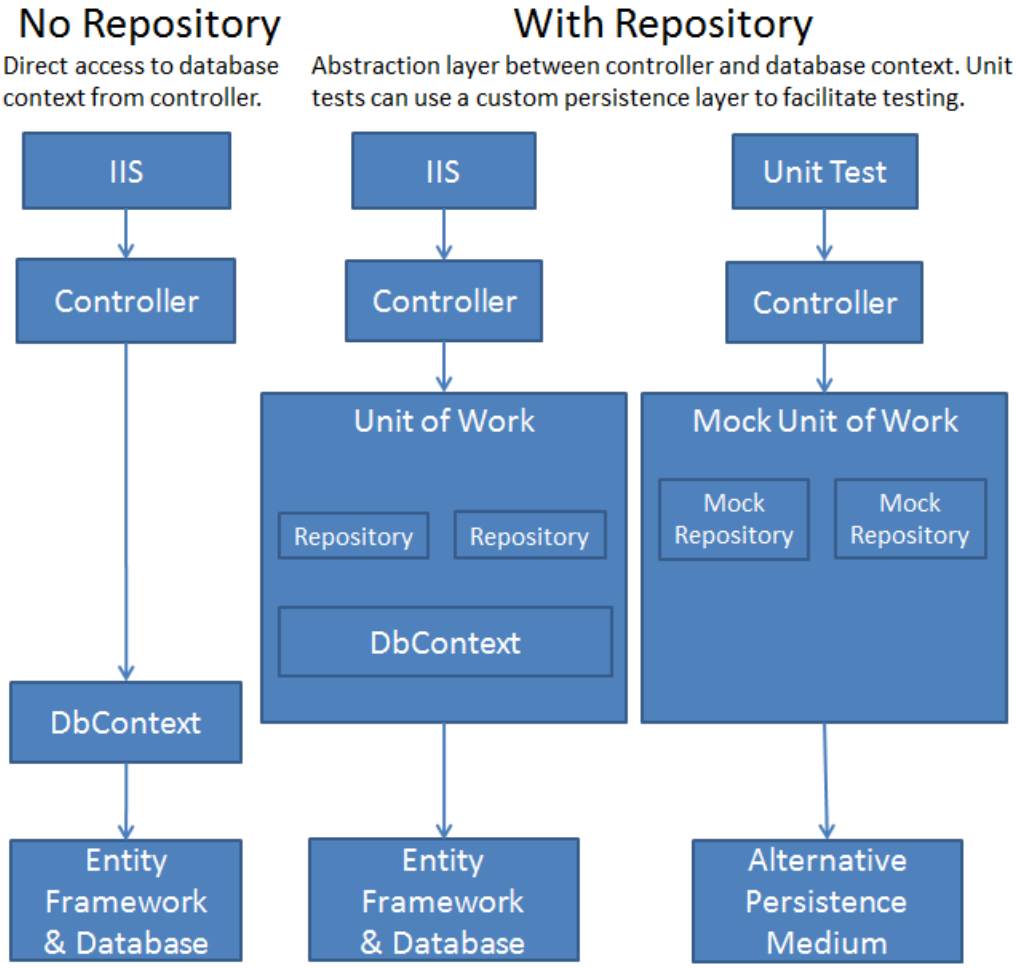


Рисунок 2.7 – Принципова схема додатків з використанням патерну UnitOfWork

Наприклад, у контролері UserController ви можете використовувати UserRepository для отримання, збереження, оновлення або видалення даних користувачів. Таким чином, контролер буде відповідальним за обробку запитів, валідацію даних, виконання бізнес-логіки та повернення відповідей клієнту, а репозиторій буде відповідальним за доступ до даних та виконання операцій з ними. Також, використання AutoMapper та базового контролера (BaseController) може спростити процес мапування даних та надати загальні функціональності для всіх контролерів у додатку.

Коментарі CommentController, повідомлення MessageController, пости PostController та підписники SubscribersController можуть використовувати відповідні репозиторії для взаємодії з даними цих сутностей.

Також, репозиторій BaseRepository може включати загальні методи доступу до даних, які можна спільно використовувати в усіх репозиторіях.

Dapper - це легковажний та швидкий MicroORM для .NET, який надає можливість виконувати операції з базою даних, мапувати результати запитів на об'єкти та забезпечувати ефективну роботу з даними [17]. Для порівняння ефективності Dapper та Entity Framework Core (EF Core), можна скористатись наступною матиматичною формулою, що вираховує ефективність ORM (Dapper або EF Core) на основі відношення між часом виконання запитів і кількістю виконаних запитів:

$$Ефективність = (1 / t) \cdot (1 / n),$$

де t - час виконання запиту (менші значення означають кращу продуктивність), n - кількість виконаних запитів (менша кількість запитів означає кращу продуктивність).

Шляхом оцінки ефективності було визначено, що ORM Dapper є ефективнішою, та доцільнішою у використанні при розробці соціалної мережі. Побачити, яке місце в програмі займає ORM Dapper можна на рисунку 2.8.

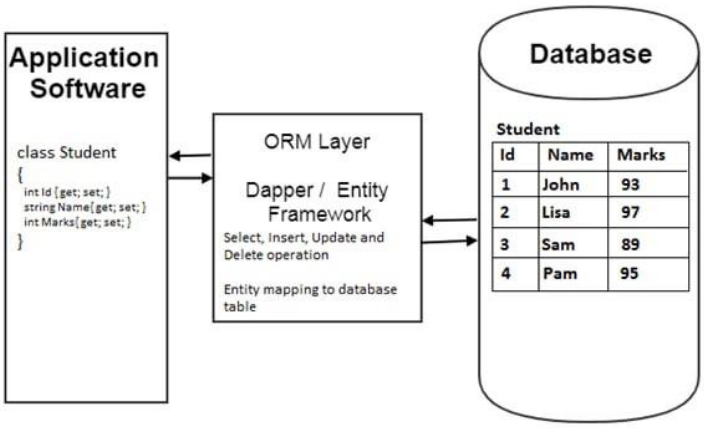


Рисунок 2.8 – Принципова схема ролі ORM в програмному додатку

Розглянувши ключові аспекти розробки серверної частини додатку, переходимо до розробки алгоритму роботи клієнтської частини.

ReactJS є бібліотекою JavaScript для побудови користувацьких інтерфейсів, яка базується на компонентах [18]. Алгоритм роботи ReactJS зображений на рисунку 2.9.

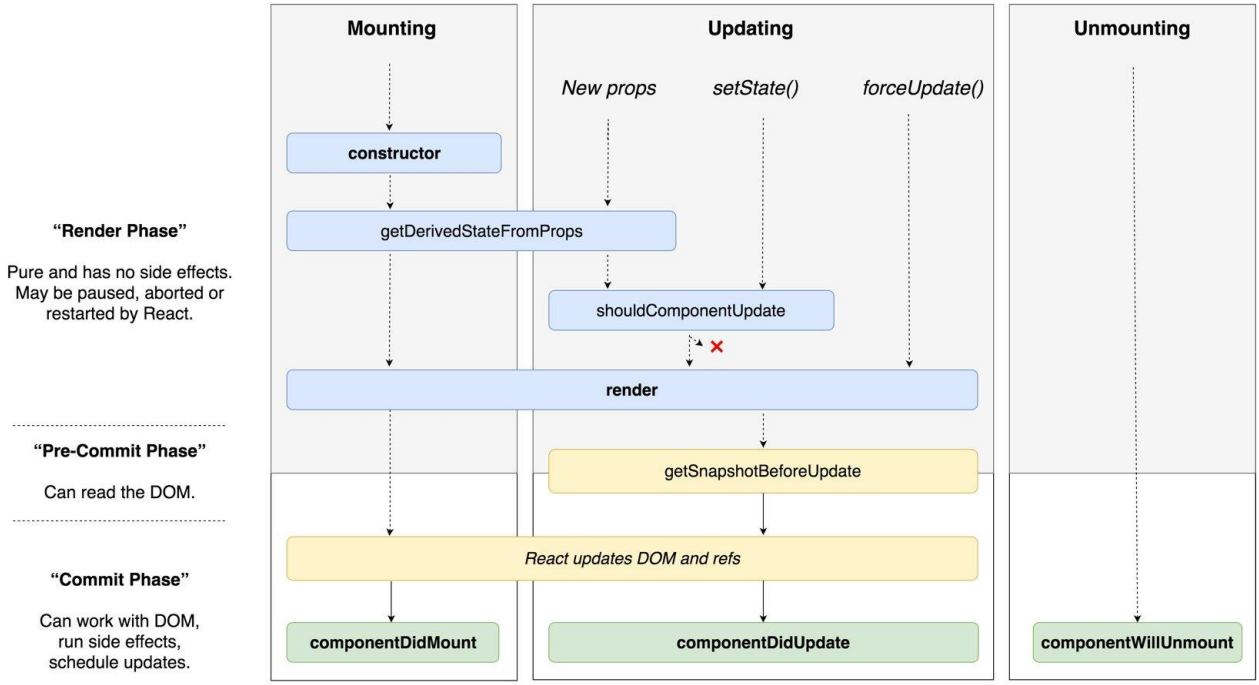


Рисунок 2.9 – Принципова схема алгоритму роботи бібліотеки ReactJS

Також при розробці клієнтської частини була використана бібліотека

Redux – яка створена для управління станом для JavaScript, алгоритм роботи якої включає наступні кроки [19]:

− Визначення початкового стану, де створюється початковий стан додатку.

− Створення дій (actions), де визначаються дії, які можуть відбуватись в додатку.

− Створення редукторів (reducers), де створюються редуктори, які змінюють стан на основі дій.

− Комбінування редукторів, де редуктори об'єднуються у кореневий редуктор.

− Створення store, де створюється об’єкт store зі станом і редуктором.

− Виклик дій за допомогою диспетчера (dispatcher) , де викликається метод dispatch() об'єкта store.

− Обробка дії в редукторі, де редуктор отримує дію і змінює стан.

− Оновлення стану, де стан додатку оновлюється згідно з виконаною дією.

− Оповіщення про зміни, де підписники отримують повідомлення про зміну стану.

− Доступ до стану, де можна отримати поточний стан за допомогою метода getState().

Описаний вище алгоритм зображено на рисунку 2.10.

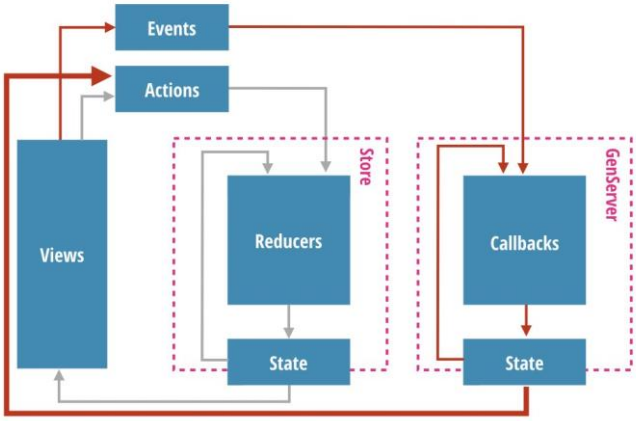


Рисунок 2.10 – Принципова схема алгоритму роботи бібліотеки Redux

2.4 Розробка алгоритму автентифікації користувача в соціальній мережі «Social Club»

Розробимо алгоритм роботи автентифікації користувача. Для автентифікації користувача використовуватиметься стандарт токена доступу на основі JSON, JSON Web Token. Він як правило, використовується для передачі даних для аутентифікації в клієнт-серверних програмах. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує цей токен для підтвердження своєї особи [20].

– Після отримання токена доступу JWT на основі JSON, процес автентифікації користувача в соціальній мережі «Social Club» включає наступні кроки:

– Вхід користувача. Користувач вводить свої облікові дані, такі як електронна пошта або ім'я користувача та пароль, на сторінці входу. Введені дані

перевіряються на належність до зареєстрованого користувача в базі даних соціальної мережі "Social Club".

− Перевірка введених даних. Система перевіряє, чи відповідають введені дані зареєстрованому користувачу. Якщо дані коректні, система генерує унікальний токен доступу для цього користувача.

− Видання та збереження токена доступу JWT. Згенерований токен доступу, який є унікальним ідентифікатором користувача, повертається на клієнтську сторону, де він зберігається. Токен може бути збережений у кеші або в локальному сховищі (наприклад, у веб-браузері) для подальшого використання при зверненні до захищених ресурсів.

− Аутентифікація при кожному запиті. При кожному запиті від клієнта до сервера, токен доступу передається разом з запитом у заголовку або як параметр запиту. Сервер перевіряє цей токен, перевіряє його цілісність, використовуючи секретний ключ, і перевіряє, чи не минув строк дії токена. Якщо токен дійсний, сервер дозволяє доступ до запитуваного ресурсу або виконує запитану дію.

− Управління токеном. У разі необхідності оновлення токена доступу (наприклад, через його закінчення строку дії), клієнт може здійснити запит на отримання нового токена з використанням свого діючого токена. При зміні облікових даних, які впливають на доступ користувача (наприклад, зміна пароля), може бути необхідно оновити токен.

Таким чином, алгоритм автентифікації користувача в соціальній мережі «Social Club» забезпечує безпеку та контроль доступу до ресурсів, забезпечуючи, що лише правомірні користувачі мають доступ до захищених функцій, таких як

сторінка чату та інші обмежені ресурси. Алгоритм його роботи зображений на рисунку 2.11.

