ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

**КУРСОВИЙ ПРОЕКТ**

з дисципліни

. «Вебтехнології» .

.(назва дисципліни)

на тему: Соціальна мережа Y

**КПІПЗ.2401188.01.07.ПЗ**

(шифр, назва)

Рівень вищої освіти             Перший (бакалаврський)

Галузь знань                      12 «Інформаційні технології»

Спеціальність     121 «Інженерія програмного забезпечення»

Освітня програма   «Інженерія програмного забезпечення»

Студента. I курсу, група . ІПЗс-24-1. \_\_\_\_\_\_\_\_\_\_\_ Максим ЛАПКО .

(шифр) (підпис) (ініціали, прізвище)

Керівник. канд. техн. наук, доцент . \_\_\_\_\_\_\_\_\_\_ Оксана ЯШИНА.

(посада, вчене звання, науковий ступінь) (підпис) (ініціали, прізвище)

Кількість балів \_\_\_\_\_\_\_

Оцінка за шкалою:

національною\_\_\_\_\_\_\_\_\_/ЄКТС\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

Хмельницький, 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

*Спеціальність.* 121 «Інженерія програмного забезпечення»

Шифр, назва

ЗАТВЕРДЖУЮ

Завідувач кафедри \_\_\_\_\_\_\_

Бедратюк Леонід Петрович

\_\_\_\_\_\_\_\_\_\_\_\_\_202\_\_\_ року

**ЗАВДАННЯ**

**НА КУРСОВИЙ ПРОЕКТ**

            Лапко М. О.

Прізвище, ім’я, по батькові студента

1. Тема проекту Соціальна мережа Y

керівник проекту Яшина Оксана Миколаївна канд. техн. наук, доцент

Прізвище, ім’я, по батькові науковий ступінь, вчене звання

2. Строк подання студентом проекту на кафедру . 1.12.2024 .

3. Вихідні дані до проекту: методичні вказівки до курсового проектування для студентів спеціальності 121 «Інженерія програмного забезпечення»» з курсу «Вебтехнології»

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) дослідження предметної області, проектування структури застосування, детальне проектування, розробка структури інформаційної системи, розробка модулів програми

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслень) Рисунки (0 шт.) Таблиці (0шт.)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Дата видачі завдання\_\_05.09.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

………………..........................................

.........................................................==…

**КАЛЕНДАРНИЙ ПЛАН**

виконання курсового проектування

з дисципліни «Вебтехнології»

на І-й семестр 2024-2025 навчального року

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів курсового проекту | Термін виконання етапів проекту  (тиждень) | Відмітка про виконання |
| *1* | *Одержання теми курсового проекту та завдання* | *1* |  |
| *2* | *Вступ* | *2* |  |
| *3* | *Особливості та обмеження реалізації проекту* | *3* |  |
| *4* | *Аналіз існуючих програмних продуктів* | *4* |  |
| *5* | *Проектування зовнішнього вигляду сайту та навігації* | *5* |  |
| *6* | *Проектування БД* | *6* |  |
| *7* | *Вибір засобів для розробки сайту* | *7* |  |
| *8* | *Розробка структури проекту* | *8* |  |
| *9* | *Підключення та розробка БД* | *9* |  |
| *10* | *Розробка зовнішнього вигляду сайту* | *10-11* |  |
| *11* | *Розробка програмних модулів* | *12-13* |  |
| *12* | *Тестування проекту* | *14* |  |
| *13* | *Опрацювання висновків та переліку літератури* | *15* |  |
| *14* | *Оформлення пояснювальної записки* | *16* |  |
| *15* | *Захист* | *17* |  |

#### Керівник проекту Яшина О.М. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

#### Затверджено:

#### Зав. кафедри ІПЗ Бедратюк Л.П. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис)

**АНОТАЦІЯ**

Курсовий проект «Соціальна мережа Y».

Автор роботи: Лапко М.О.

Керівник роботи: Яшина О.М.

Обсяг – 0 с., 0 рис., 0 табл., 0 додатки, 0 джерел.

Мета курсового проекту: розробити та реалізувати соціальну мережу з базовими функціональними можливостями для комунікації так взаємодії між користувачами.

У курсовому проекті виконана розробка веб-сервісу із застосуванням мови програмування Go для серверної частини, з використанням фреймворків REST та gRPC для комунікації між мікросервісами, а також MySQL для зберігання даних. Інтерфейс реалізовано на React для забезпечення інтерактивної взаємодії користувачів.

Дата Підпис

ЗМІСТ

*Змн.*

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*6*

КПІПЗ.2401188.01.07.ПЗ

*Розроб.*

*Лапко М.О.*

*Перевір.*

*Яшина О.М.*

*Реценз.*

*Н. Контр.*

*Затверд.*

Соціальна мережа Y

*Літ.*

*Акрушів*

69

*ХНУ. ІПЗс-24-1*

Перелік скорочень 7

ВСТУП 8

1 Дослідження предметної області 10

1.1 Характеристика функціональної структури предметної області 10

1.2 Аналіз останніх публікацій, досліджень та існуючих рішень 12

1.3 Аналіз вимог до програмного забезпечення. 18

1.4 Розробка технічного завдання 22

2. Проектування та визначення та визначення методів і засобів розробки 23

2.1 Проектування архітектури ПЗ 23

2.2 Проектування бази даних 27

2.3 Модель проектування 37

2.4 Вибір засобів розробки 41

3. Програмна реалізація проекту 44

3.1 Структура та функціональне призначення модулів програмного забезпечення, їх взаємозв’язок. 44

3.2 Розробка програмного продукту 47

3.3 Інструкція користувача 55

3.4 Вимоги до програмно-технічного середовища функціонування ПЗ 56

4 Тестування ПЗ 57

4.1 Методика тестування ПЗ 57

4.2 Результати тестування ПЗ 59

ВИСНОВКИ 62

Перелік джерел посилань 64

Додаток А 66

## ПЕРЕЛІК СКОРОЧЕНЬ

|  |  |  |
| --- | --- | --- |
| БД | – | База даних |
| SQL | **–** | Structured Query Language |
| ORM | – | Object-Relational Mapping |
| СКБД | – | Система керування базами даних |
| ПЗ | – | Програмне забезпечення |
| IDE | – | Integrated Development Environment |
| UML | **–** | Unified Modeling Language |
| Backend | – | Серверна частина застосунку |
| Frontend | – | Клієнтська частина застосунку |
| API | – | Application Programming Interface |
| HTTP | – | HyperText Transfer Protocol |
| gRPC | – | Remote Procedure Call |
| REST | – | Representational State Transfer |
| JSON | – | JavaScript Object Notation |
| JWT | – | JSON Web Token |

## ВСТУП

## У сучасному світі, де цифрові технології та соціальні платформи відіграють важливу роль у повсякденному житті, існує постійна потреба у нових інструментах для комунікації, взаємодії та обміну інформацією. Саме соціальні мережі стали важливими майданчиками для обміну думками, ідеями, контентом та досвідом. З урахуванням цих тенденцій виникає необхідність створення інноваційної соціальної платформи, яка забезпечить користувачам зручні й ефективні можливості для взаємодії.

## Курсовий проект спрямований на розробку соціальної мережі "Y" — веб-сервісу, що дозволить користувачам ділитися власними думками, створювати контент, обмінюватися ідеями та досвідом. Основною метою є створення простору, де користувачі зможуть легко знаходити спільні інтереси, підтримувати зв’язок, ділитися контентом та будувати спільноту.

## Об'єктом дослідження є процес розробки веб-сервісу для соціальної мережі, який включає технічні, дизайнерські та функціональні аспекти створення такої платформи. Це передбачає розробку надійної серверної архітектури, розробку клієнтського інтерфейсу для зручної взаємодії з користувачем, забезпечення безпеки даних та захисту приватної інформації користувачів.

## Предметом дослідження є ключові аспекти розробки соціальної мережі, зокрема технічні рішення, архітектурні особливості, дизайн інтерфейсу, а також вплив соціальної платформи на користувачів та її роль у підтримці спільноти.

## Завдання дослідження включають наступне: створення ефективної архітектури платформи, розробку зручного користувацького інтерфейсу, реалізацію функцій реєстрації та управління обліковими записами, забезпечення безпеки та конфіденційності, розробку механізмів пошуку, фільтрації та взаємодії з контентом, а також тестування та оптимізацію продукту.

## Зміст роботи полягає в аналізі сучасних трендів у сфері соціальних мереж, огляді інструментів для розробки веб-додатків, проектуванні бази даних для зберігання інформації про користувачів та контент, а також у створенні користувацького інтерфейсу й інтерактивних функцій.

## Мета роботи – розробити та реалізувати соціальну мережу з базовими функціональними можливостями для комунікації так взаємодії між користувачами.

## Практична значимість роботи полягає у створенні зручного інструменту для спілкування, що відповідає потребам сучасних користувачів та сприяє формуванню активної спільноти.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика функціональної структури предметної області

У сучасному інформаційному суспільстві соціальні мережі стали невід'ємною частиною комунікації та обміну інформацією між людьми. У зв'язку із зростаючою популярністю соціальних платформ виникає потреба створити соціальну мережу, яка забезпечить користувачам зручний і безпечний спосіб ділитися контентом, обговорювати теми, що цікавлять, і формувати спільноти за інтересами. Щоб детально зрозуміти вимоги до такої платформи, необхідно провести аналіз предметної області, визначити основні поняття, розглянути невирішені питання та проблеми, враховуючи потреби користувачів.

Предметна область соціальної мережі охоплює не тільки механізми публікації та обміну контентом, а й взаємодію між користувачами через коментарі, оцінки (лайки), повідомлення та інші соціальні функції. Аналізуючи структуру соціальної мережі, важливо враховувати типи контенту, який створюватиметься на платформі, а також особливості взаємодії між користувачами, зокрема можливості для персоналізації та модерації контенту.

Функціональні аспекти охоплюють процеси публікації постів, пошуку, фільтрації контенту, а також механізми для забезпечення безпеки та конфіденційності. Необхідно також забезпечити зручний інтерфейс, який полегшить користувачам навігацію на сайті, та впровадити надійні методи захисту даних для запобігання несанкціонованому доступу.

Загальний аналіз цих аспектів дозволить створити більш глибоке розуміння предметної області та визначити ключові вимоги до розробки соціальної мережі, що сприятиме успішній реалізації проєкту. Таким чином, було виділено наступні етапи та об'єкти для автоматизації в соціальній мережі "Y":

* автоматизація процесу створення та публікації постів;
* автоматизація пошуку та фільтрації контенту за темами або ключовими словами;
* автоматизація системи сповіщень для інформування користувачів про оновлення.

Необхідність автоматизації цих функцій виникає з потреби спростити взаємодію користувачів із платформою та забезпечити ефективний механізм взаємодії з контентом.

Для досягнення мети проєкту визначено наступні завдання:

* аналіз потреб користувачів у сфері обміну інформацією та створення контенту;
* проєктування структури соціальної мережі та її функціональності;
* розробка системи реєстрації та управління обліковими записами;
* створення бази даних для зберігання даних про користувачів та контенту;
* реалізація програмних модулів для публікації та пошуку контенту;
* розробка користувацького інтерфейсу;
* створення технічних вимог та інструкції користувача.

Очікувані результати включають:

* розроблену соціальну мережу, яка дозволить користувачам публікувати та переглядати контент;
* впроваджену систему автентифікації для захисту облікових записів;
* створену базу даних із необхідною інформацією для оптимальної роботи платформи;
* підготовлену інструкцію користувача, що пояснює правила користування соціальною мережею.