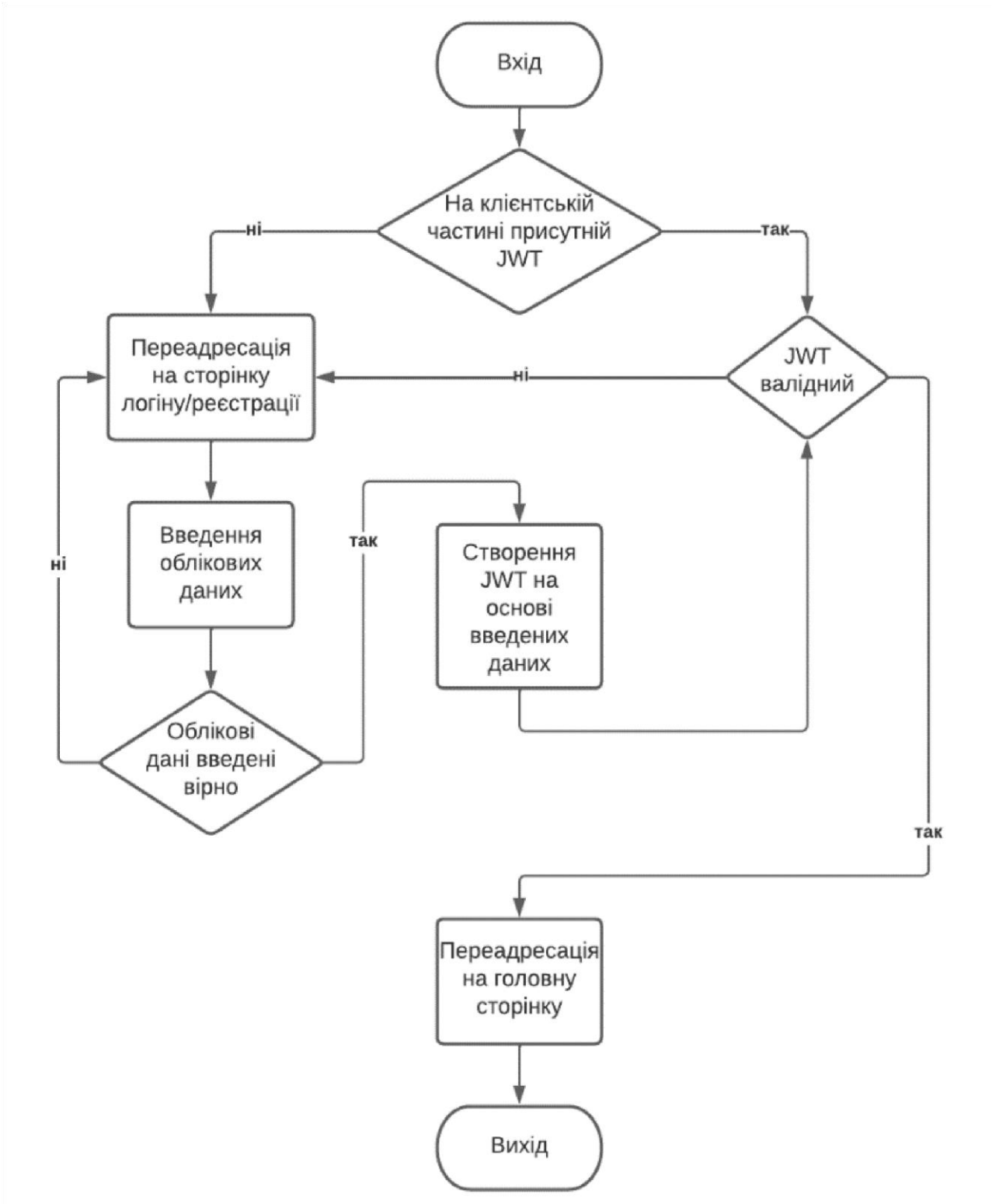


Рисунок 2.11 – Алгоритм автентифікації користувача в соціальній мережі "Social Club"

2.5 Висновок до розділу 2

У другому розділі дипломної роботи розглянуто структуру бази даних та інтерфейсу користувача для соціальної мережі «Social Club». Проведено аналіз ключових складових бази даних і розроблено реляційну структуру з таблицями, полями та зв'язками. Розроблено ефективні зв'язки та індекси для забезпечення доступу до даних. Щодо інтерфейсу користувача, проведено аналіз та проектування його структури з урахуванням зручності використання та функціональності. Обрано веб-інтерфейс як оптимальний варіант для доступу до соціальної мережі «Social Club». Розроблено структуру сторінок, включаючи реєстрацію, вхід та профіль користувача, використовуючи сучасні технології та дизайн-принципи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛІВ БАЗИ ДАНИХ ТА ІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Обґрунтування вибору мови програмування та середовища розробки

Мовою програмування для розробки серверної частини програмного модуля соціальної мережі «Social Club» було обрано C#. Серверна частина відіграє ключову роль у системі, обробляючи запити користувачів, виконуючи бізнес-логіку, спілкуючись з базою даних і надаючи результати на клієнтську частину. Вона складається з сервера, бази даних і додаткового програмного забезпечення, необхідного для правильного функціонування програми [21].

Клієнтська частина відповідає за доступність та зрозумілість програмного продукту для користувача. Вона забезпечує візуальне відображення та інтерактивність програми на стороні користувача, включаючи структуру, дизайн, поведінку та взаємодію елементів інтерфейсу. Клієнтська частина виконує роль відображення веб-сторінок, обробки подій користувача, передачі та отримання даних з сервера та інших функцій, необхідних для зручного та ефективного використання програмного продукту.

C# є об’єктно-орієнтованою мовою програмування з безпечною системою типізації для платформи .NET. Вона має високий рівень абстракції, зручний синтаксис та широкий набір вбудованих функцій і бібліотек, що полегшують розробку і прискорюють процес створення програм [22].

Переваги C# включають:

− Об'єктно-орієнтований підхід, що сприяє модульності і повторному використанню коду.

− Платформа .NET, що надає широкі можливості для розробки різних типів програм.

− Високорівневість, що полегшує написання коду і реалізацію бізнеслогіки.

− Багатофункціональність, яка включає роботу зі стрічками, масивами, колекціями, базами даних, мережевим програмуванням та іншими.

− Підтримка розробки, зокрема інтегроване середовище розробки Visual Studio та різноманітні сторонні інструменти.

Під час розробки програмного модуля були використані наступні мови програмування та технології: C#, JavaScript (з використанням React), HTML і CSS. JavaScript є високорівневою, інтерпретованою мовою програмування для розробки динамічних та інтерактивних веб-сторінок. HTML використовується для створення структури веб-сторінок, а CSS - для їх стилізації. Разом ці мови та технології доповнюють одна одну, надаючи можливість реалізувати широкий спектр функціональності та забезпечувати зручний та привабливий користувацький досвід.

Середовища програмування, використані під час розробки додатку, включають Visual Studio для мови C#, Visual Studio Code для JavaScript та DBeaver для роботи з базами даних. Visual Studio (VS) - інтегроване середовище розробки для мови C#, яке надає розширену підтримку створення, налагодження та керування проектами. Visual Studio Code (VS Code) - легке інтегроване середовище розробки для JavaScript, зі швидким редактором коду та масивною екосистемою розширень. DBeaver - потужний інструмент для роботи з базами даних, з підтримкою різних СУБД, включаючи можливості редагування, запитування та візуалізації даних. Ці середовища допомагають забезпечити продуктивність та

ефективність при розробці додатку, надаючи засоби для написання, налагодження та керування кодом, а також роботи з базами даних [23].

3.2 Розробка програмних модулів баз даних та інтерфейсу користувача

Першим етапом розробки додатку є конфігурація та налаштування веб додатку, включаючи різноманітні компоненти та сервіси, необхідні для його роботи. Для прикладу, було обрано фрагмент коду, що відповідає за додавання основних сервісів, що ініціалізуються при компілюванні проекту.

```
builder.Services.AddControllers(); builder.Services.AddSignalR();
builder.Services.AddAutoMapper(typeof(Program).Assembly);
builder.Services.AddTransient<IUnitOfWork, UnitOfWork>();
builder.Services.AddTransient<ISecretHasher, SecretHasher>();
builder.Services.AddTransient<IJwtClaimsService, JwtClaimsService>();
var bus = RabbitHutch.CreateBus(Environment.GetEnvironmentVariable(“RabbitMq”));
builder.Services.AddSingleton(bus).
```

Наступним кроком є реалізація класу UnitOfWork. Основними частинами цього класу є:

− Поля і залежності. Код містить приватні поля для зберігання з'єднання з базою даних (_connection), транзакції (_transaction) та репозиторіїв (_postRepository, _userRepository, тощо). Також використовується залежність IConfiguration, яка передається у конструктор;

− Конструктор. У конструкторі UnitOfWork виконується ініціалізація з'єднання з базою даних та початок транзакції. З'єднання та транзакція отримуються з конфігурації, використовуючи рядок підключення DefaultConnection;

− Властивості репозиторіїв. Клас UnitOfWork містить властивості, які повертають екземпляри репозиторіїв, таких як UserRepository, PostRepository і т.д. Якщо екземпляр репозиторію вже ініціалізовано, він повертається замість

створення нового. Репозиторії отримують транзакцію у якості параметра конструктора;

− Методи Commit() та RollBack(). Метод Commit() виконує збереження змін у базі даних шляхом виклику методу Commit() на об'єкті транзакції. Якщо виникає помилка, транзакція відкатується за допомогою методу Rollback(). Після цього транзакція і репозиторії скидаються до початкового стану;

− Метод Dispose(). Цей метод викликається для звільнення ресурсів, таких як з'єднання з базою даних та транзакція. Він також забезпечує виклик Dispose() для транзакції та з'єднання. Виклик GC.SuppressFinalize(this) попереджає фіналізацію об'єкта у сбірнику сміття.

Після цього, варто приступити до створення контролерів та репозиторіїв.

Контролери є важливими компонентами архітектури програмного забезпечення і широко використовуються в розробці програмних додатків. Наведемо основні властивості контролерів:

− Контролери є частиною шару презентації в архітектурі додатка;

− Контролери відповідають за обробку вхідних HTTP-запитів та відповідну взаємодію з іншими компонентами системи;

− Контролери приймають запити від клієнтів, виконують необхідну обробку та повертають відповідь;

− Контролери можуть отримувати дані з запиту, виконувати бізнес-логіку, взаємодіяти з сервісами та репозиторіями, і повертати результати у відповідь.

Далі можна побачити приклад реалізації одного з методів контролера UserController, що відповідає за обробку всіх запитів, що стосуються користувачів.

Цей метод відповідає за створення нового користувача, тобто реєстрацію, та виконує певну бізнес-логіку.

```
[HttpPost]
public async Task<IActionResult> AddUser(AddUserDto newUser)
{
   User mappedUser = _mapper.Map<User>(newUser);
   int userId = await _unitOfWork.UserRepository.AddUserAsync(mappedUser);

   await SendConfirmEmail(userId);

   string token = await CreateUserJWTAcessToken(mappedUser.UserName, userId);
_unitOfWork.Commit();

   return Ok(token);
  }.
```

Наведемо основні властивості репозиторіїв:

− Репозиторії відповідають за доступ до даних і виконання операцій збереження, видалення, оновлення та отримання даних з бази даних або іншого джерела даних.

− Надають абстракцію над джерелом даних та реалізують операції, необхідні для роботи з цими даними;

− Репозиторії допомагають відокремити бізнес-логіку від деталей доступу до даних, що спрощує тестування та збереження цілісності даних;

− Вони можуть містити методи для отримання окремих записів, списків даних, виконання пошуку, фільтрації та інших операцій, пов'язаних з доступом до даних.

Далі можна побачити приклад одного з методів класу UserRepository, що безпосередньо формує, та надсилає запит до бази даних у вигляді SQL скрипта,

який надходить в базу даних за допомогою Dapper на створення в ній запису нового користувача, та повертає id новоствореного користувача.

```
public async Task<int> AddUserAsync(User entity)
{
    ArgumentNullException.ThrowIfNull(entity);
return await Connection.ExecuteScalarAsync<int>(
“INSERT INTO [User](Email, Name, UserName,
CreationDate, Password) OUTPUT
INSERT.[UserId] VALUES(@Email, @Name, @UserName, @CreationDate, @Password);”,
param: new { Email = entity.Email, Name = entity.Name, UserName = entity.UserName,
CreationDate = entity.CreationDate, Password = entity.EncryptedPassword },
transaction: Transaction
    );
}.
```

Контролери та репозиторії співпрацюють один з одним: контролери використовують репозиторії для отримання та збереження даних, а репозиторії використовуються контролерами для взаємодії з джерелом даних. Це дозволяє забезпечити високу рівень модульності, розділення рівнів абстракції, повторне використання та розширення коду.

Далі буде розглянуто процес формування та надсилання запиту з клієнтської частини, щоб в подальшому цей запит був оброблений контролером. За допомогою таких запитів і відбувається взаємодія між клієнтом та сервером [24].

Запит формується за допомогою виклику функції та передачі в неї всіх необхідних параметрів, далі payload. Далі описана функція, що відповідає за виклик безпосередньо тієї функції, що надсилатиме запит на сервер.

```
async function addUser(fullname, userName, email, password)
    {
      const body = {      name:
      fullname,      userName:
      userName,          email:
      email,
          encryptedPassword: password,
       }
```

```
      await                AddUser(body)
.then((data) => data.json())
    .then((response) => Cookies.set("JWT", response))

  await navigate(`/${ ConstantsList.ROUTE_MAIN_PAGE}`)
 }.
```

Після цього, виконується функція AddUser, що приймає весь необхідний payload. Тіло цієї функції зображено далі.

```
      export function AddUser(payload)
      {
        return      baseApiFetch("api/User/AddUser",      {
method: "POST",
            body: JSON.stringify(payload),
            headers:
    {
        Accept: "application/json",
            "content-type": "application/json",
            },
});}.
```

3.3 Тестування програмних модулів соціальної мережі

Для тестування додаток був запущений на localhost:7004, що означає що він не є в публічному доступі. При переході на головну сторінку додатку, нас переносить на сторінку реєстрації, адже доступ до додатку мають лише авторизовані користувачі. Дизайн сторінки реєстрації зображено на рисунку 3.1.

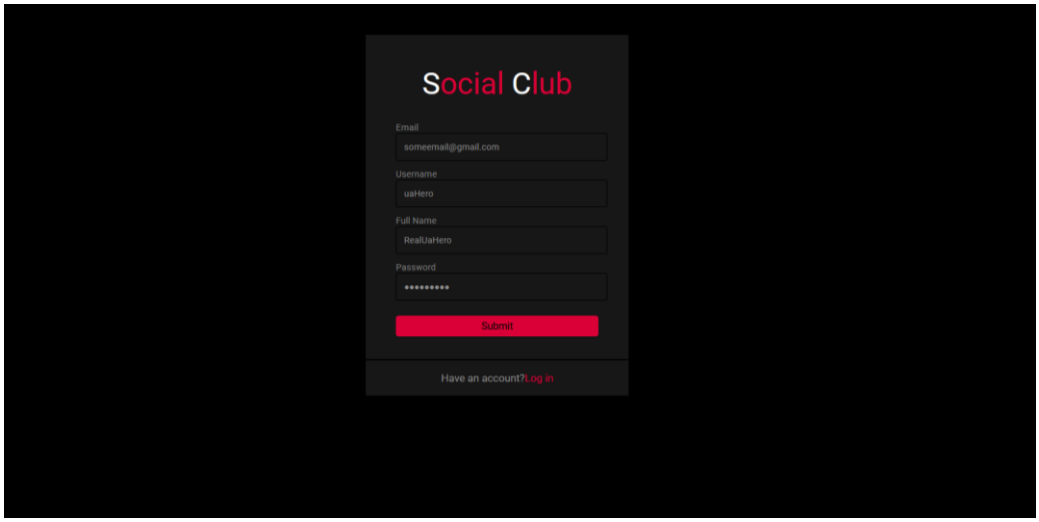


Рисунок 3.1 – Загальний вигляд інтерфейсного вікна реєстрації користувача

Також є можливість входу в систему шляхом логіну, за умови що користувач вже зареєстрований в системі, це зображено на рисунку 3.2.

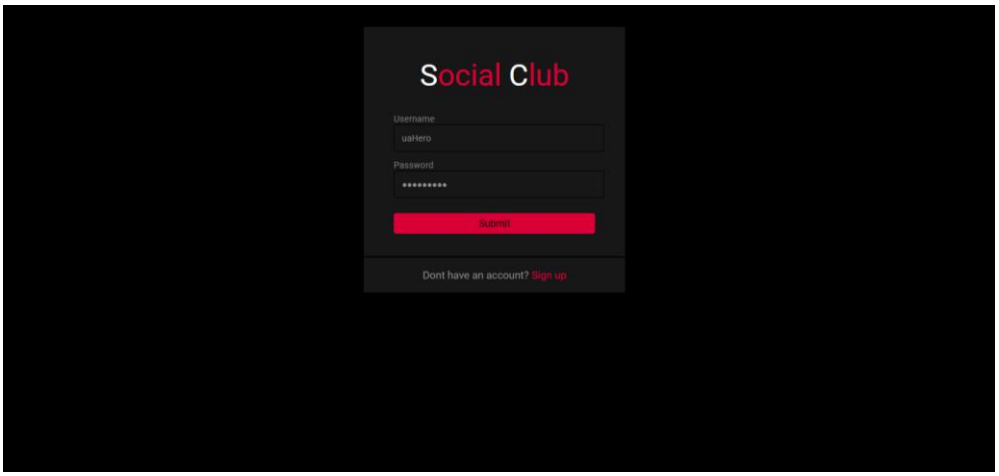


Рисунок 3.2 – Загальний вигляд інтерфейсного вікна логіну користувача

Після входу в систему нас зустрічає головна сторінка, на якій можна переглядати дописи інших користувачів, отримувати рекомендації щодо підписок, та наявний доступ до навігації по додатку. Вигляд головної сторінки зображено на рисунку 3.3.

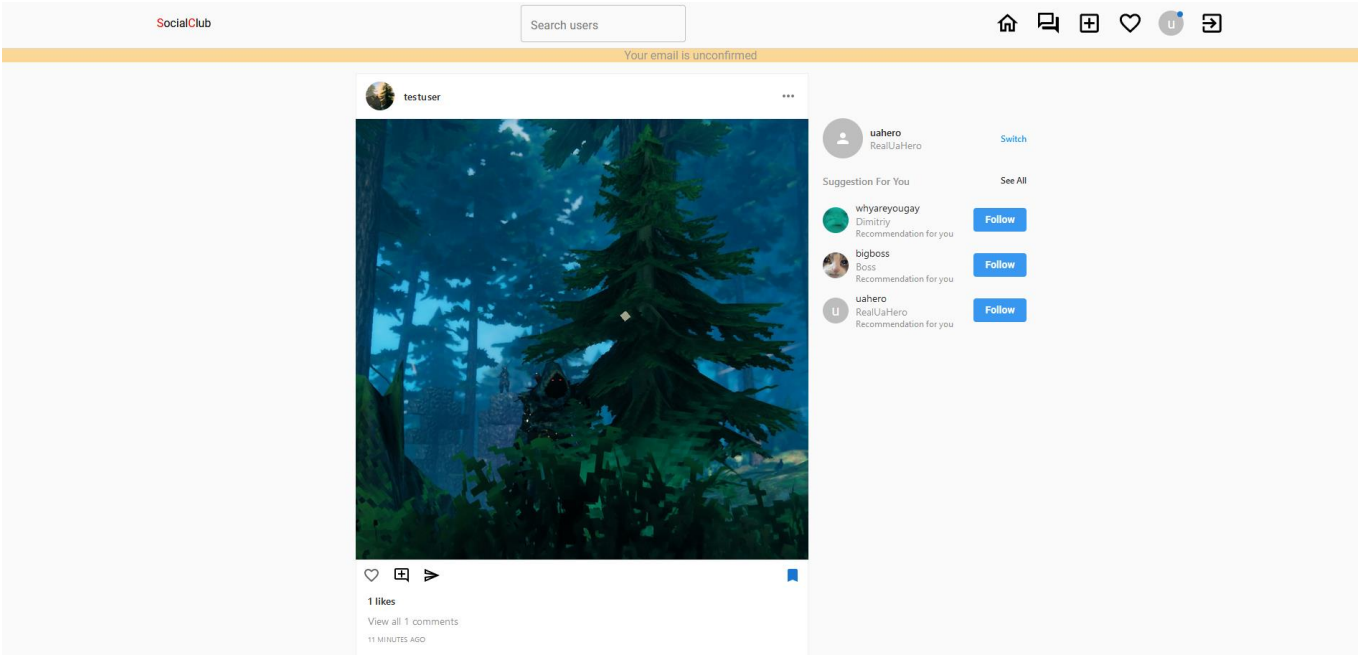


Рисунок 3.3 – Загальний вигляд інтерфейсного вікна головної сторінки

Звернувши увагу на кнопки навігації в верхньому правому кутку екрана, бачимо, що користувач також має доступ до сторінок власного профілю (рис. 3.4), на якому можна керувати власним аккаунтом, до чатів (рис. 3.5), де можна спілкуватися з іншими користувачами, та рекомендацій (рис. 3.6), де можна переглядати дописи відповідні інтересам користувача.

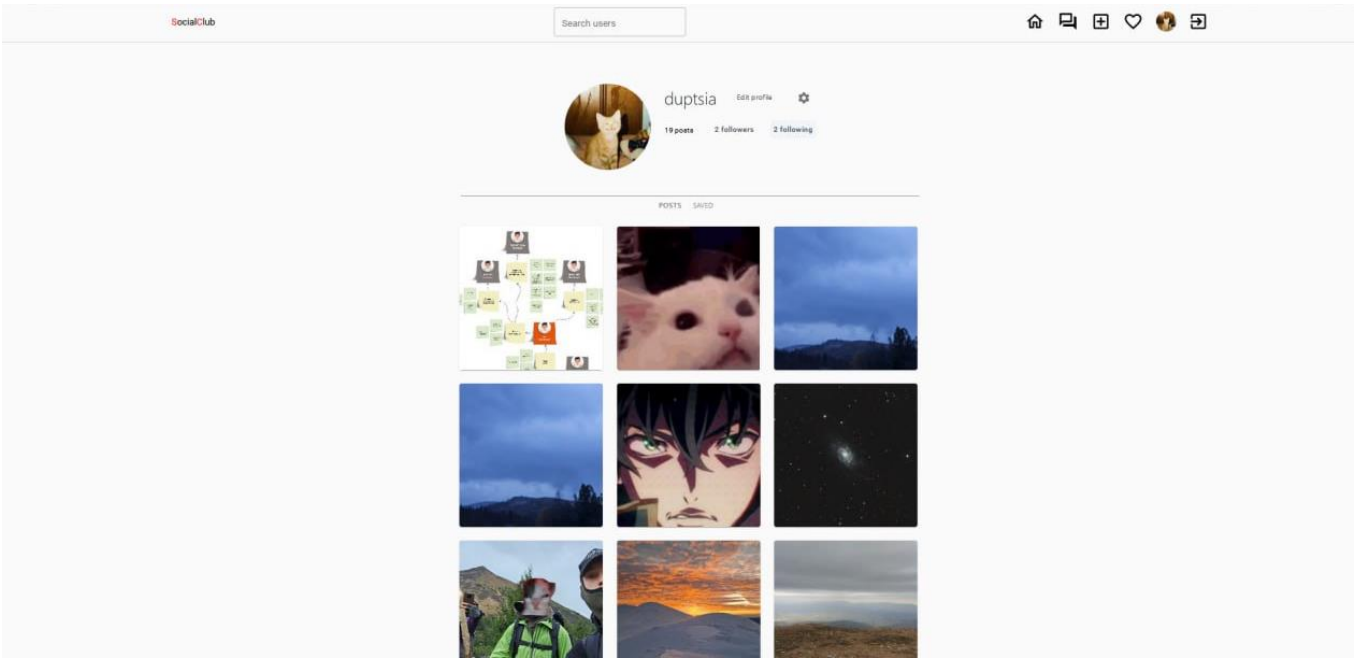


Рисунок 3.4 – Загальний вигляд інтерфейсного вікна профілю користувача

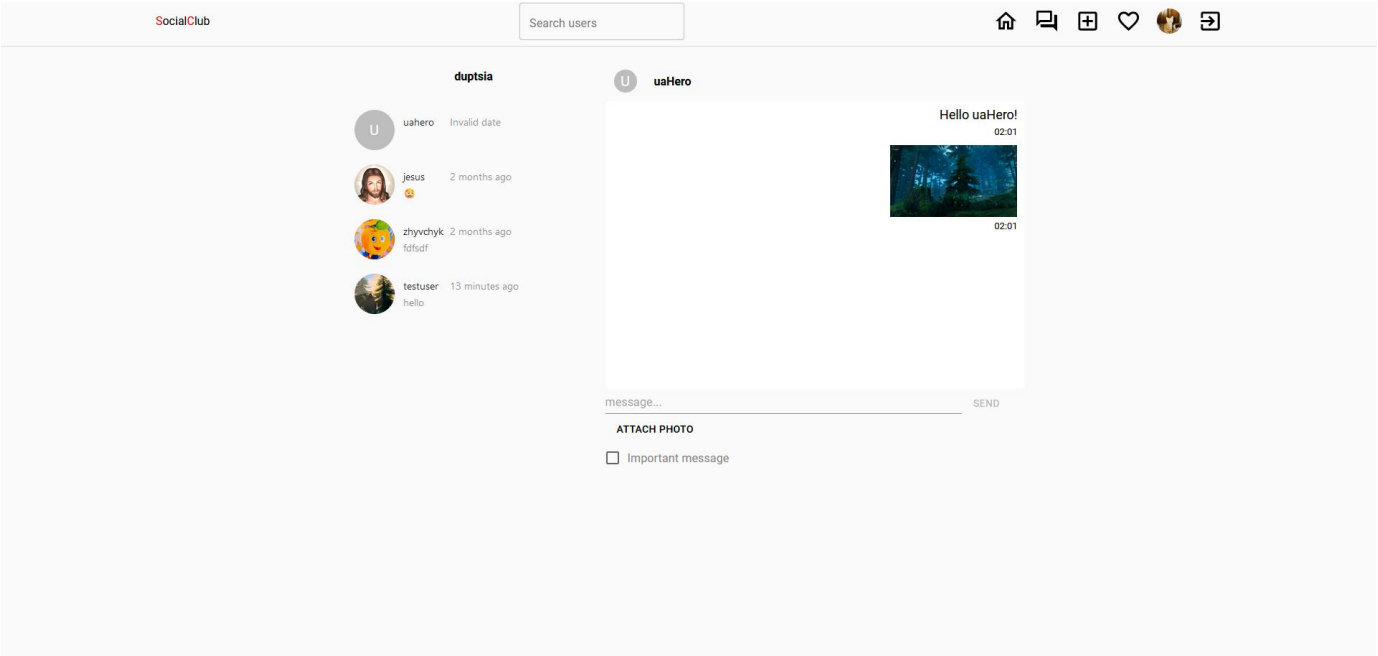


Рисунок 3.5 – Загальний вигляд інтерфейсного вікна сторінки чатів

Також, для взаємодії з дописами, передбачена сторінка для перегляду конкретного допису (рис. 3.7)

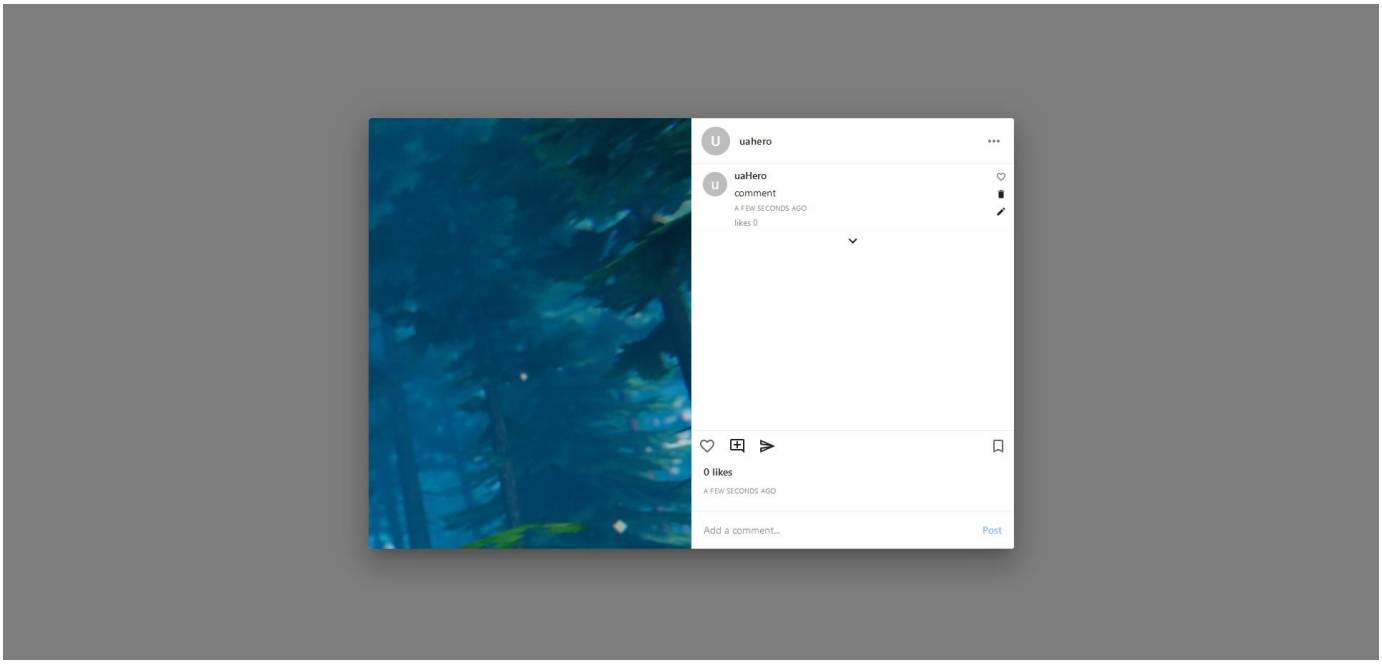


Рисунок 3.7 – Загальний вигляд інтерфейсного вікна сторінки допису

Після завершення тестування було перевірено функціональність системи користувача, і результати були успішними. Всі передбачувані функції, пов'язані з додаванням нових користувачів, працюють коректно та відповідають очікуванням.

3.4 Висновок до розділу 3

Отже, у третьому розділі дипломної роботи був обґрунтований вибір мови програмування та програмного середовища для реалізації програмних модулів бази даних та інтерфейсу користувача.