1.2 Аналіз останніх публікацій, досліджень та існуючих рішень

Аналіз існуючих рішень у контексті дослідження предметної області та постановки задач є важливим етапом, адже це дозволяє не лише зрозуміти поточний стан галузі та визначити основні тенденції, але й знайти нові ідеї для нашого проекту.

Процес аналізу можна умовно розділити на декілька етапів:

1. Визначення цілей та завдань аналізу.
2. Вибір джерел інформації.
3. Збір та аналіз інформації.
4. Формулювання висновків.

Перш за все було сформульовано цілі та завдання аналізу, з урахуванням інформації про проект та предметну область. У результаті аналізу мали бути визначені наявні рішення, що потенційно задовольняють потреби нашого проекту, а також їхні переваги й недоліки. Це допоможе знайти найкращі аспекти для інтеграції у нашому рішенні, а також виявити недоліки, які слід уникати.

Джерелом інформації стали популярні платформи, що вже охоплюють частини нашої предметної області.

Соціальна мережа **X**, раніше відома як Twitter, є платформою мікроблогінгу, яка дозволяє користувачам ділитися короткими повідомленнями, фото та відео, а також взаємодіяти через лайки, репости та коментарі. X створена для швидкого обміну думками та новинами в реальному часі, що робить її однією з найвпливовіших платформ для обговорення поточних подій та висловлення поглядів.

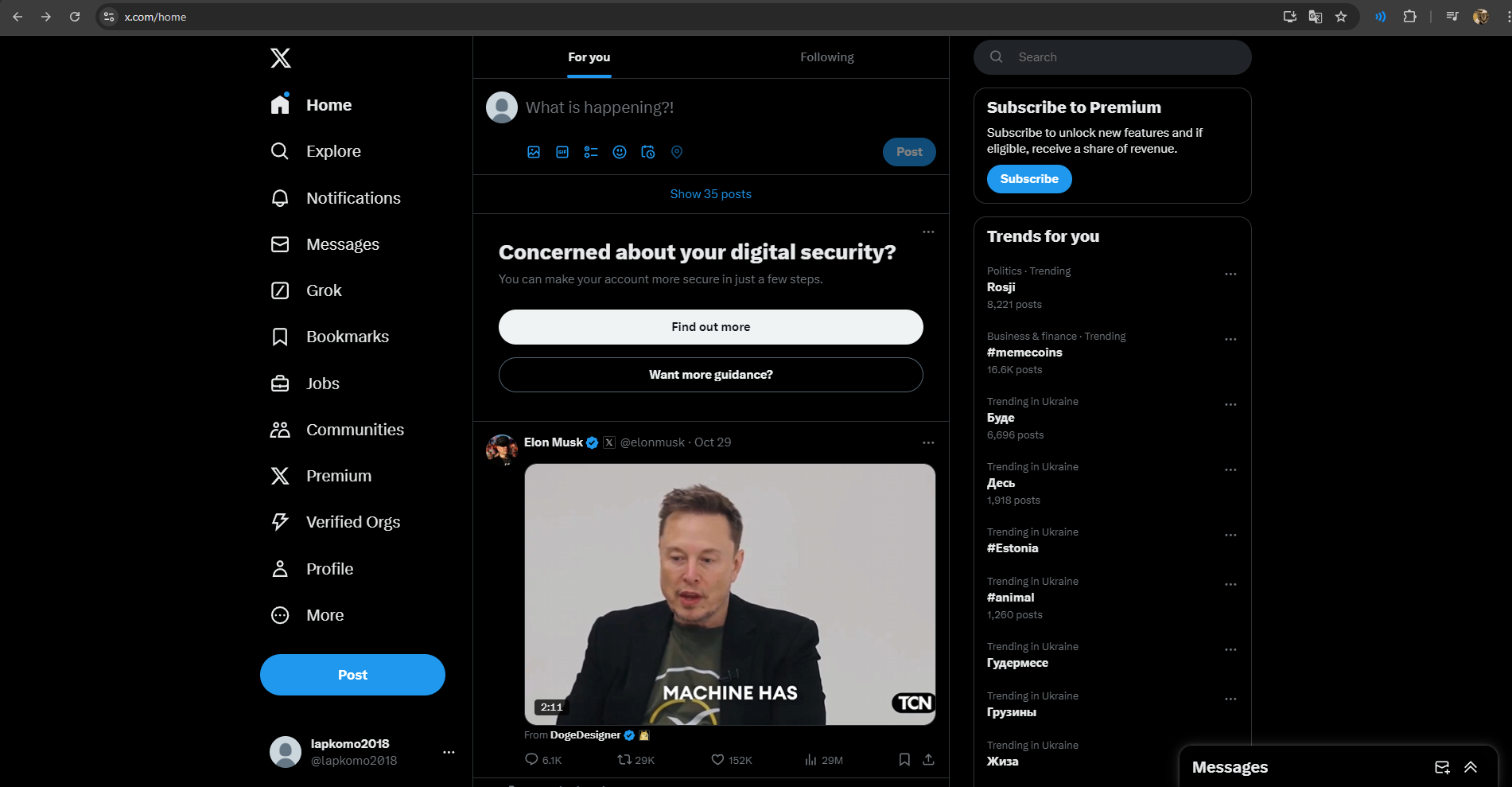


Рисунок 1.2.1 —Головна сторінка «X» [1]

Функціональні особливості:

* **Стрічка новин**: основна сторінка, де відображаються твіти від користувачів, на яких підписаний користувач, а також рекомендовані записи.
* **Короткі повідомлення (твіти)**: повідомлення обмежені певною кількістю символів, що робить їх компактними й змістовними.
* **Репости та лайки**: користувачі можуть ділитися твітами інших (репост) і виражати схвалення через лайки.
* **Хештеги та тренди**: система хештегів дозволяє користувачам об’єднувати дописи за певними темами, а розділ трендів відображає найпопулярніші теми.
* **Прямі повідомлення (Direct Messages)**: можливість для користувачів обмінюватися приватними повідомленнями.
* **Функціональність Spaces**: кімнати для аудіочатів, що дозволяють проводити живі розмови з аудиторією.
* **Рекламні твіти та аналітика**: X також підтримує просування контенту та детальну аналітику для бізнесів і рекламодавців.

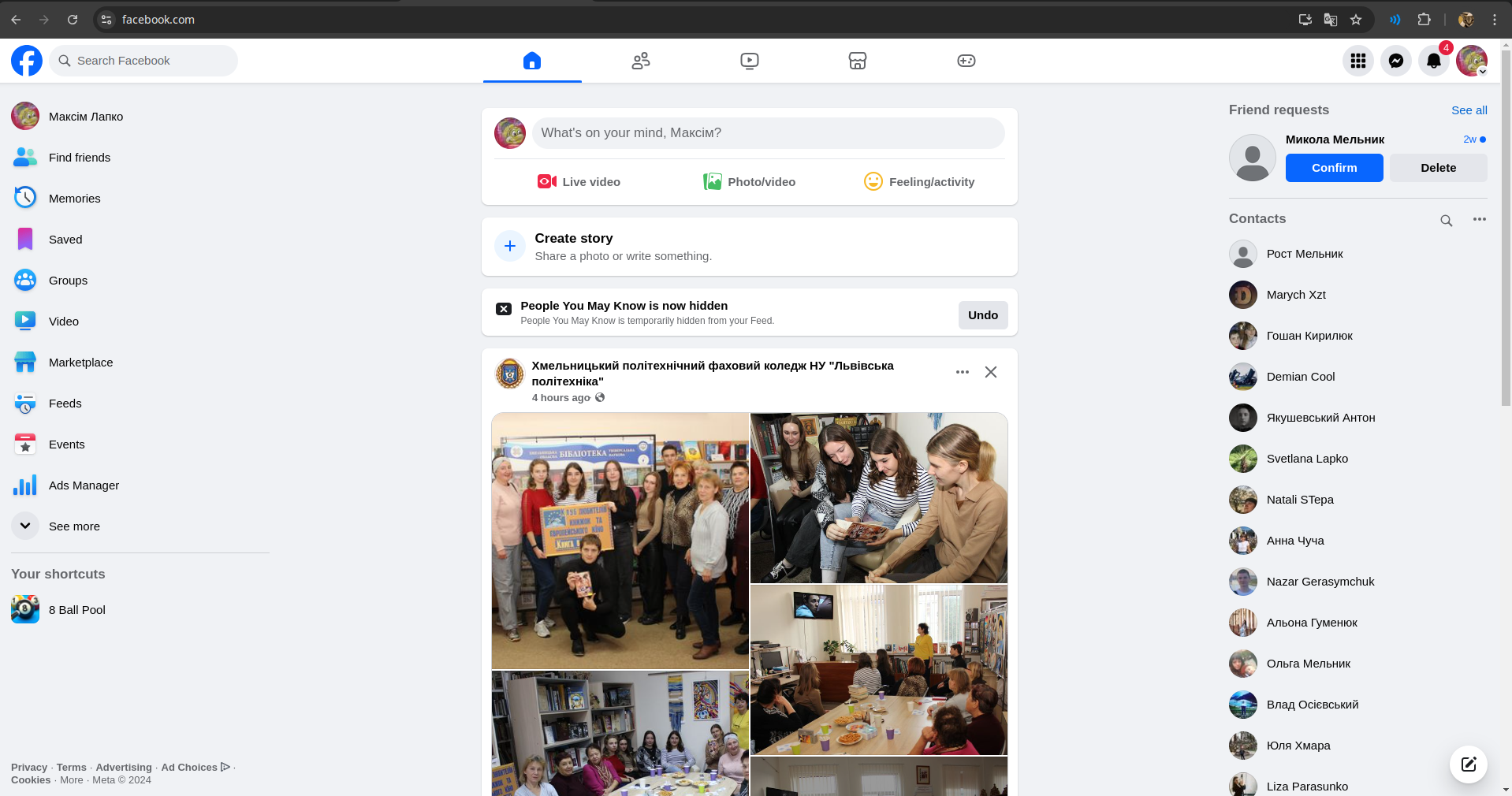
Переваги:

* **Швидкий обмін інформацією**: обмеження на кількість символів сприяє коротким і змістовним повідомленням, що підходить для обговорення новин.
* **Трендовий контент**: хештеги та розділ трендів дозволяють швидко дізнатися про найактуальніші теми та події.
* **Простота використання**: мінімалістичний дизайн інтерфейсу робить платформу легкою для навігації.
* **Широкі можливості для бізнесу**: рекламні твіти та аналітика дають компаніям змогу ефективно взаємодіяти з аудиторією.

Недоліки:

* **Виклики в управлінні контентом**: значний обсяг інформації ускладнює модерування, що часто призводить до проблем із небажаним контентом.
* **Обмежена функціональність редагування**: відсутність або обмеженість у редагуванні твітів.
* **Високий рівень інформаційного шуму**: через короткі пости і відсутність чіткої структури користувачам може бути складно знаходити необхідну інформацію.

Facebook [2] — одна з найстаріших і найпопулярніших соціальних мереж у світі. Запущена в 2004 році, ця платформа пропонує широкий набір функцій, який дозволяє користувачам спілкуватися, обмінюватися інформацією та розважатися.