Були розглянуті мови програмування C#, JavaScript (React), HTML та CSS. Здійснено порівняльну характеристику мов C#, Python, Java, JavaScript з метою обґрунтування вибору оптимальної мови програмування для досягнення поставленої мети. Також були детально розглянуті програмні середовища Visual Studio 2022, Visual Studio Code та DBeaver, виокремлені їх особливості та переваги.

У розділі були розроблені програмні модулі та програмне забезпечення, яке забезпечує взаємодію між цими модулями. Були наведені приклади фрагментів коду, які ілюструють реалізацію окремих функцій та взаємодію між модулями. Кожен фрагмент коду супроводжувався відповідним описом, що детально пояснював його функціонал та роль у системі.

Для перевірки роботи програмного модуля був наведений тестовий приклад роботи програми, а також проведено тестування, що дозволило підтвердити коректну роботу програмного модуля та відповідність його функціональних вимог.

ВИСНОВКИ

Поставлені перед бакалаврською дипломною роботою задачі виконано в повному обсязі, а саме:

- зроблено огляд та проаналізовано відомі соціальні мережі;
- спроєктовано складові програмні модулі соціальної мережі «Social Club», базу даних та інтерфейс користувача;
- реалізовано програмні модулі соціальної мережі «Social Club» у вибраних мовних середовищах, а саме базу даних та інтерфейс користувача;
- протестовано роботу програмних модулів бази даних та інтерфейсу користувача соціальної мережі «Social Club».

У ході виконання бакалаврської дипломної роботи були проаналізовані найпопулярніші соціальні мережі, проведено їх класифікацію та проаналізована цільова аудиторія, в результаті чого було обрано певний тип реалізації соціальної мережі «Social Club».

Було проаналізовано різні системи управління базами даних та обрано і розроблено оптимальну структуру бази даних, також було проаналізовано різні типи структур вебсайтів та обрано і розроблено оптимальну структуру вебресурсу та розроблено алгоритми функціонування частин вебресурсу.

Відповідно з завданням до бакалаврської роботи були розроблені програмні модулі бази даних та інтерфейсу користувача, а також реалізована взаємодія між програмними модулями. Також, за допомогою автоматизованих тестів було проведено тестування.

Мета роботи - розширення функціональних можливостей соціальної мережі за допомогою розробки програмного модуля соціальної мережі досягнута за рахунок того, що в порівнянні з найближчим аналогом [25] досягнуто простішого впровадження та надано можливість вертикального масштабування серверів. Також спрощено управління додатком та зменшено кількість мережевих взаємодій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Г.А.Гайна ОСНОВИ ПРОЕКТУВАННЯ БАЗ ДАНИХ [Електронний ресурс] – Режим доступу: http://kist.ntu.edu.ua/textPhD/osnovProectBD.pdf

2. CT Carr, RA Hayes 2015 Social media: Defining, developing, and divining.

3. Соціальна мережа Instagram. URL: https://www.instagram.com/ .

4. P Sareen, P Kumar 2015 NoSQL Database and its comparison with SQL Database .

5. J Warren, N Marz 2015 Big Data: Principles and best practices of scalable realtime data systems.

6. Роберт Виейр. 2011. Програмування баз даних Microsoft SQL Server.

7. ОВ Адамик 2016 Бази і сховища даних–інформаційний фундамент бухгалтерського обліку та аналізу.

8. What is react framework. URL: https://www.techmagic.co/blog/why-weuse-react-js-in-the-development/ .

9. What are HTML and CSS used for. URL: https://www.w3.org/standards/webdesign/htmlcss.

10. B Shneiderman, C Plaisant, MS Cohen, S Jacobs 2016 Designing the user interface : strategies for effective human-computer interaction.

11. What is react and how to work with it. URL: https://uk.legacy.reactjs.org/docs/getting-started.html.

12. Програмний модуль. іт словник укр. URL: http://xn--r1a3b.xn-b1amgblet.xn--j1amh/index.php/Програмний_модуль.

13. J Johnson 2020 Designing with the mind in mind: simple guide to understanding user interface design guidelines .

14. ASP.NET documentation. Introducing asp.net. URL: https://learn.microsoft.com/uk-ua/aspnet/overview.

15. Middleware in ASP.NET Core. URL: https://www.dotnetcurry.com/aspnetcore/middleware-in-aspnetcore.

16. What is client-server architecture. URL: https://www.simplilearn.com/whatis-client-server-architecture-article.

17. Welcome To Learn Dapper. URL: https://www.learndapper.com .

18. What is react and how to work with it. URL: https://uk.legacy.reactjs.org/docs/getting-started.html.

19. Why Redux toolkit is how to use redux today. URL: https://redux.js.org/introduction/why-rtk-is-redux-today.

20. What is JWT and how to use it. URL: https://jwt.io/introduction .

21. Why you should use c#. URL: https://learn.microsoft.com/ukua/dotnet/csharp/tour-of-csharp/.

22. Alex Mackey; William Stewart Tulloch; Mahesh Krishnan (10 October 2012). Introducing .NET 4.5. Apress. pp. 143.

23. Larson, Jennifer M. (11 May 2021). «Networks of Conflict and Cooperation». Annual Review of Political Science. 24 (1): 89–107.

24. What is client-server architecture. URL: https://www.simplilearn.com/whatis-client-server-architecture-article.

25. Соціальна мережа Instagram. URL: https://www.instagram.com/.

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль соціальної мережі "Social Club"

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,28% Схожість 2,72%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Лесюк М.М.

Керівник роботи Озеранський В.С.

ДОДАТОК Б Інструкція користувача

Відкрийте веб-браузер на своєму пристрої та введіть URL-адресу соціальної мережі «Social Club».

Після завантаження головної сторінки соціальної мережі «Social Club» ви побачите екран з полями для реєстрації вашого облікового запису (рис. Б.1). Якщо ви ще не маєте облікового запису, введіть необхідні облікові дані. Введіть свою електронну пошту, ім'я користувача, повне ім’я, та пароль у відповідних полях. Після реєстрації вам потрібно підтвердити свою електронну пошту, для використання функції чату.

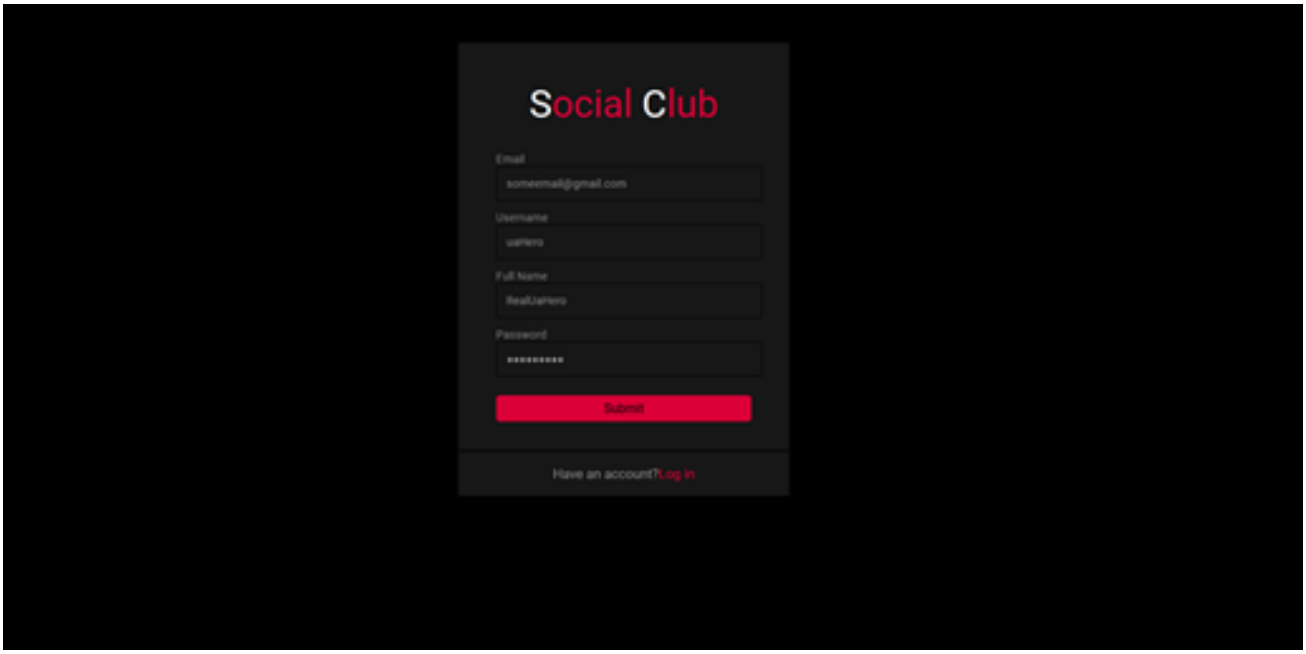


Рисунок Б.1 – Загальний вигляд інтерактивного вікна реєстрації користувача

Якщо ви вже маєте обліковий запис, натисніть кнопку «Log in», та введіть свої облікові дані для входу в нього (електронну пошту та пароль) (рис. Б.2).

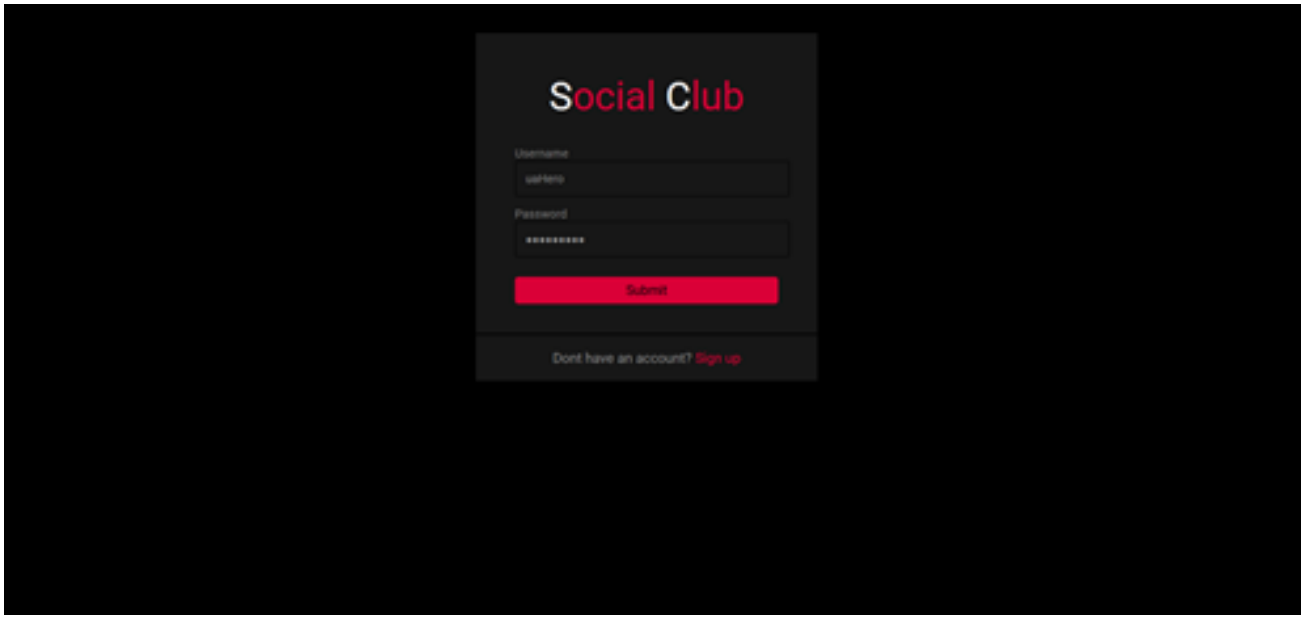


Рисунок Б.2 – Загальний вигляд інтерактивного вікна входу користувача у свій обліковий запис

Після успішного входу або створення облікового запису вас буде перенаправлено на головний екран соціальної мережі «Social Club» (рис. Б.3). Тут ви зможете переглядати фотографії і Gif-матеріали, додавати нові публікації, коментувати і ставити лайки.

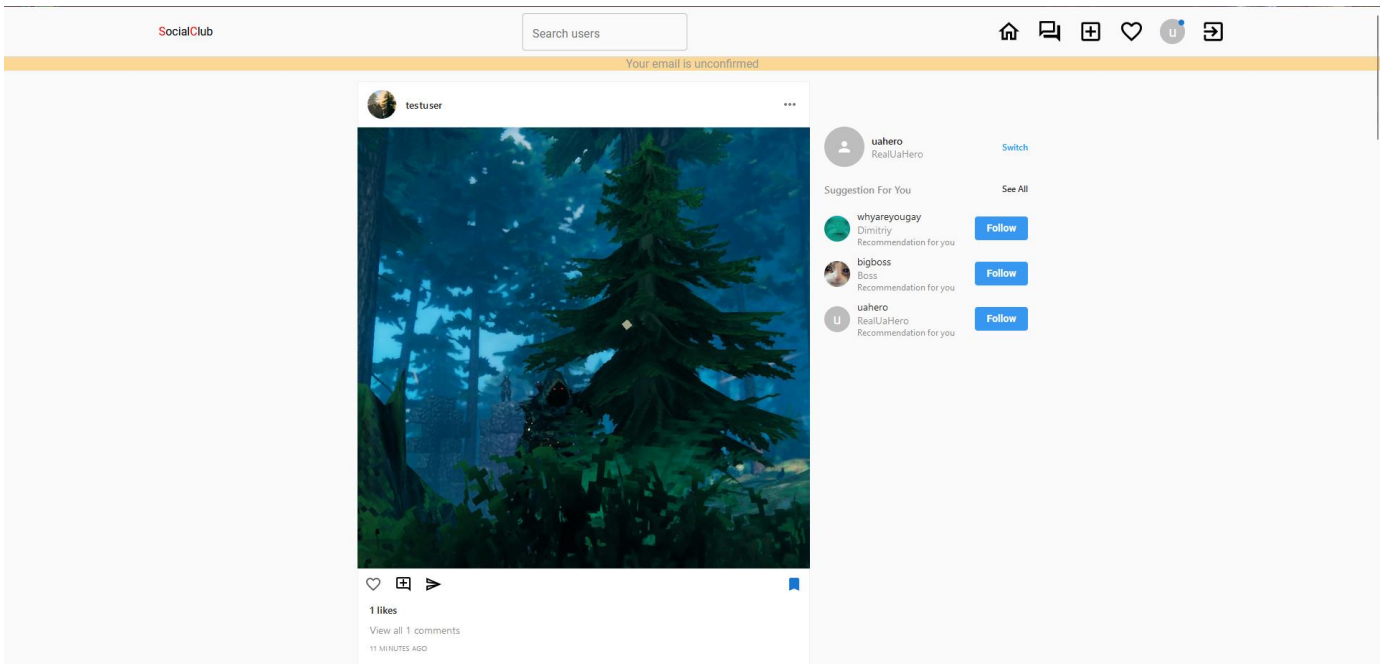


Рисунок Б.3 – Загальний вигляд інтерфейсного вікна головної сторінки

Щоб додати нову публікацію, натисніть кнопку "Додати публікацію" або іконку, яка зображується плюсом (+). За допомогою цієї кнопки ви зможете вибрати фотографію зі свого комп'ютера. Додайте опис, щоб описати свою публікацію. Після завершення редагування натисніть кнопку "Post", щоб розмістити публікацію на вашому сайті (рис. Б.4).

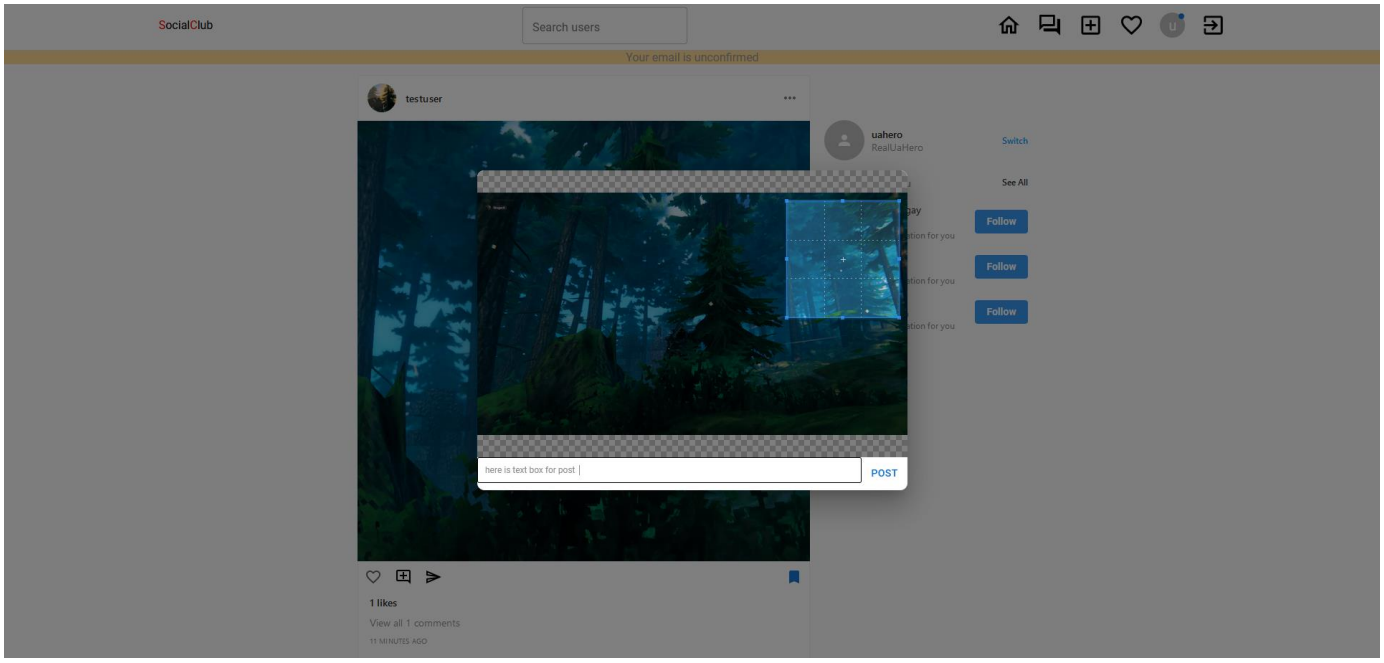


Рисунок Б.4 – Загальний вигляд інтерфейсного вікна публікації допису

Щоб переглянути публікації інших користувачів, перейдіть на сторінку їх профілю, натиснувши на їх ім'я або фото (рис. Б.5). На головному екрані є можливість перегляду останніх публікацій інших користувачів, на яких ви підписані. На екрані рекомендацій є можливість перегляду останніх публікацій інших користувачів, які вам можуть бути цікаві (рис. Б.6). На сторінці їх профілю ви зможете переглядати їхні публікації, залишати коментарі та ставити лайки (рис. Б.7).

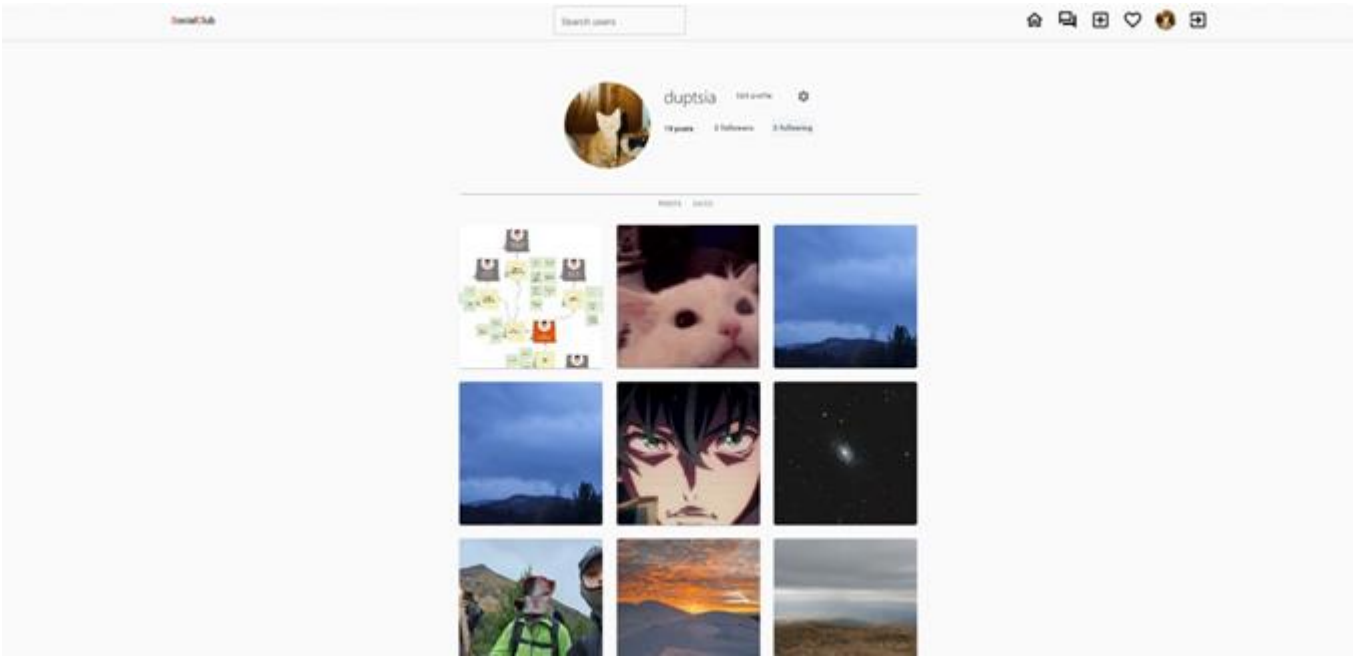


Рисунок Б.5 – Загальний вигляд інтерфейсного вікна сторінка користувача

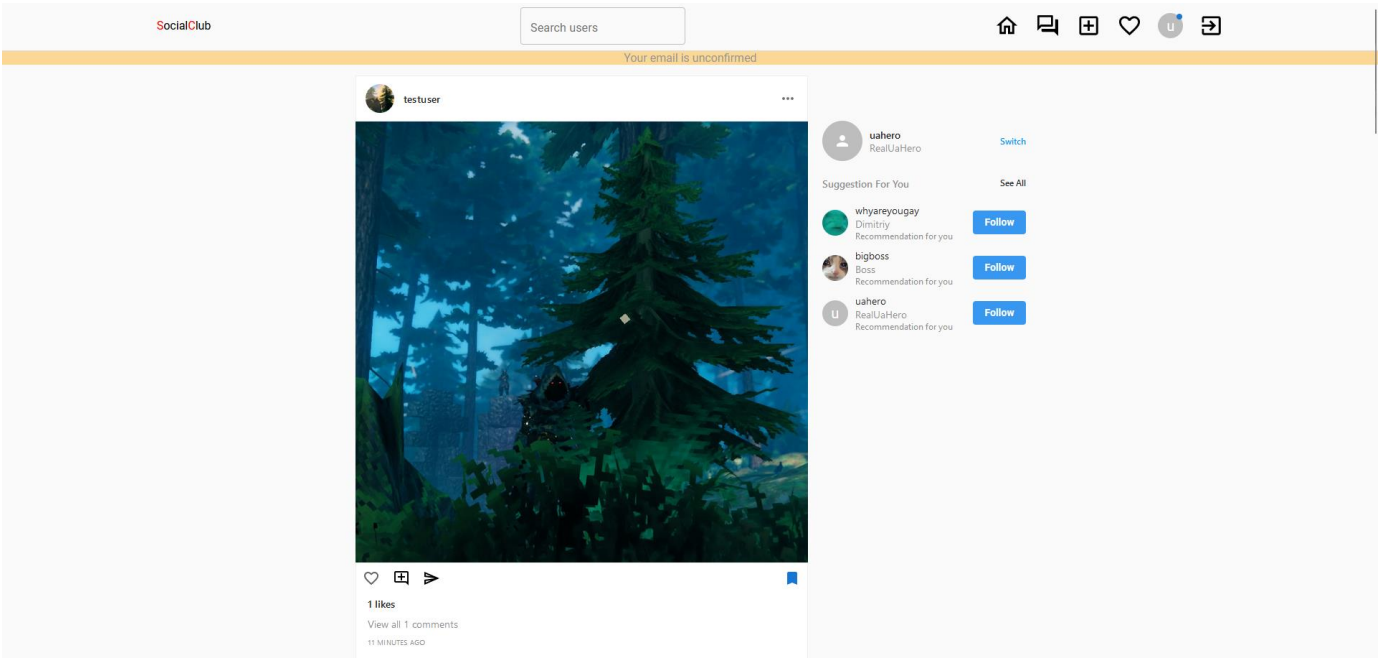


Рисунок Б.6 – Загальний вигляд інтерфейсного вікна сторінка рекомендацій

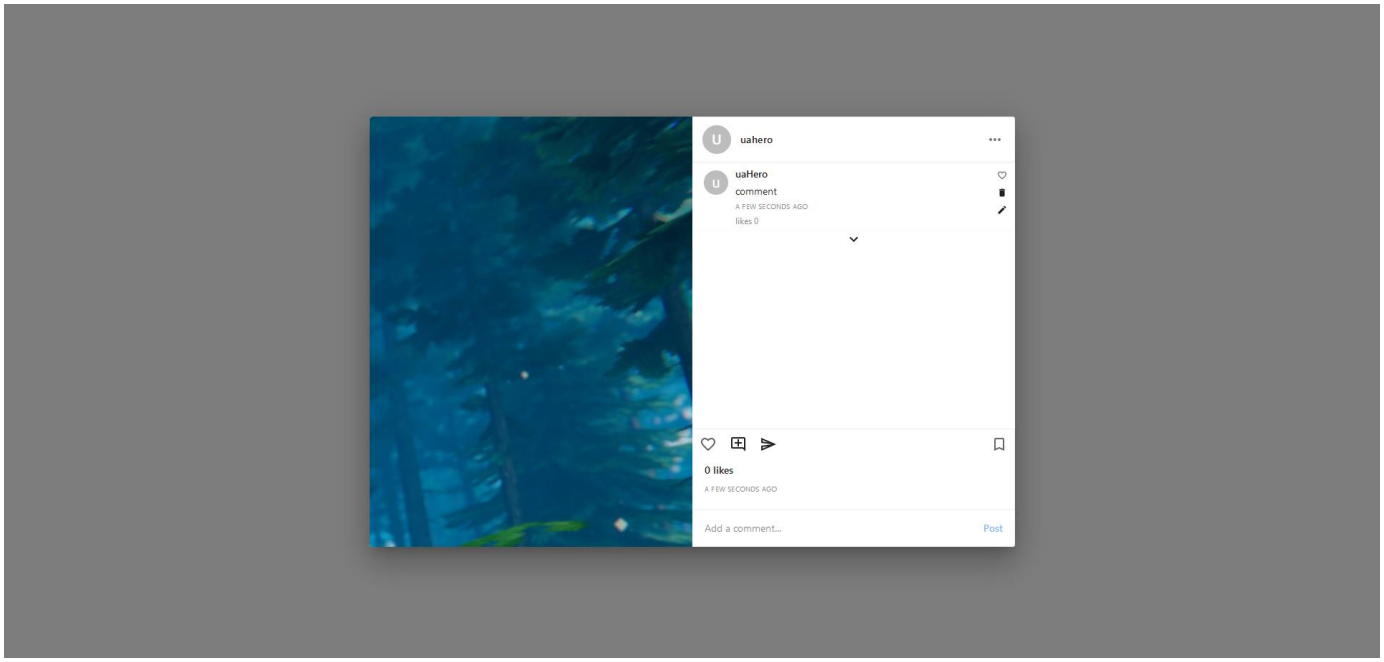


Рисунок Б.7 – Загальний вигляд інтерфейсного вікна сторінка допису

Для спілкування з іншими користувачами в соціальній мережі «Social Club» доступний модуль чату. Зверху, на панелі навігації по соціальній мережі «Social Club», ви знайдете відповідну іконку, яка є кнопкою "Чату" та відкриває чатове вікно. У цьому вікні, за умови підтвердження електронної пошти, ви зможете обмінюватися повідомленнями з іншими користувачами, ділитися фотографіями, а також створювати групові чати (рис. Б.8).