Рисунок 1.2.2 —Головна сторінка «Facebook» [2]

**Функціональні особливості:**

* **Стрічка новин**: основна сторінка, де користувачі бачать оновлення від друзів та груп, на які вони підписані.
* **Профіль користувача**: кожен користувач має персональну сторінку профілю, що відображає його публікації, особисту інформацію, список друзів і фотографії.
* **Групи та сторінки**: платформа підтримує різні типи взаємодій через сторінки для компаній, брендів або груп для спільних інтересів.
* **Обмін повідомленнями та відеодзвінки**: Facebook Messenger є основним інструментом для приватного обміну повідомленнями та дзвінків.
* **Реклама та аналітика**: компанії можуть використовувати Facebook для просування продукції та послуг з детальною аналітикою.

**Переваги**:

* Можливість масштабування для обробки великої кількості користувачів.
* Потужний рекламний механізм для монетизації.
* Збалансована екосистема для особистого спілкування, групових взаємодій і комерційних цілей.

**Недоліки**:

* Часті скандали через політику конфіденційності та безпеки даних.
* Складність підтримки великого коду, особливо з урахуванням безперервного зростання функціональності.

Instagram [3], запущений у 2010 році, є соціальною мережею, орієнтованою на візуальний контент — фотографії та відео. Вона спочатку була розроблена для мобільних пристроїв і фокусується на простому, інтуїтивному дизайні.

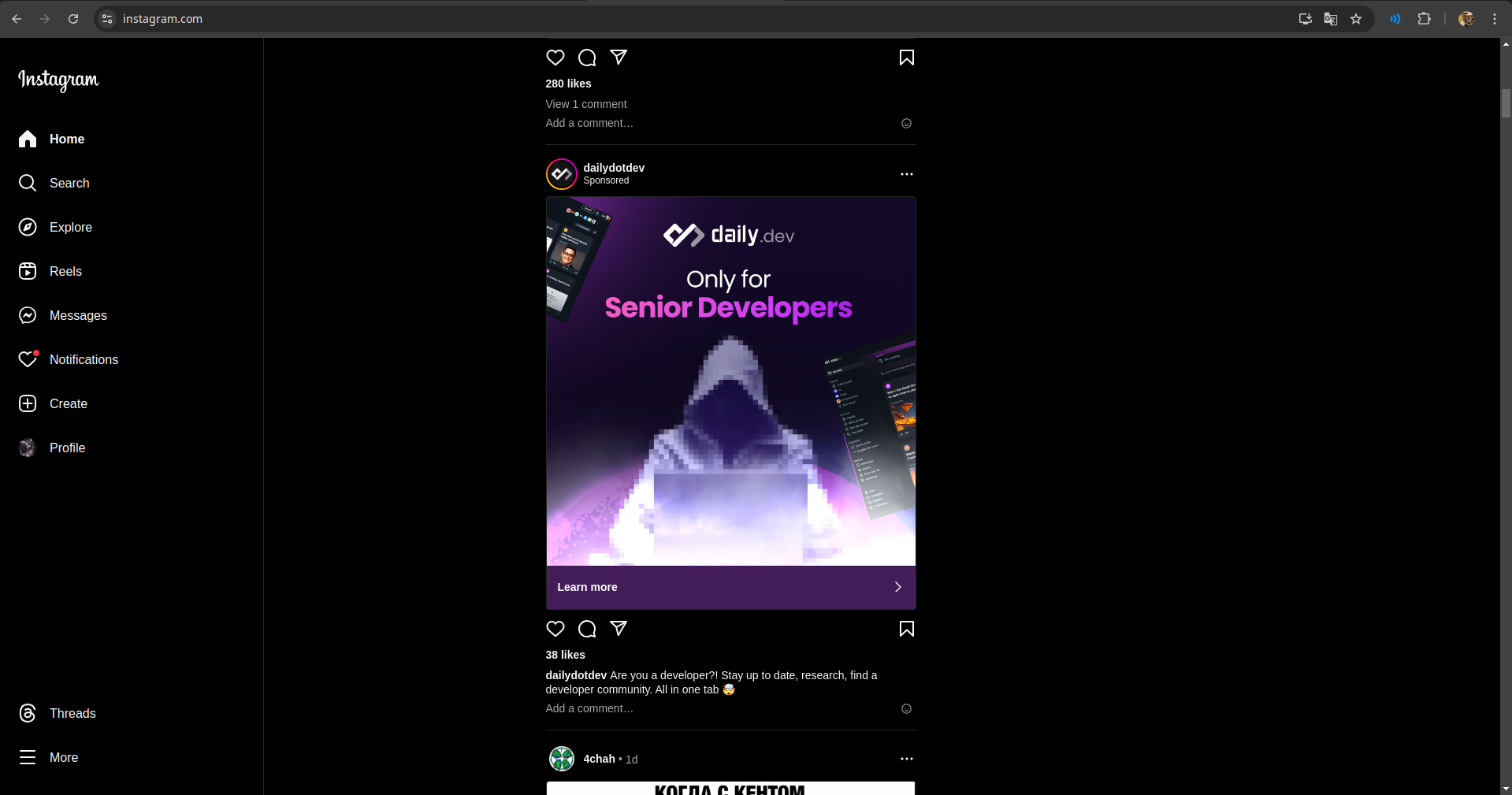


Рисунок 1.2.3 —Головна сторінка «Instagram» [3]

**Функціональні особливості**:

* **Стрічка новин**: основна стрічка контенту з публікаціями користувачів, на яких вони підписані.
* **Сторіз (Stories)**: тимчасові публікації, що зникають через 24 години.
* **Рекомендації**: механізм, який пропонує користувачам новий контент на основі їхніх інтересів.
* **Direct Messages**: вбудована функція для приватного обміну повідомленнями.
* **Інтерактивність**: можливість лайкати, коментувати та ділитися контентом.
* **Фільтри та редагування**: велика кількість інструментів для редагування зображень і відео.

**Переваги**:

* Орієнтованість на візуальний контент, що дозволяє залучати користувачів через простий і привабливий інтерфейс.
* Широкі можливості для контент-маркетингу та залучення брендів.
* Зручний інтерфейс для мобільних пристроїв.

**Недоліки**:

* Обмежені можливості редагування постів після публікації.
* Складнощі з монетизацією контенту без використання зовнішніх посилань.
* Питання конфіденційності через агресивне використання особистих даних для рекомендацій.

Отже, проведений аналіз показав, що деякі елементи функціоналу X, а також загальні підходи Facebook та Instagram до інтерфейсу і організації пошуку, можуть бути корисними для нашого проєкту. Однак, для реалізації особливих цілей, таких як рівноцінний обмін контентом, специфічне відстеження користувачів і чітка модерація через систему репортів, потрібно застосувати індивідуальний підхід. Це дозволить створити унікальне рішення, яке найкраще відповідає концепції проєкту і потребам користувачів.

1.3 Аналіз вимог до програмного забезпечення та розробка технічного завдання

Виконання аналізу вимог до програмного забезпечення та розробка технічного завдання є невід'ємною складовою процесу розробки програмних продуктів. У сучасному інформаційному суспільстві, де програмне забезпечення використовується в різних сферах діяльності, важливо забезпечити якість та відповідність програмного продукту вимогам його користувачів. Чітке розуміння вимог є основою для успішного проектування та розробки програмного забезпечення.

Аналіз вимог до програмного забезпечення передбачає збір, аналіз та документування потреб користувачів. Цей процес вимагає взаємодії зі зацікавленими сторонами, включаючи замовників, кінцевих користувачів та фахівців з інформаційних технологій. Він допомагає зрозуміти функціональні та нефункціональні вимоги до програмного продукту, його обмеження та контекст використання. Результатом аналізу вимог є документ, який описує всі виявлені вимоги та їх пріоритети.

Для веб-сервісу "Social Network Y" було проведено формування бізнес-моделі, яка визначає стратегію реалізації проекту та його комерційний потенціал. Бізнес-модель окреслює цілі проекту, його мету і способи досягнення успіху, а також розкриває ключові аспекти, такі як цільова аудиторія, цінова політика, канали розповсюдження та прибуткові потоки.

Перш за все, було визначено потенційних користувачів системи. Для веб-сервісу "Social Network Y" потенційними користувачами є люди, які цікавляться спілкуванням, обміном інформацією та взаємодією з іншими користувачами. Цільова аудиторія сервісу охоплює людей віком від 18 років, які проживають в Україні та інших країнах світу.

Цінність сервісу полягає в забезпеченні зручного та доступного способу взаємодії між користувачами, включаючи можливість створення дописів, коментування, лайкання та обміну повідомленнями. Основними каналами залучення користувачів до "Social Network Y" є веб-сайт та мобільні додатки, а також можливість співпраці з партнерами для реклами.

Основними ресурсами, якими володітиме веб-сервіс, будуть: інфраструктура проекту (веб-сайт, база даних, сервер), програмне забезпечення та контент сервісу. Ключовими потенційними партнерами можуть бути: маркетологи, агенції з реклами та сервіси доставки.

Для визначення користувацьких вимог було вирішено провести опитування з метою збору інформації про потреби та побажання потенційних користувачів веб-сервісу "Social Network Y". Опитування допомогло визначити, які функції та можливості повинні бути включені в сервіс, щоб він був максимально корисним і зручним для користувачів. Для проведення опитування було використано онлайн-інструменти, такі як Google Forms, а також соціальні мережі та месенджери.

Основні питання для проведення опитування:

1. Як часто ви користуєтеся соціальними мережами?
2. Які функції соціальних мереж ви вважаєте найбільш важливими?
3. Чи є у вас досвід використання соціальних мереж для обміну інформацією або комунікації?
4. Які можливості ви хотіли б бачити у соціальній мережі?
5. Чи зацікавлені ви у функціях, що дозволяють адміністрування та модерування контенту?

За результатами проведення опитування були отримані наступні відповіді від потенційних користувачів системи.

Як часто ви користуєтеся соціальними мережами?

* Щодня. Я заходжу в соціальні мережі кожен день, щоб перевірити новини та оновлення від друзів.
* Кілька разів на тиждень. Я заходжу, коли маю вільний час або щось цікаве.
* Рідко. Я не використовую соціальні мережі часто, тільки коли це необхідно.
* Ніколи. Я не користуюсь соціальними мережами.

Які функції соціальних мереж ви вважаєте найбільш важливими?

* Створення та публікація контенту. Можливість ділитися своїми думками та фото.
* Лайки та коментарі. Можливість взаємодії з контентом інших користувачів.
* Інструменти для адміністрування. Можливість керувати контентом і взаємодією в групах.
* Групи та спільноти. Можливість приєднуватися до інтересів та обговорювати теми з однодумцями.
* Обмін повідомленнями. Можливість спілкуватися з друзями в реальному часі.

Чи є у вас досвід використання соціальних мереж для обміну інформацією або комунікації?

* Так. Я активно використовую соціальні мережі для спілкування з друзями та обміну інформацією.
* Частково. Я іноді використовую їх для обміну інформацією, але не часто.
* Ні. Я не використовую соціальні мережі для обміну інформацією.

Які можливості ви хотіли б бачити у соціальній мережі?

* Більш детальна система фільтрації контенту. Я хотів би мати можливість налаштувати, що я бачу у своїй стрічці.
* Інструменти для створення подій. Я хотів би організовувати та запрошувати друзів на події.
* Розширений пошук. Я хотів би легше знаходити користувачів, пости та групи за інтересами.
* Можливість редагувати свої пости. Я хотів би мати можливість змінювати свої публікації після їх створення.
* Анонімність. Я хотів би мати можливість використовувати платформу анонімно для певних обговорень.

Чи зацікавлені ви у функціях, що дозволяють адміністрування та модерування контенту?

* Так. Я вважаю, що адміністрування і модерування важливі для підтримки порядку в соціальних мережах.
* Можливо. Це цікаво, але я не знаю, наскільки це важливо для мене.
* Ні. Я не цікавлюсь адмініструванням та модеруванням контенту.

Відповіді на ці питання можуть допомогти зрозуміти потреби користувачів у вашій соціальній мережі "Social Network Y". Наприклад, якщо більшість респондентів використовують соціальні мережі щодня, це вказує на високий попит на функціональність платформи. Визначення важливих функцій, таких як публікація та коментування контенту, дозволить зосередитися на їх розробці. А також відгуки про адміністрування та модерування контенту можуть вказати на потребу у функціях для забезпечення безпеки та комфорту користувачів.

1.4 Розробка технічного завдання

Із технічним завданням можна ознайомитися у Додатку A

2. ПРОЕКТУВАННЯ ТА ВИЗНАЧЕННЯ МЕТОДІВ І ЗАСОБІВ РОЗРОБКИ

2.1 Проектування архітектури ПЗ

## У рамках проектування архітектури соціальної мережі визначення архітектури програмного забезпечення є критично важливим етапом. Загальна архітектура системи формує структуру програмного забезпечення, що включає розподіл компонентів, їхню взаємодію та загальний механізм роботи системи. Вона також містить ключові рішення щодо організації програмної системи, зокрема вибір структурних елементів і їх інтерфейсів, а також поведінку, що визначена у взаємодії між цими елементами. Однією з основних функцій архітектури програмного забезпечення є подолання труднощів, що виникають при проектуванні і розробці складних програмних систем.

## Тому перед початком розробки необхідно чітко визначити правильну архітектуру програмного забезпечення, щоб забезпечити надійність і фундаментальну організацію системи. Спочатку було розглянуто основні види архітектури, кожна з яких має свої переваги та недоліки, що частково визначають можливості масштабування і стабільної роботи системи. В рамках інженерії програмного забезпечення існує багато архітектурних підходів, але в даному проекті розглянуто основні з них.

## Архітектура клієнт-сервер (Client-Server Architecture). Ця архітектура базується на принципі, що користувач (клієнт) взаємодіє з сервером, надсилаючи запити і отримуючи відповідь з необхідними даними чи послугою. Ця модель може функціонувати як на одному пристрої, так і через велику мережу.

## Переваги:

## висока масштабованість.

## поліпшена безпека.

## висока надійність.

## Недоліки:

## залежність від доступності сервера.

## можливість перевантаження сервера.

## Монолітна архітектура (Monolithic Architecture). Весь додаток складається з одного великого блоку, де всі компоненти і залежності інтегровані разом, що полегшує розробку та розгортання.

## Переваги:

## простота розробки.

## легкість в утриманні та відлагодженні.

## одинична точка розгортання.

## Недоліки:

## обмежена масштабованість;

## складність при внесенні змін;

## можливість витрат на невикористані ресурси.

## Мікросервісна архітектура (Microservices Architecture). Додаток розбивається на невеликі, незалежні сервіси, кожен з яких відповідає за конкретну функціональність, що полегшує масштабованість та розвиток додатку.

## Переваги:

## висока масштабованість.

## висока гнучкість.

## швидкість розробки.

## незалежність сервісів.

## ефективне використання ресурсів.

## Недоліки:

## складність управління комунікацією між сервісами.

## складність керування даними.

## складність відлагодження та моніторингу.

## Аналізуючи переваги та недоліки розглянутих архітектур програмного забезпечення, можна зробити висновок, що для розроблюваної соціальної мережі найкраще підходить мікросервісна архітектура. Цей тип архітектури дозволяє поділити систему на невеликі, незалежні сервіси, кожен з яких відповідає за певну функціональність. Мікросервісна архітектура надає низку переваг у контексті створення веб-сервісу, який забезпечує обмін інформацією та взаємодію між користувачами.

## По-перше, ця архітектура дозволяє підвищити масштабованість і гнучкість, оскільки кожен мікросервіс можна розробляти, розгортати та масштабувати незалежно від інших. Це також спрощує підтримку і розвиток кожної окремої частини системи, що полегшує внесення змін і впровадження нових функцій.

## Крім того, розподіл функцій між окремими мікросервісами дозволяє оптимізувати ресурси, а також підвищити надійність системи — відмова одного сервісу не впливає на роботу інших компонентів. Кожен сервіс має свою базу даних, що полегшує управління даними і забезпечує автономність даних кожного мікросервісу.

## У контексті обраної архітектури важливо визначити ключові терміни:

## Мікросервіс: Незалежний сервіс, який виконує конкретну функцію в системі, обробляє запити від клієнтів або інших мікросервісів, і повертає відповідь.

## Клієнт: Користувацька частина системи, яка надсилає запити до мікросервісів та обробляє отримані відповіді.

## Запит: Спосіб, яким клієнт передає інструкції або дані мікросервісу для отримання відповіді.

## Відповідь: Інформація або ресурс, який мікросервіс повертає клієнту чи іншому мікросервісу у відповідь на його запит.

## Протокол зв'язку: Специфікація, що визначає правила обміну даними між клієнтом і мікросервісами (наприклад, REST або gRPC).

## Загалом основну ідею мікросервісної архітектури можна представити у вигляді рисунка (див. Рисунок 2.1.1).

## 

## Рисунок 2.1.1 — Концепція мікросервісної архітектури [4]

## Таким чином, було розглянуто різні архітектурні підходи для проекту соціальної мережі, і обрано мікросервісну архітектуру через численні переваги, що роблять її найбільш відповідною для розробки сервісу.

2.2 Проектування бази даних

## На основі обраної мікросервісної архітектури та вимог до системи було розроблено проект бази даних. Для цього використано модель «сутність-зв’язок», що допомогла визначити сутності, атрибути та зв’язки між ними, а також зрозуміти, як дані пов’язані між собою. Це дозволило спроектувати базу даних, яка буде ефективною та зручною для використання в умовах розподіленої архітектури.

## Діаграма ER (Entity-Relationship) — це графічне представлення сутностей та їхніх зв’язків один з одним у базі даних. Вона є концептуальним інструментом, який використовується в розробці бази даних для опису даних та зв’язків, важливих для організації даних. ER-діаграма відображає зв’язки між сутностями, які визначають, як дані (атрибути) в цих сутностях взаємопов’язані. На ER-діаграмі зв’язки зображуються лініями, що з’єднують сутності.

## Основні типи зв’язків між сутностями включають:

## Один-до-одного (1:1): один екземпляр однієї сутності може бути пов’язаний з одним екземпляром іншої сутності.

## Один-до-багатьох (1): один екземпляр однієї сутності може бути пов’язаний з багатьма екземплярами іншої сутності.

## Багато-до-одного (N:1): багато екземплярів однієї сутності можуть бути пов’язані з одним екземпляром іншої сутності.

## Багато-до-багатьох (M): багато екземплярів однієї сутності можуть бути пов’язані з багатьма екземплярами іншої сутності.

## Загалом модель сутність-зв’язок може бути представлена в різних нотаціях, зокрема в нотації Бахмана, Петера Чена, IDEF1X, Вороняча лапка тощо. У контексті обраного проекту для розробки бази даних використано нотацію IDEF1X, оскільки вона відповідає як особистим уподобанням, так і об’єктивним вимогам проекту щодо обробки та зберігання даних для забезпечення коректної роботи сервісу.

## IDEF1X — це метод проектування баз даних, який дозволяє створити концептуальну схему даних через ідентифікацію сутностей та зв’язків між ними. Концептуальна схема є абстрактним зображенням даних, незалежним від конкретної програми або фізичного представлення. Моделі IDEF1X складаються з таких основних компонентів:

## Сутності: представляють реальні об’єкти або концепції.

## Атрибути: визначають характеристики об’єкта.

## Відносини: визначають зв’язки між об’єктами.

## 

## Рисунок 2.2.1 — Компоненти нотації IDEF1X [5]

## Для формування початкових таблиць було розглянуто, як система мала функціонувати в загальному сценарії, що дало уявлення про сутності та атрибути, необхідні для проектування. В основі процесу лежали сформовані функціональні вимоги до соціальної мережі. Сценарій, на основі якого здійснювалося проектування:

## Користувач реєструється на платформі, вказуючи особисті дані.

## Користувач авторизується, щоб отримати доступ до основних функцій.

## Користувач створює та публікує нові дописи.

## Користувач коментує дописи інших користувачів.

## Користувач ставить позначки «подобається» на дописи.

## Користувач редагує власний профіль (аватар, нікнейм, електронну адресу).

## Адміністратор перевіряє звіти про порушення та модерує дописи.

## Далі було виділено імена об'єктів у кожному пункті сценарію. Вони стали кандидатами на сутності для подальшого етапу проектування бази даних. На основі загального сценарію роботи системи було визначено шість основних сутностей, які зберігаються в різних базах даних для підвищення масштабованості та незалежності мікросервісів:

## Користувачі — зберігається в БД auth service.

## Профіль — зберігається в БД user service.

## Пости — зберігається в БД post service.

## Коментарі — зберігається в БД post service.

## Лайки — зберігається в БД post service.

## Репорти — зберігається в БД report service.

## Кожен із зазначених мікросервісів використовує окрему базу даних для управління своїми специфічними даними, що забезпечує автономність та розширюваність кожного з компонентів. Після цього сценарій був розбитий на окремі компоненти для виявлення потенційних атрибути всіх сутностей. Опис цих атрибутів наведено за допомогою неформального тексту. Цей неформальний опис атрибутів дає загальне розуміння того, які дані можуть бути використані для моделювання цих сутностей.

## Користувач реєструється в системі:

## При реєстрації користувач вказує username, електронну адресу, пароль.

## Система присвоює користувачеві унікальний ідентифікатор.

## Дані зберігаються у БД auth service та user service.

## Користувач створює та публікує пост:

## Користувач вказує заголовок, текст, прикріплює зображення, відео.

## Пости отримують унікальний ідентифікатор.

## Дані про пости зберігаються у БД post service.

## Користувач коментує дописи інших користувачів:

## Користувач вводить текст коментаря до допису.

## Коментарі отримують унікальний ідентифікатор та прив’язку до поста і користувача.

## Дані про коментарі зберігаються у БД post service.

## Користувач ставить позначки «подобається»:

## При взаємодії з постом зберігається ID користувача, ID поста.

## Система зберігає інформацію про кожну позначку «подобається» у БД post service.

## Користувач редагує профіль:

## Користувач змінює аватар, нікнейм, електронну пошту, username.

## Дані про профіль зберігаються в окремій БД user service та обновляється в auth service якщо змінюєтся почта чи username.