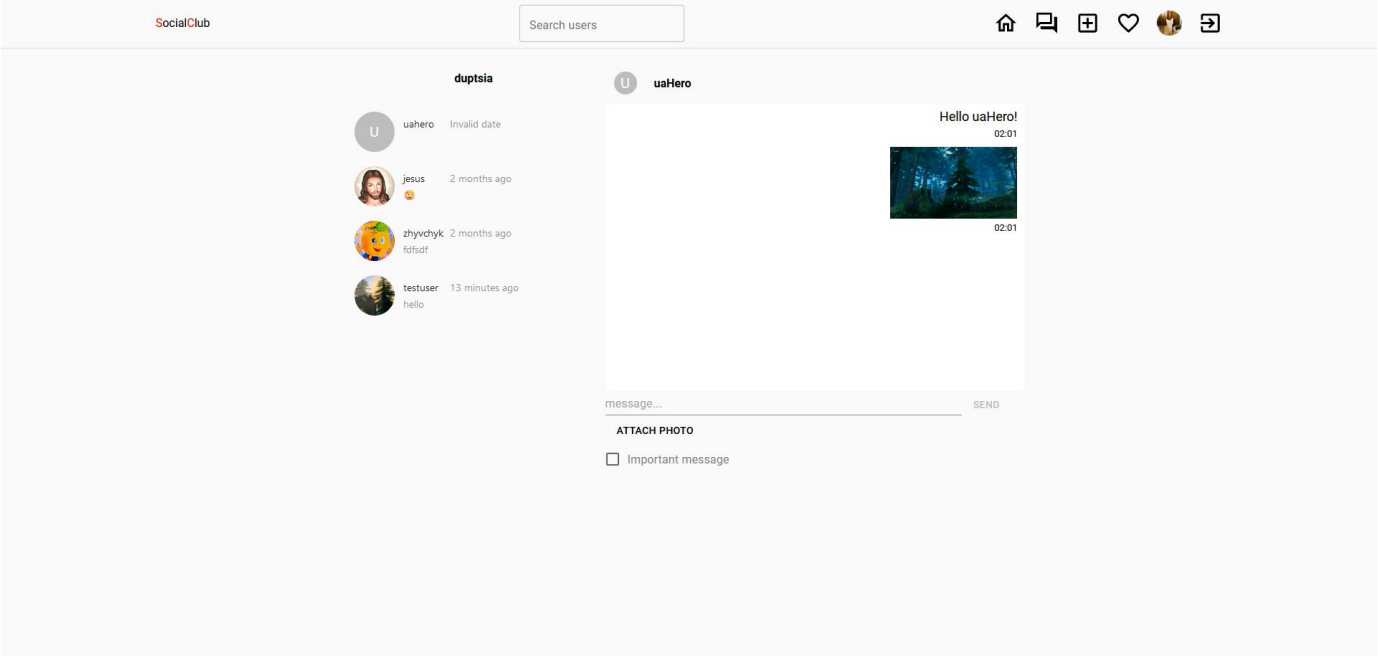


Рисунок Б.8 – Загальний вигляд інтерфейсного вікна сторінка чатів

Щоб надіслати важливе повідомлення, відмітьте що це повідомлення важливе у відповідному полі, перед надсиланням повідомлення (рис. Б.9).

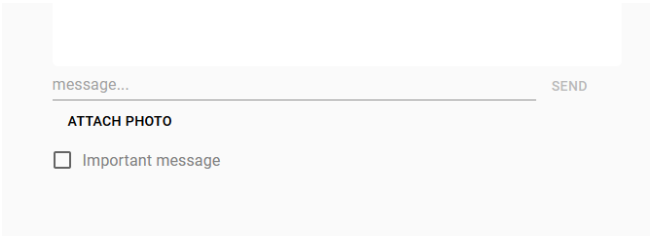


Рисунок Б.9 – Загальний вигляд відмітки важливого повідомлення

Щоб вийти з облікового запису, потрібно натиснути на клавішу, що знаходиться в правій частині навігаційної панелі (рис. Б.10), в наслідок чого, користувача перенаправить на сторінку входу в систему.

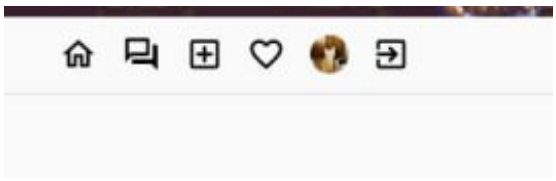


Рисунок Б.9 – Загальний вигляд кнопки виходу з облікового запису

ДОДАТОК В Лістинг програми

```
using EasyNetQ;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Mvc; using
Microsoft.IdentityModel.Tokens; using NLog; using
NLog.Web; using SocialApp.Infrastructure; using
SocialApp.Infrastructure.Interfaces; using
SocialApp.SharedKernel.Implementations; using
SocialApp.SharedKernel.Interfaces; using
SocialApp.Web.Hubs; using
SocialApp.Web.Middleware; using System.Text; using
EasyNetQ; using System.ComponentModel; using
SocialApp.Core.BackgroundWorkers; using
SocialApp.Core.Interfaces; using
SocialApp.SharedKernel.Authentication;
using SocialApp.Common.Validators.DtoValidators.UserDtoValidators;
using FluentValidation; using FluentValidation.AspNetCore;
using Microsoft.AspNetCore.Hosting;


var logger = NLog.LogManager.Setup().LoadConfigurationFromAppSettings().GetCurrentClassLogger();
logger.Debug("init main");


try {
   var builder = WebApplication.CreateBuilder(args);
builder.Services.AddResponseCaching();
   builder.Services.AddControllers(options =>
   {
      options.CacheProfiles.Add("Default14days",
new CacheProfile()
         {
            Duration = 1209600,
            Location = ResponseCacheLocation.Client
         });
   });
   builder.Services.AddEndpointsApiExplorer();
   builder.Services.AddSwaggerGen(options =>
   {
      options.AddSecurityDefinition(JwtBearerDefaults.AuthenticationScheme, new
Microsoft.OpenApi.Models.OpenApiSecurityScheme
      {
         Name = "Authorization",
         Type = Microsoft.OpenApi.Models.SecuritySchemeType.Http,
         Scheme = JwtBearerDefaults.AuthenticationScheme,
         BearerFormat = "JWT",
         In = Microsoft.OpenApi.Models.ParameterLocation.Header,
         Description = "JWT Authorization header using the Bearer scheme."
      });
      options.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement {
      {
         new Microsoft.OpenApi.Models.OpenApiSecurityScheme {
               Reference = new Microsoft.OpenApi.Models.OpenApiReference {

                  Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,
Id = JwtBearerDefaults.AuthenticationScheme
```

```
                }
            },
            new string[] {}
        }
    });
    });
    builder.Services.AddAuthentication(options => options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme)
.AddJwtBearer(options =>
    {
        options.RequireHttpsMetadata = false;
options.SaveToken = true;
        options.TokenValidationParameters = new TokenValidationParameters()
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidAudience = builder.Configuration[“Jwt:Audience”],
ValidIssuer = builder.Configuration[“Jwt:Issuer”],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration[“Jwt:Key”]))
        };


        options.Events = new JwtBearerEvents
        {
            OnMessageReceived = context =>
            {
                var accessToken = context.Request.Query[“access_token”];


                var path = context.HttpContext.Request.Path;
                if (!string.IsNullOrEmpty(accessToken) && path.StartsWithSegments(“/chat”))
                {
                    context.Token = accessToken;
                }


                return Task.CompletedTask;
            }
        };
})
    .AddScheme<TokenAuthenticationSchemeOptions, TokenAuthenticationSchemeHandler>(
“EmailVer”, options => {


    });



    builder.Services.AddCors(options =>
    {
        options.AddDefaultPolicy(builder =>
        {
            builder.WithOrigins(“http://localhost:3000”)
                .AllowAnyHeader()
                .AllowAnyMethod()
.AllowCredentials();
        });
    });


    builder.Services.AddControllers();
builder.Services.AddSignalR();
```

```
   builder.Services.AddAutoMapper(typeof(Program).Assembly);
builder.Services.AddTransient<IUnitOfWork,                    UnitOfWork>();
builder.Services.AddTransient<ISecretHasher,                  SecretHasher>();
builder.Services.AddTransient<IJwtClaimsService, JwtClaimsService>();
   var bus = RabbitHutch.CreateBus(Environment.GetEnvironmentVariable("RabbitMq"));
builder.Services.AddSingleton(bus);
   builder.Services.AddSingleton<IEmailSendingWorker, EmailSendingWorker>();
   builder.Services.AddHostedService(sp => sp.GetService<IEmailSendingWorker>() as EmailSendingWorker);
builder.Services.AddValidatorsFromAssemblyContaining<AddUserDtoValidator>();
builder.Services.AddFluentValidationAutoValidation().AddFluentValidationClientsideAdapters();
builder.Logging.ClearProviders();     builder.Host.UseNLog();




   var app = builder.Build();



   // Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
   {
      app.UseSwagger();
app.UseSwaggerUI();
      app.UseDeveloperExceptionPage();


   }
   if (app.Environment.IsStaging())
   {
      // code to be executed in staging environment


   }


   if (app.Environment.IsProduction())
   {
      // code to be executed in production environment


   }


   app.UseMiddleware<ExceptionMiddleware>();
app.UseCors();     app.UseResponseCaching();
app.UseRouting();
   //app.UseHttpsRedirection();
app.UseAuthentication();     app.UseAuthorization();
   app.UseEndpoints(endpoints =>
   {
      endpoints.MapHub<ChatHub>("/chat");
endpoints.MapControllers();
   });


   app.UseStaticFiles();
app.MapControllerRoute(
name: "index",           pattern:
"{url}",
         constraints: new { url = "^/*(\\w|\\/| )*$", },
```

```
        defaults: new { controller = “Home”, action = “Index” });
app.MapGet(“/”, async (HttpContext context) =>
   {
      var staticIndex = File.ReadAllText(“wwwroot/index.html”);
await context.Response.WriteAsync(staticIndex);
   });
   app.Run();


}
catch (Exception exception)
{
   logger.Error(exception, “Stopped program because of exception”);
throw; } finally {
   // Ensure to flush and stop internal timers/threads before application-exit (Avoid segmentation fault on Linux)
NLog.LogManager.Shutdown();
}



using SocialApp.Infrastructure.Interfaces.IRepositories; namespace

SocialApp.Infrastructure.Interfaces

{    public interface IUnitOfWork :

IDisposable

   {

      IPostRepository PostRepository { get; }

      IUserRepository UserRepository { get; }

      ICommentRepository CommentRepository { get; }

      ISubscribersRepository SubscribersRepository { get; }

IMessageRepository MessageRepository { get; }        void

Commit();        void RollBack();

   }

} using Dapper; using SocialApp.Core.Models; using

SocialApp.Core.Models.Dtos.UserDto; using

SocialApp.Core.Models.Pagination; using

SocialApp.Infrastructure.Interfaces.IRepositories; using

SocialApp.Infrastructure.Repositories.Base; using

System.Data; using static Dapper.SqlMapper;


namespace SocialApp.Infrastructure.Repositories

{    public class UserRepository : BaseRepository,

IUserRepository

   {

      public UserRepository(IDbTransaction transaction)

: base(transaction)
```

```
    {
    }

    public async Task<IEnumerable<User>> GetAllUsersAsync()
    {
        return await Connection.QueryAsync<User>(
            “SELECT UserId, Email, Name, Dob, CreationDate, LastSeen,” +
            “PhoneNumber, UserName, Password as EncryptedPassword FROM [User]”,
transaction: Transaction
        );

    }

    public async Task<IEnumerable<UserBasicInfoDto>> GetFilteredByUserNameUsersAsync(string userName)
    {
        return await Connection.QueryAsync<UserBasicInfoDto>(
            “SELECT TOP 10 UserId, Name, UserName FROM [User] WHERE UserName LIKE '%' + @UserName + '%'“,
param: new { UserName = userName },          transaction: Transaction
        );
    }

    public async Task<User> FindUserByUserIdAsync(int userId)
    {
        return await Connection.QueryFirstAsync<User>(
            @”SELECT UserId, Email, Name, Dob, CreationDate, LastSeen,
             PhoneNumber, UserName FROM [User] WHERE UserId = @UserId
            “,
        param: new { UserId = userId },
transaction: Transaction
        );
    }

    public async Task<User> GetCurrentUserAsync(int userId) // to remove profile pic
    {
        return await Connection.QueryFirstAsync<User>(
            “SELECT UserId, Email, Name, Dob, CreationDate,” +
```

```
            " PhoneNumber, UserName, BioProfile FROM [User] WHERE UserId = @UserId",
param: new { UserId = userId },

            transaction: Transaction
            );
        }


        public async Task<User> GetUserInfoAsync(int userId)
        {
            return await Connection.QueryFirstAsync<User>(
                "SELECT UserId, Email, Name," +
                " UserName FROM [User] WHERE UserId = @UserId",
param: new { UserId = userId },            transaction: Transaction
            );
        }


        public async Task<byte[]> GetUserProfilePicAsync(int userId)
        {
            return await Connection.QueryFirstOrDefaultAsync<byte[]>(
"SELECT ProfilePic FROM [User] WHERE UserId = @UserId",
param: new { UserId = userId },            transaction: Transaction
             );


        }


        public async Task<int> AddUserAsync(User entity)
        {
            ArgumentNullException.ThrowIfNull(entity);
return await Connection.ExecuteScalarAsync<int>(
            "INSERT INTO [User](Email, Name, UserName, CreationDate, Password) OUTPUT INSERTED.[UserId]
VALUES(@Email, @Name, @UserName, @CreationDate, @Password);",
            param: new { Email = entity.Email, Name = entity.Name, UserName = entity.UserName, CreationDate
= entity.CreationDate, Password = entity.EncryptedPassword },            transaction: Transaction
            );
        }