## Адміністратор перевіряє репорти про порушення:

## Система зберігає репорти, які користувачі подають на інші пости.

## Звіти містять ID користувача, ID поста, причину, відповідь та статус.

## Дані про звіти зберігаються у БД report service.

## Таблиці з бд кожного сервісу приведено нижче.

## Таблиці в auth service db

## Таблиця 2.2.1. Таблиця «users» для сутності «User»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| email | longtext | YES |  |
| username | longtext | YES |  |
| password | longtext | YES |  |
| is\_activated | tinyint(1) | YES |  |
| created\_at | datetime(3) | YES |  |
| updated\_at | datetime(3) | YES |  |
| deleted\_at | datetime(3) | YES |  |

## Таблиця 2.2.2. Таблиця «activation\_tokens» для сутності «Activation Token»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| user\_id | bigint unsigned | NO | auto\_increment |
| token | varchar(191) | YES |  |

## Таблиця 2.2.3. Таблиця «refresh\_tokens» для сутності «Refresh Token»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| token | varchar(191) | YES |  |

## Таблиці в user service db

## Таблиця 2.2.4. Таблиця «users» для сутності «User»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| email | longtext | YES |  |
| username | longtext | YES |  |
| nickname | longtext | YES |  |
| avatar | longtext | YES |  |
| role | bigint unsigned | YES |  |
| created\_at | datetime(3) | YES |  |
| updated\_at | datetime(3) | YES |  |
| deleted\_at | datetime(3) | YES |  |

## Таблиця 2.2.5. Таблиця «bans» для сутності «Ban»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra | foreign\_key |
| id | bigint unsigned | NO | auto\_increment |  |
| user\_id | bigint unsigned | YES |  | users(id) |
| banned\_by | bigint unsigned | YES |  |  |
| ban\_reason | longtext | YES |  |  |
| banned\_at | datetime(3) | YES |  |  |
| duration | bigint | YES |  |  |
| expired\_at | datetime(3) | YES |  |  |
| unbanned\_by | bigint unsigned | YES |  |  |
| unban\_reason | longtext | YES |  |  |
| unbanned\_at | datetime(3) | YES |  |  |
| created\_at | datetime(3) | YES |  |  |
| updated\_at | datetime(3) | YES |  |  |
| deleted\_at | datetime(3) | YES |  |  |

## Сутності в post service db

## Таблиця 2.2.6. Таблиця «posts» для сутності «Post»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| user\_id | bigint unsigned | YES |  |
| title | longtext | YES |  |
| content | longtext | YES |  |
| image\_urls | longblob | YES |  |
| video\_urls | longblob | YES |  |
| posted\_at | datetime(3) | YES |  |
| edited\_by | bigint unsigned | YES |  |
| created\_at | datetime(3) | YES |  |
| updated\_at | datetime(3) | YES |  |
| deleted\_at | datetime(3) | YES |  |

## Таблиця 2.2.7. Таблиця «likes» для сутності «Like»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| user\_id | bigint unsigned | YES |  |
| post\_id | bigint unsigned | YES |  |
| created\_at | datetime(3) | YES |  |
| deleted\_at | datetime(3) | YES |  |

## Таблиця 2.2.8. Таблиця «comments» для сутності «Comment»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| user\_id | bigint unsigned | YES |  |
| post\_id | bigint unsigned | YES |  |
| content | longtext | YES |  |
| edited\_by | bigint unsigned | YES |  |
| created\_at | datetime(3) | YES |  |
| updated\_at | datetime(3) | YES |  |
| deleted\_at | datetime(3) | YES |  |

## Таблиця 2.2.9. Таблиця «tags» для сутності «Tag»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| name | longtext | YES |  |

## Таблиця 2.2.10. Таблиця «post\_tags»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | foreign\_key |
| tag\_id | bigint unsigned | NO | tags(id) |
| post\_id | bigint unsigned | NO | posts(id) |

## Таблиці в report service db

## Таблиця 2.2.11. Таблиця «reports» для сутності «Report»

|  |  |  |  |
| --- | --- | --- | --- |
| column\_name | data\_type | is\_nullable | extra |
| id | bigint unsigned | NO | auto\_increment |
| user\_id | bigint unsigned | YES |  |
| post\_id | bigint unsigned | YES |  |
| reason | longtext | YES |  |
| admin\_id | bigint unsigned | YES |  |
| answer | longtext |  |  |
| status | longtext |  |  |
| created\_at | datetime(3) | YES |  |
| updated\_at | datetime(3) | YES |  |
| deleted\_at | datetime(3) | YES |  |

## Після визначення основних сутностей і їх атрибутів було прийнято рішення встановити зв’язки між ними, з урахуванням вимог до функціональності системи.

## Спочатку розглянемо зв’язок між сутностями "User" у сервісах auth і user. Взаємозв’язок між цими сутностями формується через унікальні ідентифікатори користувачів та основні атрибути для аутентифікації та профілювання, такі як email, username, password (у auth сервісі) і додаткові атрибути, як nickname, avatar (у user сервісі). Основний зв’язок – «один користувач має один профіль», що вказує на зв’язок типу 1:1.

## Далі, аналізуються сутності у post сервісі, зокрема "User" і "Post". Зв'язок між ними формулюється так: «Один користувач може створювати безліч постів, і кожен пост належить одному користувачу». Це вказує на зв’язок «один до багатьох» 1:N.

## Наступний зв’язок — між сутностями "Post" та "Tag", де кожен пост може бути позначений кількома тегами, а кожен тег може бути пов’язаний із кількома постами. Це вказує на зв’язок «багато до багатьох» M:N. Для нормалізації бази даних було створено проміжну таблицю post\_tags, яка містить ідентифікатори поста та тегу, що дозволяє зберігати зв’язок між ними як два окремих зв’язки «один до багатьох» M:N між Post та Tag.

## Сутність "Like" також пов’язана з "User" та "Post". Охарактеризувати цей зв'язок можна так: «Один користувач може вподобати безліч постів, і кожен пост може мати безліч вподобань від різних користувачів». Це знову є зв’язок «багато до багатьох», і для цього передбачено таблицю Like.

## У сервісі report основний зв’язок — між сутностями "Report" і "Post". Один користувач може надіслати декілька скарг на різні пости, тому тут знову маємо зв’язок «один до багатьох» 1:N. Крім того, кожен звіт може бути оброблений різними адміністраторами, що вказує на зв’язок «один до багатьох» між сутностями "Report" і "User".

## Таким чином, усі сутності були інтегровані у відповідних сервісах, що сприяє зручному розподіленню даних і полегшує роботу з різними базами даних для кожного сервісу.

## Отже, було проведено комплексне проектування бази даних. Спочатку обрано відповідні методології та нотації для проектування, після чого визначено основні сутності й атрибути. На основі попередньо сформованого сценарію роботи системи були відображені зв’язки між сутностями в межах кожного сервісу.

## Такий підхід забезпечив ефективну організацію та збереження даних, забезпечуючи оптимальний доступ до них та високу продуктивність системи. Вірно встановлені зв’язки та оптимізована структура бази даних дозволили сервісам системи взаємодіяти з користувачами та іншими компонентами максимально ефективно, сприяючи надійній та масштабованій архітектурі.

2.3 Модель проектування

## Модель проєктування — це абстрактне уявлення системи, яке використовується під час її розробки для опису структури, поведінки та взаємодії компонентів. Оскільки модель проєктування базується на вимогах до системи, визначених під час етапу аналізу, для створення такого представлення доцільно використовувати інструменти, які є невід’ємною частиною уніфікованого процесу розробки програмного забезпечення, зокрема UML-діаграми.

## UML-діаграми — це тип візуального представлення, що використовується для моделювання й документування програмних систем.

## На цьому етапі проєктування було вирішено почати з опису загальної поведінки та особливостей системи, зокрема функцій та взаємодій між потенційними користувачами та системою. Для цього доречним стало використання одного з інструментів моделювання — діаграми варіантів використання.

## UML-діаграма варіантів використання (Use Case) — це вид UML-діаграми, який застосовується для розробки структури інформаційної системи та демонстрації її загального функціоналу. Ця діаграма не передбачає деталізації конкретних методів, які будуть використовуватися в системі; тому при створенні діаграми необхідно врахувати всі функціональні вимоги та загальні вимоги до системи (див. Рисунок 2.3.2).

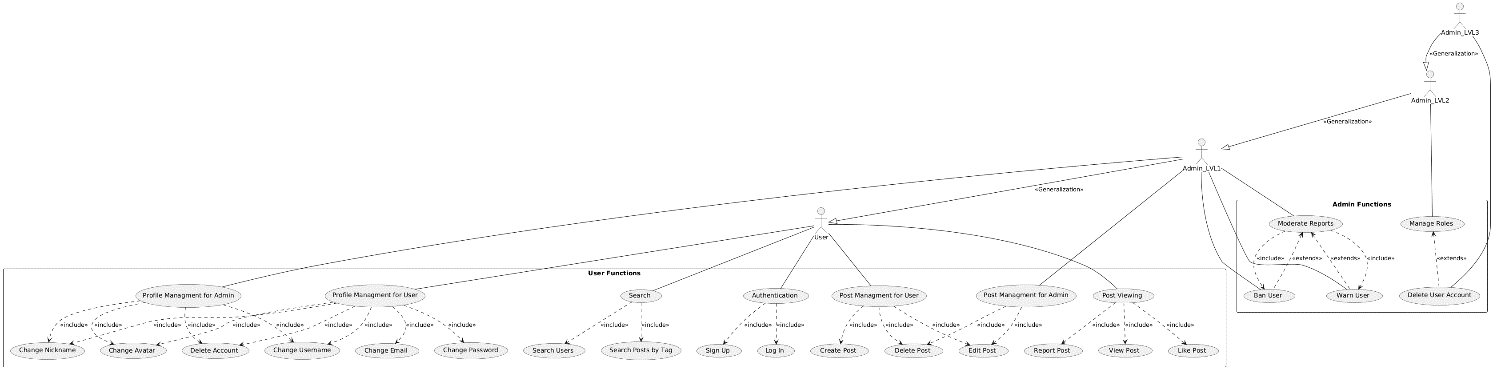


Рисунок 2.3.2 — Діаграма Use Case для соціальної мережі

Після опису загальної поведінки системи було детальніше проаналізовано процес покрокової взаємодії компонентів сервісу, для чого була використана діаграма послідовності.

UML-діаграма послідовності — це тип діаграми UML, що ілюструє порядок повідомлень між об'єктами в процесі їхньої взаємодії. Діаграма послідовності складається з групи об'єктів, представлених лініями життя, та повідомлень, якими вони обмінюються протягом взаємодії. Для наочності використано спрощену діаграму, яка демонструє основні кроки взаємодії (див. Рисунок 2.3.3).

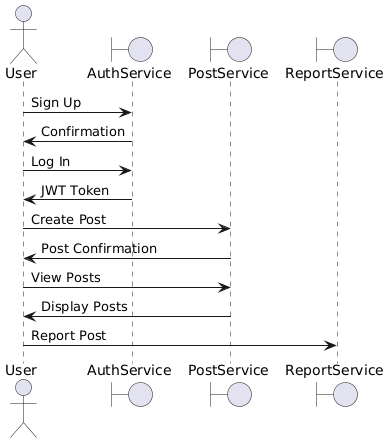


Рисунок 2.3.3 — Діаграма послідовності для соціальної мережі

У межах деталізації аналізу поведінки системи було розглянуто, як саме система виконує поставлені завдання, що були продемонстровані на діаграмі послідовностей. Для цього було використано ще один інструмент UML — діаграму активності.

UML-діаграма активності є типом діаграми UML, яка використовується для опису динамічної поведінки системи. Вона відображає потік управління через систему та взаємодії між різними етапами діяльності за допомогою блок-схем.

Були створені та описані діаграми активності для основних процесів у головних сценаріях роботи системи. Як приклад наведено діаграму процесу модерації, що виконується адміном (див. Рисунок 2.3.4).

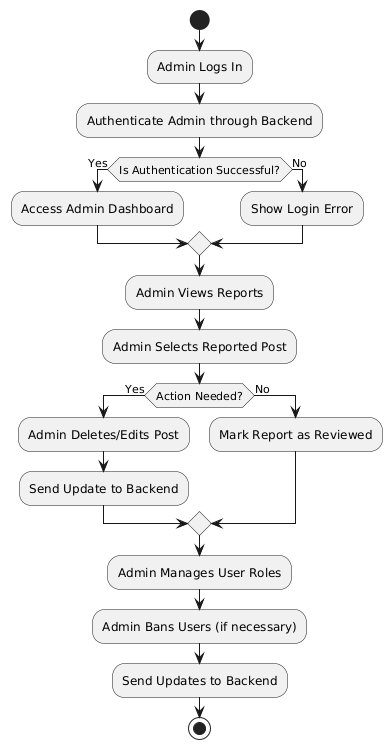


Рисунок 2.3.4 — Діаграма активності для процесу «Модерації» адміном

Таким чином, розробивши діаграми активності для основних функцій системи, було детально описано процес виконання різних дій усередині програмної системи та проаналізовано структуру роботи продукту на конкретному прикладі. Це дозволило створити детальну та комплексну модель програмної системи, що відображає ключові процеси, такі як реєстрація користувачів, модерація постів та управління користувачами в соціальній мережі.

2.4 Вибір засобів розробки

Розробка будь-якої програмної системи є складним і багатогранним процесом, що вимагає використання різноманітних інструментів і технологій. Тому, обираючи засоби розробки системи, важливо враховувати характеристики найпопулярніших мов програмування та інших технологій для створення веб-сервісів. Це допоможе зробити оптимальний вибір, який забезпечить ефективну та якісну розробку.

У контексті вибору засобів для створення соціальної мережі було розглянуто, порівняно та обрано ключові компоненти: мова програмування для бекенду, фреймворк для фронтенду та система керування базами даних (СКБД). Також було розглянуто можливість використання додаткових компонентів, таких як бібліотеки та сторонні сервіси для інтеграції.

Вибір засобів розробки розпочато з визначення мови програмування для бекенду, оскільки цей компонент є фундаментальним для розробки будь-якої системи. Було обрано Go, яка є однією з найкращих мов для створення серверної частини веб-сервісів завдяки своїй високій продуктивності та простоті використання.

Go [6] — це сучасна мова програмування, розроблена компанією Google, яка підходить для створення надійних і продуктивних веб-додатків. Вона відрізняється простим синтаксисом, високою швидкістю виконання та вбудованою підтримкою паралельного програмування.

Плюси:

* Висока продуктивність.
* Простий і зрозумілий синтаксис.
* Вбудована підтримка паралелізму.
* Підтримка численних бібліотек і фреймворків.

Мінуси:

* Менша кількість сторонніх бібліотек у порівнянні з іншими мовами.
* Порівняно нова, тому деякі підходи можуть бути менш зрозумілими для новачків.

Для фронтенду було обрано React [7], популярну бібліотеку JavaScript для створення користувацьких інтерфейсів. React дозволяє ефективно розробляти інтерактивні веб-додатки з компонентним підходом, що спрощує розробку та підтримку проекту.

Плюси:

* Компонентний підхід, що дозволяє повторно використовувати код.
* Велика спільнота та безліч бібліотек для інтеграції.
* Підтримка стану додатка через контекст та хуки.

Мінуси:

* Потребує додаткових знань про JavaScript та JSX.
* Може бути перевантаженим для простих проектів.

## Як систему керування базами даних (СКБД) було обрано MySQL [8], одну з найпопулярніших реляційних баз даних, що забезпечує високу продуктивність, надійність та гнучкість у зберіганні даних. MySQL добре інтегрується з Go і React, що робить її ідеальним вибором для даної соціальної мережі.

## Плюси:

## Висока продуктивність запитів.

## Широка підтримка спільноти та документації.

## Гнучка система управління даними.

## Мінуси:

## Може бути складною для налаштування при роботі з великими обсягами даних.

## Обмежена підтримка деяких типів даних у порівнянні з NoSQL базами даних.

## Таким чином, після детального аналізу обраних мов програмування та технологій було вирішено використовувати Go для розробки бекенду соціальної мережі, React для створення фронтенду та MySQL як СКБД. Цей вибір забезпечить високу продуктивність, зручність у розробці та легкість у підтримці проекту, що особливо важливо для соціальної мережі, де взаємодія з користувачами та обробка даних є критичними аспектами.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Структура та функціональне призначення модулів програмного забезпечення, їх взаємозв’язок

Програмне забезпечення соціальної мережі побудовано на основі мікросервісної архітектури, що передбачає розподіл системи на незалежні сервіси, кожен із яких відповідає за конкретну функціональність. Такий підхід дозволяє спрощувати масштабування системи, покращувати ізоляцію даних, а також забезпечує гнучкість у розробці та підтримці.

Кожен мікросервіс взаємодіє з іншими через визначені API, використовуючи REST або gRPC, залежно від контексту взаємодії. Кожен сервіс має свою власну базу даних, що забезпечує ізоляцію даних, підвищує безпеку та зменшує залежності між сервісами.

Крім того, система має frontend-компонент, який виступає інтерфейсом для кінцевих користувачів і надає доступ до функціональності сервісів.

Основні мікросервіси системи

* Auth-сервіс (сервіс автентифікації):
  + Функціональність:
    - Реєстрація, активація та авторизація користувачів.
    - Генерація та валідація JWT токенів.
  + База даних: Зберігає облікові дані користувачів (логіни, паролі в зашифрованому вигляді) та ролі.
* User-сервіс (сервіс користувачів):
  + Функціональність:
    - Управління профілями користувачів (зміна аватарів, нікнеймів, електронної пошти, паролів).
    - Управління ролями користувачів (звичайний користувач або адміністратор із різними рівнями доступу).
    - Пошук користувачів за критеріями.
    - Видалення облікових записів.
  + База даних: Зберігає деталі профілів користувачів (нікнейми, аватари, дати реєстрації, статуси облікових записів).
* Post-сервіс (сервіс постів):
  + Функціональність:
    - Створення, редагування, перегляд і видалення постів.
    - Додавання, видалення лайків.
    - Додавання, редагування, перегляд і видалення коментів.
    - Пошук постів за тегами.
  + База даних: Зберігає інформацію про пости (зміст, автор, дати створення та оновлення, лайки, коменти).
* Report-сервіс (сервіс репортів):
  + Функціональність:
    - Надсилання та обробка скарг на порушення.
    - Модерація постів адміністраторами.
    - Зміна статусів репортів (очікування, розглянутий, відхилений).
  + База даних: Зберігає деталі репортів (постраждалий об'єкт, користувач, причина, відповідальний адміністратор).

Frontend

* Функціональність:
  + Інтерфейс для користувачів (реєстрація, авторизація, перегляд і управління профілем, створення та перегляд постів, скарги на порушення).
  + Панель адміністратора з функціями модерації (управління постами, обліковими записами, обробка репортів).
  + Взаємодіє з backend через REST API для отримання та передачі даних.
* Технології:
  + Реалізовано на React для створення динамічного та інтерактивного користувацького досвіду.

Взаємозв’язок між модулями

Взаємодія між мікросервісами відбувається через чітко визначені API:

* Auth-сервіс забезпечує автентифікацію та видачу токенів, які перевіряються іншими сервісами перед виконанням захищених операцій.
* User-сервіс з Report для модерації користувачів.
* Post-сервіс працює з Report для обробки скарг на контент.
* Report-сервіс взаємодіє з User та Post для перевірки деталей скарг.

Переваги архітектури

* Масштабованість: Кожен сервіс можна масштабувати незалежно від інших залежно від навантаження.
* Незалежність розробки: Різні команди можуть працювати над сервісами паралельно.
* Ізоляція: Розділення баз даних зменшує ризики збоїв та конфліктів даних.
* Гнучкість: Легке додавання нових мікросервісів без впливу на вже існуючі.

Ця структура дозволяє ефективно організувати функціональність системи, забезпечуючи високий рівень продуктивності та зручності.

3.2 Розробка програмного продукту

Після визначення основних сервісів системи було детально розглянуто процес їх розробки, оскільки створення програмного продукту в мікросервісній архітектурі є багатоступеневим процесом, який включає налаштування взаємодії між сервісами, їх масштабованість та ізоляцію. Нижче описано реалізацію процесу реєстрації користувача, що є важливою частиною функціоналу системи.

Реєстрація користувача включає кілька основних кроків, які взаємодіють між сервісами auth і user.

Заповнення форми на фронтенді.

Користувач заповнює форму реєстрації, де вказує базову інформацію, таку як ім'я, email, пароль. Після цього фронтенд відправляє HTTP-запит на сервіс auth за ендпоінтом /register.

Код з формою та відправкою:

import { handleInputFocus, handleInputBlur } from "../popup/popup";

import { useRef, useState } from "react";

import { useNavigate } from "react-router-dom";

import {useAuth} from "../../context/authContext";

import {constants} from "../../constants";

import {wait} from "@testing-library/user-event/dist/utils";

export function PopupAuth() {

const [email, setEmail] = useState('');

const [pass, setPass] = useState('');

const [confirmPass, setConfirmPass] = useState('');

const [username, setUsername] = useState('');

const {setAccessToken} = useAuth();

const [error, setError] = useState('');

const [loading, setLoading] = useState(false);

const navigate = useNavigate();

const loginRegex = /^[a-zA-Z0-9]{3,52}$/;

const passwordRegex = /^(?=.\*[A-Za-z])(?=.\*\d)[A-Za-z\d]{8,255}$/;

const [isBtnBlocked,setBtnBlocking] = useState(false);

async function dataPickUp(event) {

event.preventDefault();

setError('');

setBtnBlocking(true);

try {

if (!loginRegex.test(username)) {

setError('Username Error');

return;

}

if (!passwordRegex.test(pass)) {

setError('Password Error');

return;

}

if (pass !== confirmPass) {

setError('Confirm Password Error');

return;

}

let user = {

username: username,

email: email,

password: pass,

};

let response = await fetch(`${constants.authApiV1}/register`, {

method: 'POST',

headers: {

'Content-Type': 'application/json;charset=utf-8',

},

body: JSON.stringify(user),

});

let json = await response.json();

if (!response.ok) {

setError(`Error request: ${json.message}`);

return;

}

let activate = await fetch(`${constants.authApiV1}/activate/${json["activation\_token"]}`);

let responseActivationToken = await activate.json();

if (!activate.ok) {

setError(`Error activation: ${responseActivationToken.message}`);

return;

}

setAccessToken(responseActivationToken.token);

navigate("/feed");

} catch (error) {

setError(`Unexpected error: ${error.message}`);

} finally {

setBtnBlocking(false);

}

}

const maxLength = 52;

function usernameOnChange(event) {

setUsername(event.target.value);

const popupInput = event.currentTarget.closest(".popup-input");

if (username.length > maxLength) {

popupInput.classList.add("error");

} else {

popupInput.classList.remove("error");

}

}

function passwordOnChange(event) {

setPass(event.target.value);

const popupInput = event.currentTarget.closest(".popup-input");

if (username.length > maxLength) {

popupInput.classList.add("error");

} else {

popupInput.classList.remove("error");