        public async Task<bool> UpdateUserAsync(User entity)
```

```
    {

        ArgumentNullException.ThrowIfNull(entity);
await Connection.ExecuteAsync(
        “UPDATE User SET Email = @Email, Name = @Name, PhoneNumber = @PhoneNumber, Dob = @Dob,” +
        “ ProfilePic = @ProfilePic WHERE UserId = @UserId”,
param: new
        {
            Email = entity.Email,
Name = entity.Name,
            PhoneNumber = entity.PhoneNumber,
            Dob = entity.Dob,
            ProfilePic = entity.ProfilePic
        },
        transaction: Transaction
        );
return true;

    }

    public async Task<bool> RemoveUserAsync(User entity)
    {
        ArgumentNullException.ThrowIfNull(entity);
await RemoveUserByUserIdAsync(entity.UserId);
return true;

    }

    public async Task<bool> RemoveUserByUserIdAsync(int userId)
    {

        await Connection.ExecuteAsync(
            “DELETE FROM User WHERE UserId = @UserId”,
param: new { UserId = userId },              transaction:
Transaction
```

```
        );
return true;
    }

    public async Task<User> FindUserByUserNameAsync(string userName)
    {
        ArgumentNullException.ThrowIfNull(userName);
return await Connection.QueryFirstAsync<User>(
            "SELECT * FROM [User] WHERE UserName = @UserName",
param: new { UserName = userName },              transaction: Transaction
        );
    }

    public async Task<UserLoginDto> GetPasswordAsync(string userName)
    {

        ArgumentNullException.ThrowIfNull(userName);          return
await Connection.QueryFirstOrDefaultAsync<UserLoginDto>(
            "SELECT Password as PasswordSec, UserId FROM [User] WHERE Username = @Username",
param: new { Username = userName },              transaction: Transaction
        );

    }

    public async Task<IEnumerable<UserMessangerInfoDto>> GetUsersInfoInMessanger(UserChatFilter userChatFilter)
    {
        ArgumentNullException.ThrowIfNull(userChatFilter);
return await Connection.QueryAsync<UserMessangerInfoDto>(
          @"SELECT m.MessageId, m.SendDate, m.MessageText, uc2.ChatId, uc2.UserId,
u.Name, u.UserName, u.ProfilePic
            FROM UserChat uc
            JOIN Chat c ON c.ChatId = uc.ChatId
            LEFT JOIN Message m ON m.ChatId = c.ChatId AND m.MessageId =
            (SELECT MAX(m2.MessageId) FROM Message m2 WHERE m2.ChatId = m.ChatId)
            LEFT JOIN UserChat uc2 ON uc2.ChatId = c.ChatId AND uc2.UserId <> @UserId
            LEFT JOIN [User] u ON u.UserId = uc2.UserId
```

```
            WHERE uc.UserId = @UserId
            ORDER BY m.MessageId OFFSET (@Skip) ROWS FETCH NEXT (@chatNumber) ROWS ONLY",
        param: new { UserId = userChatFilter.UserId, Skip = userChatFilter.Page * userChatFilter.CountNumber,
chatNumber = userChatFilter.CountNumber },
        transaction: Transaction
        );


    }


    public async Task<string> GetEmailByUserIdAsync(int userId)
    {
        return await Connection.ExecuteScalarAsync<string>(
            @"SELECT Email FROM [User] WHERE UserId = @UserId",
param: new { UserId = userId },          transaction: Transaction
        );


    }


    public async Task<bool> ConfirmUserEmail(int userId)
    {
        ArgumentNullException.ThrowIfNull(userId);
await Connection.ExecuteAsync(
        "UPDATE [User] SET EmailVerified = 1" +
        " WHERE UserId = @UserId",
param: new
        {
            UserId = userId
        },
        transaction: Transaction
        );
return true;
    }


    public async Task<bool> GetConfirmUserEmailStatus(int userId)
    {
```

```
        ArgumentNullException.ThrowIfNull(userId);          bool
isEmailConfirmed = await Connection.ExecuteScalarAsync<int>(
        "SELECT EmailVerified FROM [User]" +
        " WHERE UserId = @UserId",
param: new
        {
           UserId = userId
        },
        transaction: Transaction
        ) > 0;
        return isEmailConfirmed;
    }

    public async Task<bool> AddProfilePicAsync(AddProfilePicDto addProfilePicDto)
    {
        return await Connection.ExecuteScalarAsync<bool>(
            @"UPDATE [User] SET ProfilePic = (@ProfilePic) WHERE UserId = (@UserId)",
param: new { ProfilePic = addProfilePicDto.ProfilePic, UserId = addProfilePicDto.UserId },
transaction: Transaction
            );

    }
    public async Task<IEnumerable<UserBasicInfoDto>> GetRecomendedUsersAsync(int userId)
    {
        string sql = @"
                    WITH x AS (
                       SELECT FollowerId
                       FROM [Subscribers]
                       WHERE FollowingId = @userId
                       UNION
                       SELECT FollowingId
                       FROM [Subscribers]
                       WHERE FollowerId = @userId
                    ),
y AS (
                       SELECT FollowerId
                       FROM [Subscribers]
```

```
                    WHERE FollowingId IN (
                        SELECT FollowerId
                        FROM x
                    )
                    UNION
                    SELECT FollowingId
                    FROM [Subscribers]
                    WHERE FollowerId IN (
                        SELECT FollowerId
                        FROM x
)
                )
                SELECT UserId, Name, UserName
                FROM [User]
                WHERE UserId IN (
                    SELECT FollowerId
                    FROM y
                )
                AND UserId NOT IN (
                    SELECT FollowerId
                    FROM x
                )
                AND UserId <> @userId
            ";

        var result = await Connection.QueryAsync<UserBasicInfoDto>(sql, new { userId }, transaction: Transaction);
        return
result;

    }
  }
} using Microsoft.AspNetCore.Mvc; using
SocialApp.Core.Models; using
System.Security.Claims; using
System.Text; using
Microsoft.IdentityModel.Tokens; using
System.IdentityModel.Tokens.Jwt; using
```

```
Microsoft.AspNetCore.Authorization; using
AutoMapper;
using SocialApp.Infrastructure.Interfaces; using
SocialApp.SharedKernel.Interfaces; using
SocialApp.Core.Models.Dtos.UserDto; using
SocialApp.Core.Models.Pagination; using System.Net.Mime; using
EasyNetQ; using Microsoft.Net.Http.Headers; using FluentValidation;
using
SocialApp.Common.Validators.DtoValidators.UserDtoValidators;
using FluentValidation.Results;

namespace SocialApp.Web.Controllers
{
   [ApiController]
   [Route(“api/[controller]/[action]”)]     public
class UserController : BaseController
   {
      private readonly ISecretHasher _secretHasher;
private readonly IMapper _mapper;         private
readonly IBus _bus;

      public UserController(
IBus bus,
         IMapper mapper,
         ISecretHasher secretHasher,
         IConfiguration configuration,
         IUnitOfWork unitOfWork,
         ILogger<BaseController> logger
         ) : base(configuration, unitOfWork, logger)

      {
         _bus = bus;
         _mapper = mapper;
         _secretHasher = secretHasher;
      }
```

```
    [HttpGet]       public async
Task<IEnumerable<User>> GetAllUsers()
    {
        var results = await _unitOfWork.UserRepository.GetAllUsersAsync();
        _unitOfWork.Commit();
return results;
    }

    [HttpPost]
    public async Task<IEnumerable<UserBasicInfoDto>> GetFilteredByUserNameUsers([FromBody] string userName)
    {
        var results = await _unitOfWork.UserRepository.GetFilteredByUserNameUsersAsync(userName);
        _unitOfWork.Commit();
return results;
    }

    [HttpGet]       public async Task<User>
GetUserById(int userId)
    {
        var results = await _unitOfWork.UserRepository.FindUserByUserIdAsync(userId);
        _unitOfWork.Commit();
return results;
    }


    [HttpGet]       public async Task<UserInfoDto>
GetUserInfo([FromQuery] int userId)
    {
        var userprofileinfo = await _unitOfWork.UserRepository.GetUserInfoAsync(userId);          var
postInfo = await _unitOfWork.PostRepository.FindAllPostByPostDateOrByIdAsync(null, userId);
        _unitOfWork.Commit();          return new UserInfoDto() { userProfileInfo =
userprofileinfo, posts = postInfo };
    }
```

```
    [HttpPost]        public async Task<IActionResult> Login(UserLoginDto
userLoginDto) //Login        {

        var userInfo = await _unitOfWork.UserRepository.GetPasswordAsync(userLoginDto.Username);
if (userInfo == null)
        {
            return StatusCode(StatusCodes.Status404NotFound);


        }
        if (userInfo!=null && _secretHasher.Verify(userLoginDto.Password, userInfo.PasswordSec))
        {
            string token = await CreateUserJWTAcessToken(userLoginDto.Username, userInfo.UserId.Value);
            _unitOfWork.Commit();
return Ok(token);
        }
else
        {
            _unitOfWork.RollBack();                return
StatusCode(StatusCodes.Status401Unauthorized);
        }


    }


    [Authorize]
    [HttpGet]        public async Task<UserInfoDto>
GetCurrentLogedUser()
    {
        var currentUser = GetCurrentUser();
        var authorizedUserProfileInfo = await
_unitOfWork.UserRepository.GetCurrentUserAsync(currentUser.UserId.GetValueOrDefault());            var postInfo =
await _unitOfWork.PostRepository.FindAllPostByPostDateOrByIdAsync(null, currentUser.UserId);


        _unitOfWork.Commit();            return new UserInfoDto() { userProfileInfo =
authorizedUserProfileInfo, posts = postInfo };
    }
```

```
    [Authorize]    [HttpGet]    public async Task<User>
GetCurrentLogedUserProfileInfo()    {

        var currentUser = await GetThisUserAsync();

        var authorizedUserProfileInfo = await
_unitOfWork.UserRepository.GetCurrentUserAsync(currentUser.UserId.GetValueOrDefault());


        _unitOfWork.Commit();

return authorizedUserProfileInfo;
    }


    private UserLoginDto GetCurrentUser()
    {
        var identity = HttpContext.User.Identity as ClaimsIdentity;
         if (identity !=
null)
        {
            var userClaims = identity.Claims;


            return new UserLoginDto
            {
                Username = userClaims.FirstOrDefault(o => o.Type == "UserName")?.Value,
                UserId = Convert.ToInt32(userClaims.FirstOrDefault(o => o.Type == "UserId")?.Value),
            };
}          return
null;
    }
```