}

}

return (

<form onSubmit={dataPickUp} className="popup-tab">

<div className="popup-title"><h3>Create your account</h3></div>

<div onClick={handleInputFocus} className='popup-input'>

<div className="popup-input\_\_text">

<span className='popup-input\_\_text-name'>Username</span>

<span className='popup-input\_\_text-symbols'>{username.length}/{maxLength}</span>

</div>

<input required onBlur={handleInputBlur} type="text" onChange = {usernameOnChange} />

</div>

<div onClick={handleInputFocus} className="popup-input">

<div className="popup-input\_\_text">

<span className='popup-input\_\_text-name'>Email</span>

</div>

<input required onBlur={handleInputBlur} type="email" onChange = {(event) => {setEmail(event.target.value)}} />

</div>

<div onClick={handleInputFocus} className="popup-input">

<div className="popup-input\_\_text">

<span className='popup-input\_\_text-name'>Password</span>

<span className='popup-input\_\_text-symbols'>{pass.length}/{maxLength}</span>

</div>

<input required onBlur={handleInputBlur} type="password" onChange = {passwordOnChange} />

</div>

<div onClick={handleInputFocus} className="popup-input">

<div className="popup-input\_\_text">

<span className='popup-input\_\_text-name'>Confirm password</span>

</div>

<input required onBlur={handleInputBlur} type="password" onChange = {(event) => {setConfirmPass(event.target.value)}} />

</div>

<button disabled={isBtnBlocked} type='submit' className="popup-next">

<span>Confirm</span>

</button>

{error && <div className="popup-error">{error}</div>}

</form>

);

}

Після отримання запиту register в сервіс auth:

1. Перевіряє коректність даних (email, пароль тощо).
2. Перевіряє відсутність користувача з таким email у своїй базі даних.
3. Створює новий запис у базі даних, але встановлює статус користувача як "неактивний".
4. Генерує код активації (одноразовий токен), який повертається фронтенду.

Код обробки реєстрації в transport layer http:

func (h \*Handler) userRegister(c echo.Context) error {

var body struct {

Email string `json:"email" validate:"required,email"`

Username string `json:"username" validate:"required,username"`

Password string `json:"password" validate:"required,password"`

}

if err := c.Bind(&body); err != nil {

return echo.NewHTTPError(http.StatusBadRequest, err.Error())

}

if err := c.Validate(&body); err != nil {

return echo.NewHTTPError(http.StatusBadRequest, err.Error())

}

activationToken, err := h.userService.Register(body.Username, body.Email, body.Password)

if err != nil {

return echo.NewHTTPError(http.StatusInternalServerError, err.Error())

}

return c.JSON(http.StatusOK, struct {

ActivationToken string `json:"activation\_token"`

}{

ActivationToken: activationToken,

})

}

Код реєстрації користувача в service layer:

func (us \*UserService) Register(username, email, password string) (activationToken string, err error) {

exists, err := us.repo.ExistsByUsername(username)

switch {

case err != nil:

return "", err

case exists:

return "", ErrUsernameTaken

}

exists, err = us.repo.ExistsByEmail(email)

switch {

case err != nil:

return "", err

case exists:

return "", ErrEmailTaken

}

hashedPassword := us.hasher.Hash(password)

user := &model.User{

Email: email,

Username: username,

Password: hashedPassword,

}

if err := us.repo.Add(user); err != nil {

return "", err

}

activationToken, err = us.activationTokenService.Generate(user.ID)

if err != nil {

return "", err

}

return activationToken, nil

}

Активація користувача через ендпоінт /activate

Після отримання коду активації фронтенд надсилає HTTP-запит на сервіс auth за ендпоінтом /activate.

Після отримання запиту activate в сервіс auth:

1. Валідує токен активації.
2. Перевіряє наявність неактивного користувача, прив'язаного до цього токена.
3. Через gRPC-запит викликає метод CreateUser у сервісі user.
4. Після створення користувача в user-сервісі, cервіс auth змінює статус користувача в своїй базі даних на "активний".
5. Токен активації видаляється через непотрібність.
6. Генеруються нові access і refresh токени, які повертаються фронтенду для авторизації.

Код обробки активації користувача в transport layer http:

func (h \*Handler) userActivate(c echo.Context) error {

activationToken := c.Param("token")

accessToken, refreshToken, err := h.userService.Activate(activationToken)

if err != nil {

return echo.NewHTTPError(http.StatusUnauthorized, err.Error())

}

return h.setAndReturnTokens(c, accessToken, refreshToken)

}

Код активації користувача в service layer:

func (us \*UserService) Activate(activationToken string) (accessToken, refreshToken string, err error) {

token, err := us.activationTokenService.GetByToken(activationToken)

if err != nil {

return "", "", err

}

user, err := us.repo.Find(token.UserID)

if err != nil {

return "", "", err

}

if err := us.userGateway.CreateUser(context.Background(), user.ID, constant.RoleUser, user.Username, user.Email); err != nil {

return "", "", err

}

user.IsActivated = true

if err := us.repo.Save(user); err != nil {

return "", "", err

}

if err := us.activationTokenService.Delete(token.UserID); err != nil {

return "", "", err

}

accessToken, refreshToken, err = us.tokenService.Generate(user.ID)

if err != nil {

return "", "", err

}

return accessToken, refreshToken, nil

}

У user-сервісі:

1. Створюється запис про користувача з інформацією, отриманою від auth.
2. Дані користувача індексуються для можливості швидкого пошуку.

Код обробки створення користувача в transport layer grpc:

func (h \*Handler) CreateUser(ctx context.Context, r \*gen.CreateUserRequest) (\*gen.CreateUserResponse, error) {

user, err := h.us.Create(uint(r.GetUserId()), uint(r.GetRole()), r.GetUsername(), r.GetEmail())

if err != nil {

return nil, err

}

return &gen.CreateUserResponse{

UserId: uint64(user.ID),

}, nil

}

Код створення користувача в service layer:

func (us \*UserService) Create(id, role uint, username, email string) (\*model.User, error) {

exists, err := us.repo.ExistsByUsername(username)

switch {

case err != nil:

return nil, err

case exists:

return nil, ErrUserUsernameTaken

}

exists, err = us.repo.ExistsByEmail(email)

switch {

case err != nil:

return nil, err

case exists:

return nil, ErrUserEmailTaken

}

user := &model.User{

ID: id,

Email: email,

Username: username,

Nickname: username,

Role: role,

}

if err := us.repo.Add(user); err != nil {

return nil, err

}

return user, nil

}

Код добавлення запису в бд в repository layer:

func (ur \*UserRepository) Add(user \*model.User) error {

if err := ur.db.Create(user).Error; err != nil {

return err

}

return ur.s.Index(user)

}

Код індексування користувача для пошуку:

func (u \*User) Index(user \*model.User) error {

res, err := u.client.Index(userIndex).Id(fmt.Sprintf("%d", user.ID)).Request(user).Do(context.TODO())

if err != nil {

return err

}

if res.Result.Name != "created" && res.Result.Name != "updated" {

return errors.New("failed to index user")

}

return nil

}

Таким чином, процес реєстрації враховує розподіл відповідальностей між сервісами, що дозволяє підвищити гнучкість і масштабованість системи. Усі дані користувачів зберігаються ізольовано в окремих базах даних кожного сервісу, що відповідає принципам мікросервісної архітектури.

3.3 Інструкція користувача

3.4 Вимоги до програмно-технічного середовища функціонування ПЗ

Вимоги для користувача та для серверної частини (див. таблиці 3.4.1–3.4.2).

Таблиця 3.4.1 - Системні вимоги для користувача вебсервісу

|  |  |
| --- | --- |
| Назва характеристики | Припустиме значення |
| Операційна система | Windows, macOS, Linux |
| Браузер | Google Chrome, Opera GX, Safari, та інші. |
| Дисплей | 1024x768 пікселів |
| Процесор | 1,6 ГГц |
| Оперативна пам'ять | 2 ГБ |
| Місце на диску | 100 МБ |

Таблиця 3.4.2 - Характеристики серверного обладнання

|  |  |
| --- | --- |
| Операційна система | Linux, Windows, macOS |
| Інтегроване середовище розробки (IDE) | GoLand, WebStorm (для фронтенду) |
| Мова програмування | Go, JavaScript (для фронтенду) |
| Система керування базами даних (СКБД) | MySQL |
| Додаткові інструменти та бібліотеки | GORM, Elasticsearch, Echo framework (для бекенду), React (для фронтенду) |

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Методика тестування програмного забезпечення

Тестування є важливою складовою частиною процесу розробки програмного забезпечення, особливо для великих проєктів, таких як "Social Network Y". Воно дозволяє перевірити функціональність, надійність та безпеку системи, а також забезпечити її правильну роботу в різних умовах. У цьому розділі розглядаються основні методи тестування, що застосовуються в проєкті, а також обґрунтований вибір підходів для реалізації тестів.

Модульне тестування дозволяє перевіряти окремі компоненти програми, такі як функції, класи або методи. Для тестування бекенду проєкту використовуються інструменти, такі як Go testing framework. Це дозволяє ізолювати та тестувати логічні одиниці програми, забезпечуючи коректність виконання окремих компонентів.

Функціональне тестування перевіряє функціональність системи на рівні взаємодії з користувачем. У контексті "Social Network Y" функціональні тести включають перевірку роботи API-ендпоінтів, реєстрації, аутентифікації, редагування профілю, створення, редагування та видалення постів, а також перевірку користувацького інтерфейсу на фронтенді. Тестування API здійснюється через Postman, де створюються тестові сценарії для перевірки коректності відповідей від серверних ендпоінтів.

Інтеграційне тестування перевіряє взаємодію між різними компонентами системи, такими як моделі, контролери та бази даних. Це дозволяє виявити помилки, які можуть виникнути під час взаємодії між різними частинами системи. Для тестування взаємодії між мікросервісами використовуються інструменти для перевірки правильності роботи гRPC та REST API.

Тестування бази даних включає перевірку коректності операцій з базою даних, таких як вставка, оновлення та видалення даних. У "Social Network Y" це забезпечується за допомогою інструментів Go для роботи з MySQL базою даних та бібліотеки GORM для перевірки правильності виконання запитів та міграцій.

Ручне тестування здійснюється тестувальниками для оцінки зручності інтерфейсу, його відображення, поведінки та перевірки складних бізнес-логік, які важко передбачити автоматизованими тестами. Ручне тестування є важливим на завершальних етапах розробки, коли необхідно перевірити, чи правильно відображається інформація на фронтенді, чи зручна взаємодія з користувачем, а також для виявлення нестандартних помилок у роботі системи.

4.2 Результати тестування програмного забезпечення

Тестування програмного забезпечення є важливим етапом розробки, спрямованим на забезпечення якості продукту, його відповідності вимогам та стабільної роботи в різних умовах. У проєкті "Social Network Y" для модульного тестування використовуються різноманітні методи, зокрема, інструменти для тестування, такі як Go testing framework і mocking бібліотеки для імітації поведінки компонентів.

Для тестування функціональності авторизації в проєкті було розроблено кілька модульних тестів, які перевіряють правильність обробки вхідних даних та генерацію токенів для успішної авторизації, а також обробку помилок у разі неправильних даних.

Код тесту для функціональності авторизації користувача:

func TestUserService\_Login(t \*testing.T) {

userRepo := new(mocks.UserRepoMock)

tokenService := new(mocks.TokenServiceMock)

hasher := new(mocks.HasherMock)

userID := uint(1)

username := "testuser"

email := "test@example.com"

password := "password123"

// Setup successful login scenario

hashedPassword := "hashed\_password123" // Hash of the password

expectedAccessToken := "access\_token"

expectedRefreshToken := "refresh\_token"

userRepo.On("FindByLogin", username).Return(&model.User{

ID: userID,

Email: email,

Username: username,

Password: hashedPassword,

IsActivated: true,

}, nil)

tokenService.On("Generate", userID).Return(expectedAccessToken, expectedRefreshToken, nil)

hasher.On("Verify", hashedPassword, password).Return(true)

service := NewUserService(userRepo, tokenService, new(mocks.UserActivationTokenServiceMock), hasher, new(mocks.UserGatewayMock))

accessToken, refreshToken, err := service.Login(username, password)

assert.NoError(t, err)

assert.Equal(t, expectedAccessToken, accessToken)

assert.Equal(t, expectedRefreshToken, refreshToken)

// Setup failure scenarios

testCases := []struct {

description string

username string

password string

expectedErr error

}{

{

description: "Invalid password",

username: username,

password: "wrongpassword123",

expectedErr: ErrInvalidPassword,

},

{

description: "User not activated",

username: username,

password: password,

expectedErr: ErrUserNotActivated,

},

}

for \_, tc := range testCases {

t.Run(tc.description, func(t \*testing.T) {

userRepo := new(mocks.UserRepoMock)

hasher := new(mocks.HasherMock)

service.repo = userRepo

service.hasher = hasher

userRepo.On("FindByLogin", tc.username).Return(&model.User{

ID: userID,

Email: email,

Username: tc.username,

Password: hashedPassword,

IsActivated: !errors.Is(ErrUserNotActivated, tc.expectedErr),

}, nil)

hasher.On("Verify", hashedPassword, tc.password).Return(!errors.Is(ErrInvalidPassword, tc.expectedErr))

accessToken, refreshToken, err := service.Login(tc.username, tc.password)

assert.Error(t, err)

assert.Empty(t, accessToken)

assert.Empty(t, refreshToken)

assert.Equal(t, tc.expectedErr, err)

})

}

}

У цьому тесті перевіряється успішний сценарій авторизації, де перевіряється правильність отриманих токенів для авторизованого користувача. Також є тести для перевірки помилок, таких як неправильний пароль або неактивований користувач. Усі ці тести використовують моки для імітації поведінки репозиторіїв, хешера та сервісу генерації токенів.

Для запуску тестів необхідно виконати команду:

go test

Результати тестування будуть відображені в терміналі, де можна побачити успішні тести та помилки (рисунок 4.2.1).

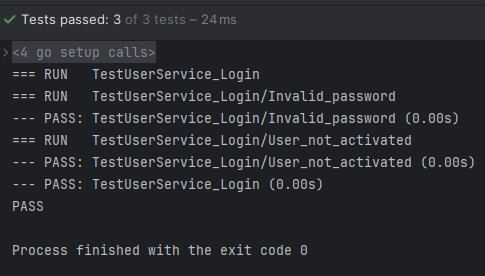
****

Рисунок 4.2.1 - Результат успішної авторизації

У програмній системі передбачено контроль введення даних у формі авторизації, що забезпечує правильність обробки інформації, введеної користувачем. Було перевірено захист від можливих помилок при заповненні форми, включаючи випадки введення некоректної або неповної інформації. Наприклад, тестування показало, що при спробі надіслати форму авторизації з порожніми або некоректними полями система генерує відповідне повідомлення про помилку, що повністю відповідає вимогам (рисунок 4.2.2).

Рисунок 4.2.2 - Результат мануального тестування

Процес тестування дав змогу впевнитися у коректній роботі системи з точки зору її використання кінцевим користувачем, а також виявити потенційні недоліки інтерфейсу та зручності взаємодії. Такі аспекти часто залишаються поза увагою автоматизованих тестів. Поєднання автоматизованого і ручного тестування дозволяє досягти високого рівня стабільності та надійності вебпорталу, своєчасно виявляючи помилки і забезпечуючи збереження якості створеного продукту.

ВИСНОВКИ

У процесі виконання курсового проєкту «Соціальна мережа Y» було створено функціональну веб-платформу, що надає можливість користувачам реєструватися, авторизуватися, публікувати та редагувати пости, здійснювати пошук користувачів і публікацій, а також взаємодіяти з контентом через лайки та коментарі. Крім того, система підтримує адміністративні функції, такі як управління користувачами, модерація постів, обробка звітів про порушення та управління правами доступу. Використання мікросервісної архітектури, а також технологій, таких як JWT для аутентифікації, Go для серверної частини та React для фронтенду, забезпечило високу надійність і масштабованість платформи.

У процесі роботи значно розширено знання в області розробки сучасних веб-додатків, зокрема щодо реалізації мікросервісної архітектури, роботи з REST/gRPC API, а також впровадження механізмів аутентифікації та авторизації користувачів за допомогою JWT. Додатково освоєно методи розробки автоматизованих тестів для перевірки основних функціональних аспектів, що дозволило забезпечити стабільність системи та швидко виявляти й усувати помилки. Проведене ручне тестування підтвердило коректну роботу ключових функцій, таких як реєстрація, авторизація, публікація постів, редагування профілю та взаємодія з контентом.

Незважаючи на досягнуті результати, проєкт має значний потенціал для подальшого вдосконалення. Зокрема, можна розширити функціональність пошуку, додавши можливість фільтрування за різними параметрами, такими як популярність постів або типи контенту. Також є можливість вдосконалення адміністративної панелі для кращого управління користувачами, правами доступу та звітами про порушення. Для покращення взаємодії з користувачами доцільно додати функціонал сповіщень про нові повідомлення, оновлення профілю або нові пости.

Проєкт досяг поставленої мети, забезпечивши створення працездатної соціальної мережі, готової до запуску та подальшого розвитку. Розроблена система може бути використана як основа для повноцінної соціальної платформи або як навчальний приклад, що ілюструє процеси розробки мікросервісів, адміністрування користувачів і контенту, а також безпеки в сучасних веб-додатках.

**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

1. X [URL] — <https://x.com/> (дата звернення 30.10.24).
2. Facebook [URL] — <https://facebook.com/> (дата звернення 30.10.24).
3. Instagram [URL] — <https://instagram.com/> (дата звернення 30.10.24).
4. Microservice Architecture pattern [URL] — https://microservices.io/patterns/microservices.html (дата звернення 30.10.24).
5. IDEF1X. Cheat Sheet [URL] — https://www.scribd.com/doc/316999129/IDEF1X-Cheat-Sheet (дата звернення 30.10.24).
6. The Go Programming Language [URL] — [https://go.dev/](https://go.dev/%20) (дата звернення 30.10.24).
7. React [URL] — [https://react.dev/](https://react.dev/%20) (дата звернення 30.10.24).
8. MySQL [URL] — <https://www.mysql.com/> (дата звернення 30.10.24)

Додаток А

Технічне завдання

* **Загальна інформація**
  + **Назва проекту**: Social network Y
  + **Мета проекту**: Розробити та реалізувати соціальну мережу з базовими функціональними можливостями для комунікації так взаємодії між користувачами.

## **Опис продукту**

* + **Функціонал для користувачів**:
    - Реєстрація та аутентифікація.
    - Публікація, редагування та видалення постів.
    - Лайкання постів.
    - Коментування постів.
    - Перегляд постів.
    - Повідомлення про порушення.
    - Зміна пароля, аватарки, нікнейму, username (login) та email.
    - Видалення акаунта.
    - Пошук користувачів.
    - Пошук постів по тегам.
  + **Функціонал для адміністрації**:
    - Модерація постів/коментів користувачів (редагування, видалення).
    - Перегляд та обробка повідомлень про порушення.
    - Керування правами користувачів та інших адміністраторів.
    - Зміна аватарок, нікнеймів, username (login) та email користувачів.
    - Блокування користувачів.
    - Видалення акаунтів користувачів (тільки для адміністраторів рівня 3).
* **Функціональні вимоги**
  + **Користувачі**
    - **Реєстрація**:
      * Реєстрація за email або username з перевіркою унікальності.
    - **Аутентифікація**:
      * Вхід користувачів з отриманням JWT токенів для подальших дій у системі.
    - **Пости**:
      * Створення, редагування, видалення та перегляд власних постів.
      * Коментування постів.
      * Лайкання постів.
    - **Зміна пароля, аватарки, нікнейму, username та email**:
      * Користувач може змінювати свій пароль, аватар, нікнейм, username (login) та email.
      * Адміністратор може змінювати ці дані для будь-якого користувача.
    - **Видалення акаунта**:
      * Користувач може видалити свій акаунт.
      * Адміністратор рівня 3 може видаляти акаунти будь-яких користувачів.
    - **Повідомлення про порушення**:
      * Повідомлення про порушення публікацій.
    - **Пошук користувачів, тегів**:
      * Можливість пошуку користувачів за їхнім ім'ям або нікнеймом.
      * Можливість пошуку постів за тегами.
  + **Адміністратори**
    - **Ролі та рівні доступу**:
      * **Рівень 1-3**: перегляд повідомлень про порушення.
      * **Рівень 2-3**: призначення ролей користувачам та адміністраторам.
      * **Рівень 3**: модерація інших адміністраторів та видалення акаунтів користувачів.
    - **Модерація контенту**:
      * Видалення та редагування контенту інших користувачів.
    - **Блокування користувачів**:
      * Блокування порушників.
    - **Зміна пароля, аватарок, нікнеймів, username та email**:
      * Адміністратори можуть змінювати пароль, аватар, нікнейм, username (login) та email користувачів.
* **Нефункціональні вимоги**
  + **Продуктивність**
    - Час відповіді сервера не більше 500 мс при навантаженні до 1000 користувачів.
  + **Безпека**
    - Захищені JWT токени для аутентифікації.
    - Хешування паролів.
    - Захист від атак (SQL-ін'єкції, XSS, CSRF).
  + **Масштабованість**
    - Система повинна підтримувати можливість розширення функціоналу та масштабування мікросервісів для обслуговування великої кількості користувачів.
* **Інтерфейс користувача**
  + **Головна сторінка**
    - Перелік постів з можливістю лайкання, коментування та повідомлення про порушення.
  + **Сторінка профілю**
    - Інформація про користувача.
    - Можливість редагування та видалення постів.
    - Зміна аватарки, нікнейму, пароля, username (login) та email.
    - Видалення акаунта.
  + **Пошук**
    - Інтерфейс для пошуку користувачів за ім'ям або нікнеймом.
    - Інтерфейс для пошуку постів за тегами.
  + **Адмін панель**
    - Інтерфейс для перегляду і модерації постів.
    - Можливість перегляду повідомлень про порушення.
    - Керування ролями користувачів та адміністраторів.
    - Можливість блокування користувачів.
    - Можливість видалення акаунтів користувачів.
* **Технологічні вимоги**
  + **Backend**: Go (Golang), GORM, MySQL.
  + **Frontend**: HTML/CSS/JavaScript/React.
  + **Сервіси**: мікросервіси (auth, post, report, user).
  + **API**: REST/gRPC для взаємодії фронтенду з бекендом.
* **Опціональний функціонал (може бути додано у майбутньому)**
  + **Чат** між користувачами.
  + **Система повідомлень** (сповіщення про нові пости, лайки, коментарі).
  + **Інтеграція зі сторонніми сервісами** (наприклад, для автентифікації через Google або Facebook).
  + **Статистика та аналітика** для користувачів і постів (кількість лайків, коментарів тощо).