ДОДАТОК Г Графічна частина

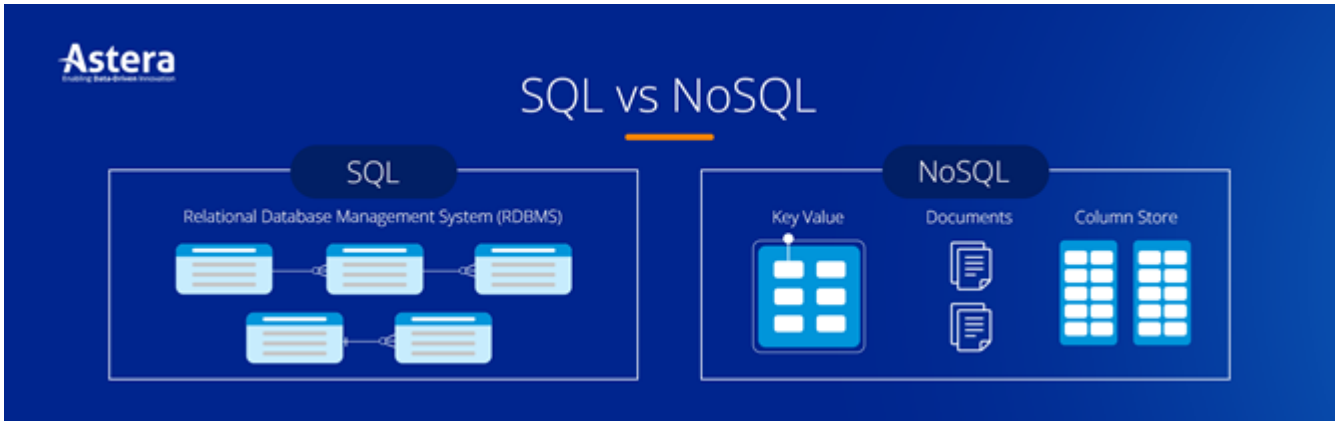


Рисунок Г.1 – Структури SQL та NoSQL баз даних

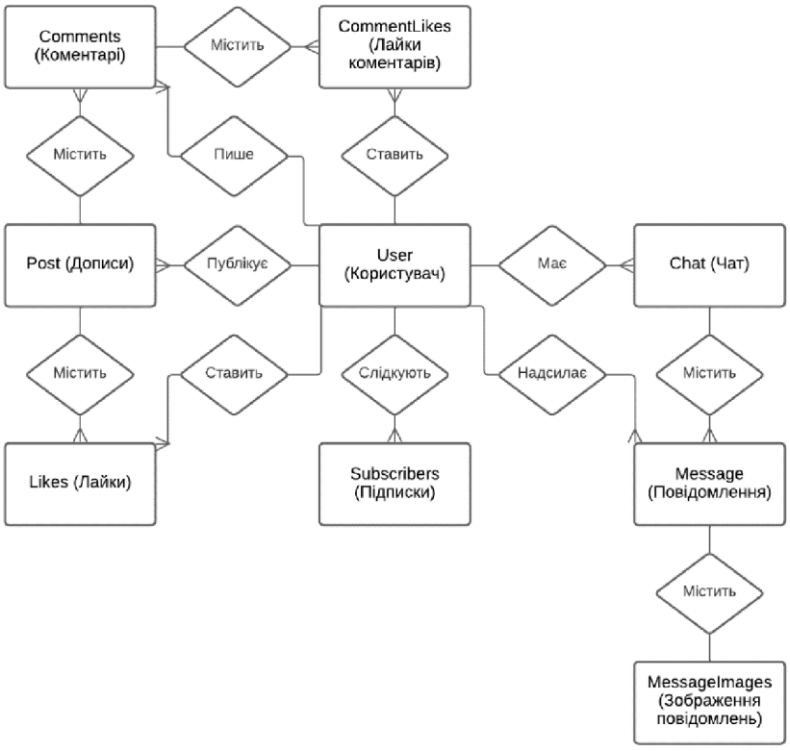


Рисунок Г.2 – Структурна схема ER-моделі бази даних соціальної мережі

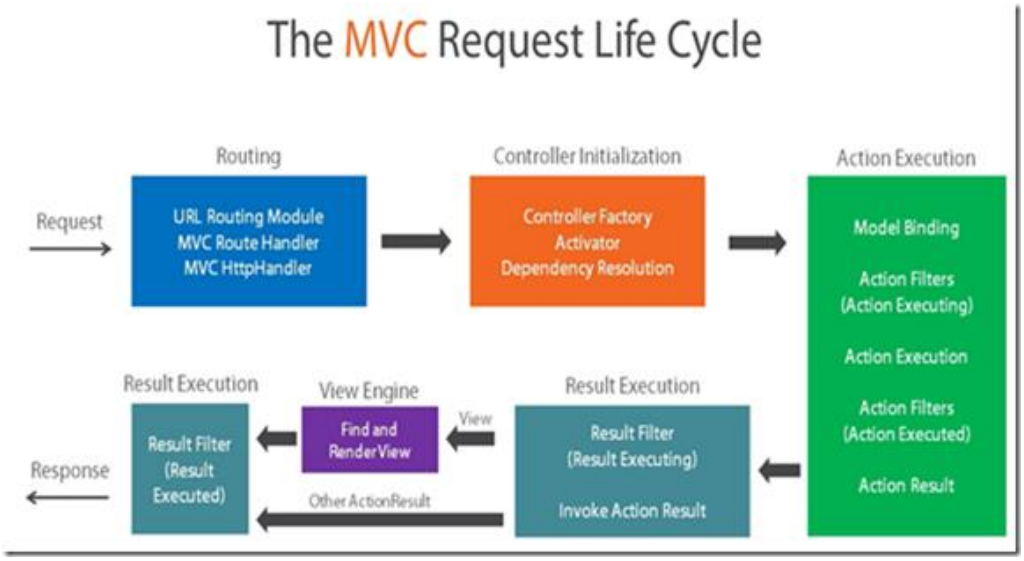


Рисунок Г.5 – Принципова схема життєвого циклу запиту в ASP.Net Core

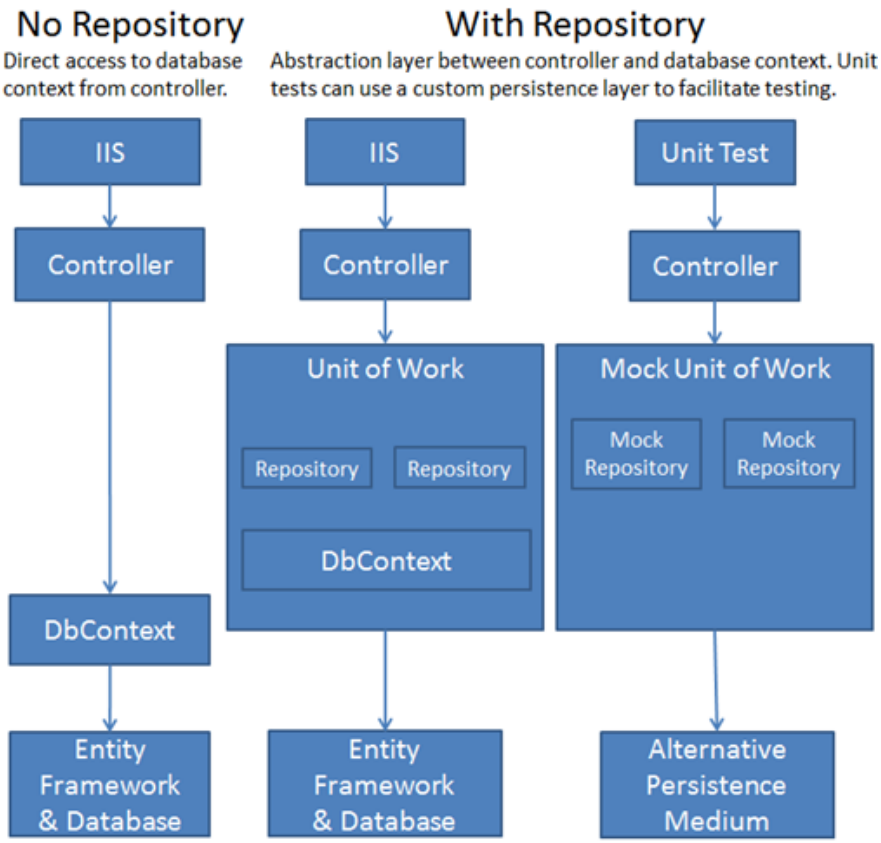


Рисунок Г.6 – Принципова схема додатків з використанням патерну UnitOfWork

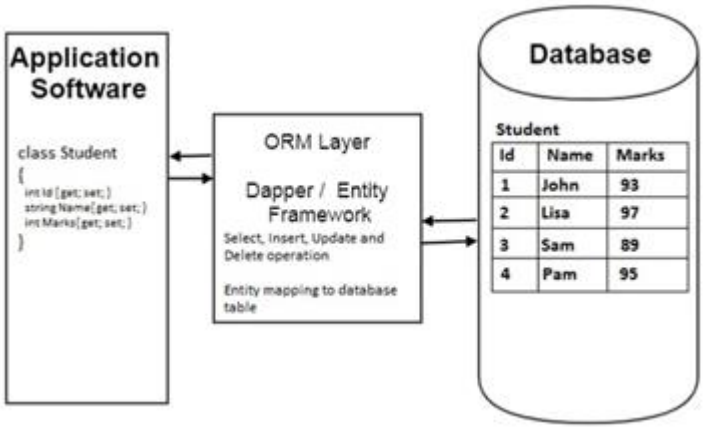


Рисунок Г.7 – Принципова схема ролі ORM в програмному додатку

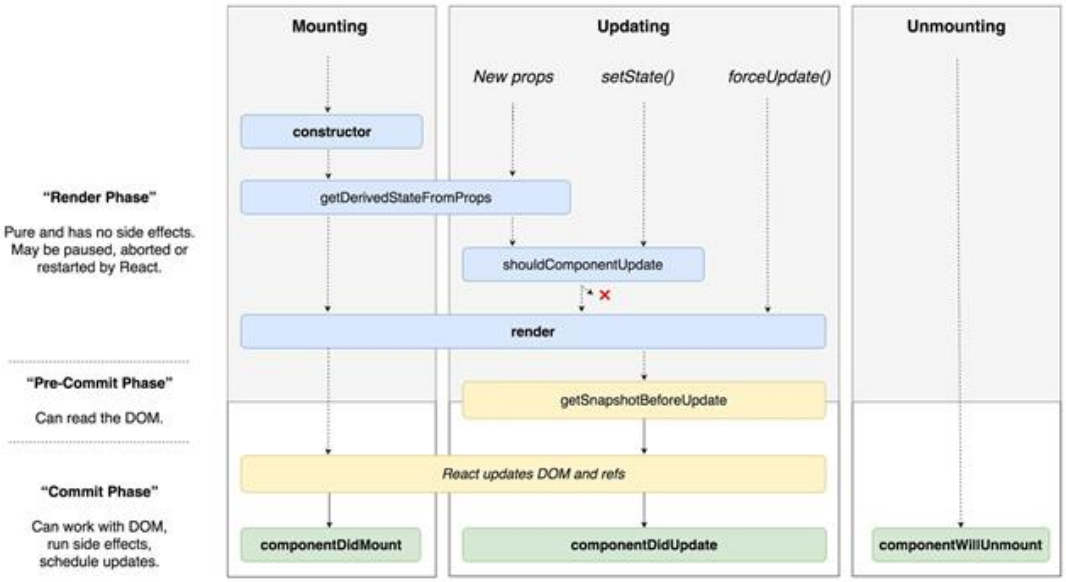


Рисунок Г.8 – Принципова схема алгоритму роботи бібліотеки ReactJS

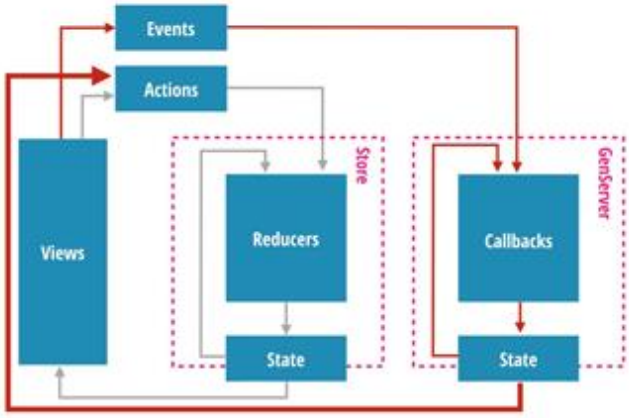


Рисунок Г.9 – Принципова схема алгоритму роботи бібліотеки Redux

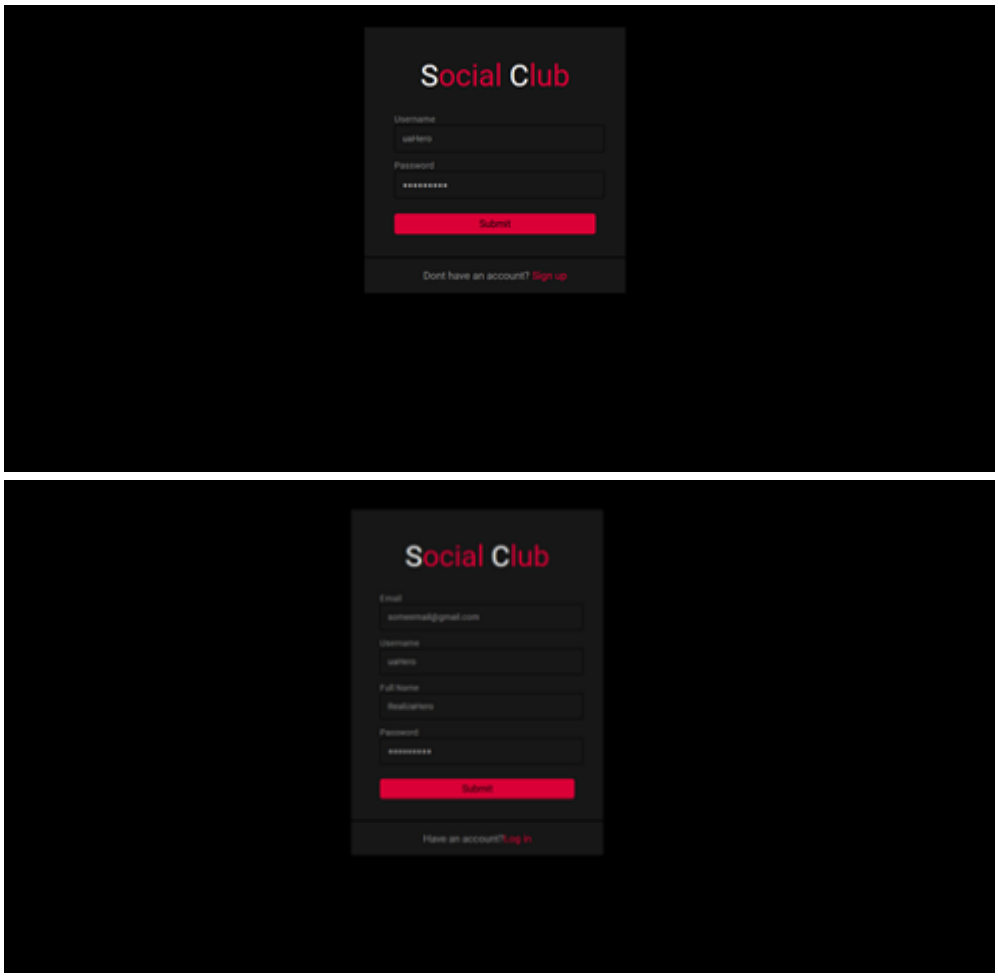
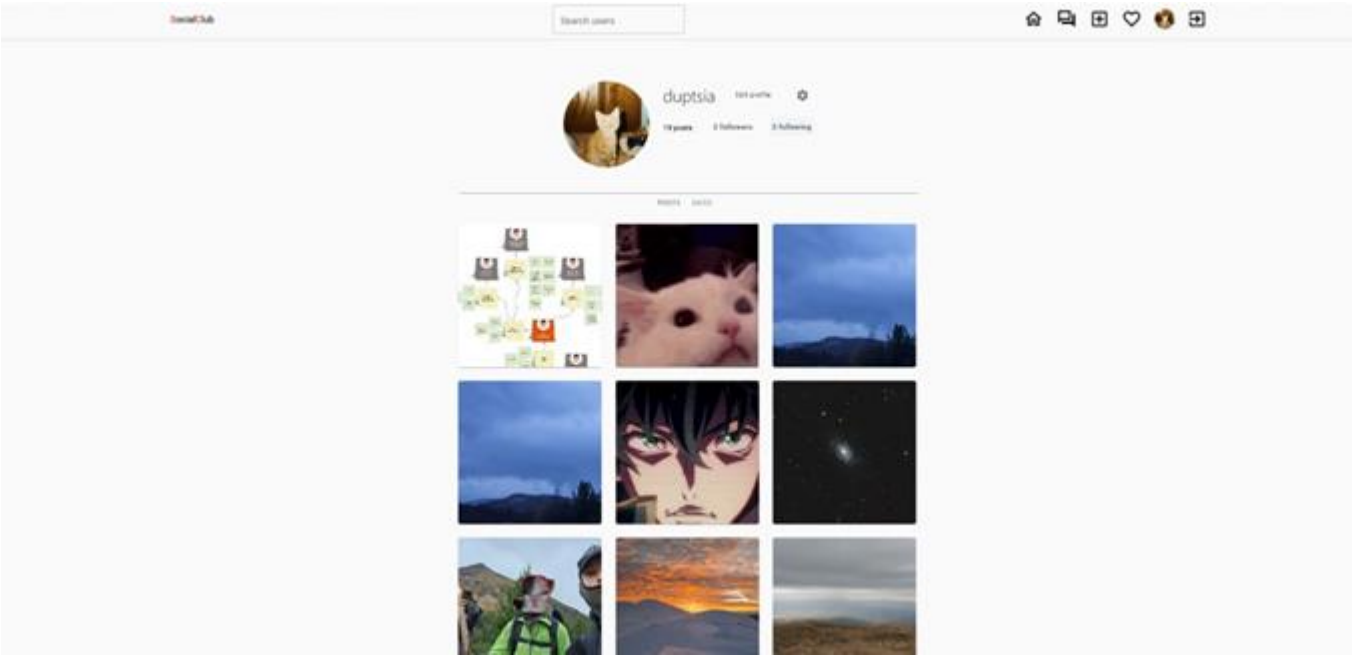


Рисунок Г.11 – Загальний вигляд інтерфейсних вікон програмного модуля



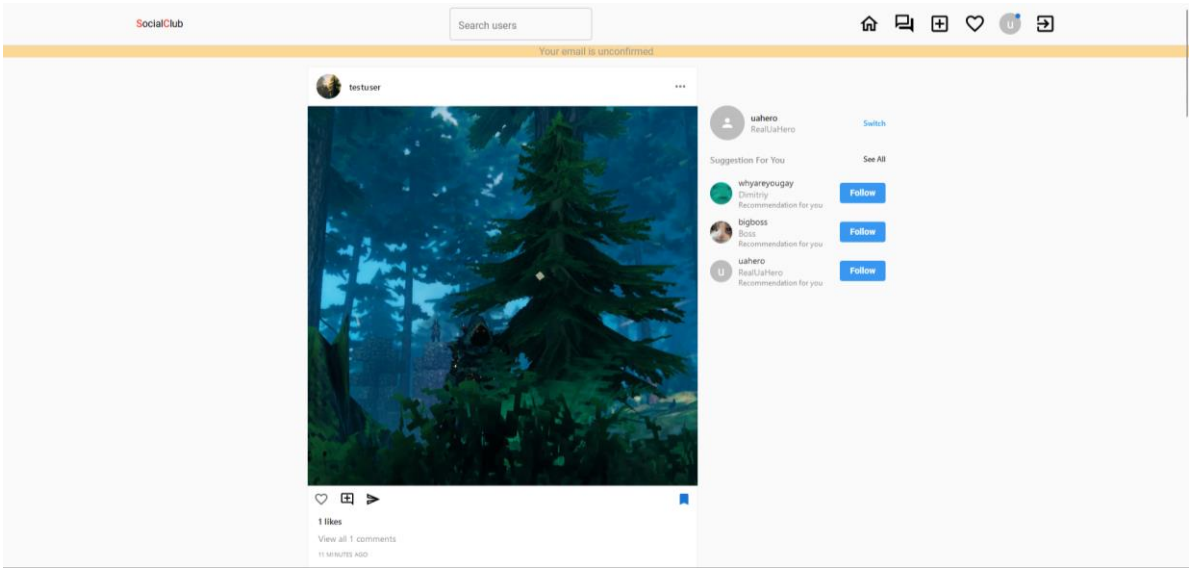


Рисунок Г.12 – Загальний вигляд інтерфейсних вікон програмного модуля