МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЇ ТА ДИЗАЙНУ

Факультет мехатроніки та комп’ютерних технологій
Кафедра комп’ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розроблення WEB-додатку соціальна мережа “MilitaryGram” для ЗСУ

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп’ютерні науки
Освітня програма Комп’ютерні науки

Виконав: студент групи МгІТ-23
Ремезенко С.В.
Науковий керівник к.т.н., доц. Чупринка Н.В.
Рецензент

_____
(науковий ступінь, вчене звання, прізвище та ініціали)

Київ 2024

Форма завдання на кваліфікаційну роботу

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЇ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук та технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри КН

_____ _____

(підпис) (Власне ім'я та ПРІЗВИЩЕ)

«_____» _____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ремезенко Сергій Віталійович

1. Тема кваліфікаційної роботи Розроблення WEB-додатку соціальна мережа "MilitaryGram" для ЗСУ

Науковий керівник роботи к.т.н., доц. Чупринка Наталія Вікторівна

Затверджені наказом КНУТД від 03.09.2024 року № 188-уч

2. Вихідні дані до кваліфікаційної роботи: Розробки кафедри комп'ютерних наук та технологій, рекомендована література, додатки.
3. Зміст кваліфікаційної роботи : Вступ, Розділ 1. Проектування концептуальної моделі; Розділ 2. Проектування бази даних; Розділ 3. Побудова серверної частини програми; Розділ 4. Побудова інтерфейсу програми; Висновки; Список літератури; Додаток А програмний код; Додаток Б презентація.
4. Дата видачі завдання 08.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапу кваліфікаційної роботи	Орієнтовний термін виконання	Примітка про виконання
1	Вступ		
2	Розділ 1. ПРОЕКТУВАННЯ КОНЦЕПТУАЛЬНОЇ МОДЕЛІ		
3	Розділ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ		
4	Розділ 3. ПОБУДОВА СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМИ		
5	Розділ 4. ПОБУДОВА ІНТЕРФЕЙСУ ПРОГРАМИ		
7	ВИСНОВКИ		
8	Оформлення (чистовий варіант)		
9	Подача кваліфікаційної роботи для рецензування		
10	Перевірка кваліфікаційної роботи на наявність ознак плагіату		
11	Подання кваліфікаційної роботи завідувачу кафедри (за 7 днів до захисту)		

З завданням ознайомлений:

Студент _____ Сергій РЕМЕЗЕНКО
(підпис)

Науковий керівник _____ Наталія ЧУПРИНКА
(підпис)

АНОТАЦІЯ

Ремезенко С.В. Розроблення WEB-додатку соціальна мережа “MilitaryGram” для ЗСУ.

Кваліфікаційна робота за спеціальністю 122 - «Комп’ютерні науки» – Київський національний університет технологій та дизайну, Київ, 2024 рік.

Кваліфікаційна робота присвячена розробленню WEB-додатку окремої соціальної мережі для ЗСУ.

В роботі були проаналізовані існуючі соціальні мережі та рішення. Також був проведений аналіз наукових статей потрібності такого додатку. Визначено сучасні інструменти, технології та методи розробки програмного забезпечення та переваги вибраних технологій.

Ключові слова: WEB-додаток, Java, Spring Boot, SQL, MySQL, REST, JWT, Angular, BootStrap, MilitaryGram.

ANNOTATION

Remezenko S.V. Development of WEB-application social network "MilitaryGram" for the Armed Forces of Ukraine.

Qualification work in specialty 122 - "Computer Science". - Kyiv National University of Technology and Design, Kyiv, 2024.

The qualification work is devoted to the development of a WEB application of a separate social network for the Armed Forces.

The work analyzed existing social networks and solutions. An analysis of scientific articles on the necessity of such an application was also conducted. Modern software development tools, technologies and methods, and advantages of selected technologies are identified.

Keywords: Web technologies, Java, Spring Boot, SQL, MySQL, REST, JWT Angular, BootStrap, MilitaryGram.

ЗМІСТ

ВСТУП

Актуальність дипломної роботи полягає в розробці та впровадженні соціальної мережі, орієнтованої на потреби військовослужбовців, їхніх родин, волонтерів та фахівців, що працюють у сфері безпеки та підтримки. В умовах сучасних інформаційних викликів та війни, потреба в ефективних інструментах для комунікації, обміну інформацією та підтримки морального духу є надзвичайно важливою. Створення безпечної та зручної платформи для спілкування в таких умовах допоможе значно підвищити оперативність обміну інформацією і підтримку бойового духу серед військовослужбовців та інших зацікавлених сторін.

Мета дипломної роботи: розробка веб-додатку для ефективної комунікації та підтримки зв’язків між військовослужбовцями, їхніми родинами, волонтерами та психологами шляхом створення спеціалізованої соціальної мережі з урахуванням специфічних потреб цієї аудиторії.

Об’єкт дослідження: розробка і впровадження веб-додатку для військових, що включає функції соціальної мережі, орієнтованої на забезпечення ефективної комунікації та підтримки користувачів в умовах війни.

Предмет дослідження: технології та методи створення соціальної мережі для військових, яка включає такі функції як створення профілів, обмін фотографіями та постами, коментування та підтримка взаємодії користувачів.

У роботі висвітлено:

- розробку архітектури веб-додатку, який відповідає вимогам безпеки, зручності та швидкості комунікації;
- створення основних функцій соціальної мережі, зокрема для обміну інформацією, підтримки морального духу та координації дій між військовими підрозділами;
- інтеграцію оптимізованих інтерфейсів для кожної з цільових аудиторій (військовослужбовці, волонтери, психологи) для забезпечення зручності та ефективності взаємодії.

РОЗДІЛ 1
ПРОЕКТУВАННЯ КОНЦЕПТУАЛЬНОЇ МОДЕЛІ

1.1. Поняття концептуальної моделі

Метою є розробка концепції веб-додатку соціальної мережі, що сприятиме взаємодії між військовослужбовцями, їхніми родинами та волонтерами, забезпечуючи зручний доступ до інформації та підтримки.

Розроблений веб-додаток повинен забезпечувати наступні можливості:

- реєстрація користувачів соціальної мережі;
- авторизація користувачів;
- редагування ім’я та прізвища;
- редагування спеціальності;
- додавання/видалення біографії користувача;
- додавання/редагування фотографії користувача;
- додавання/видалення постів користувача з можливістю прикріпити фото і коментар;
- додавання коментарів та вподобайок до постів інших користувачів;
- видалення коментарів інших користувачів зі свого посту;
- перегляд вподобайок.

Концептуальна модель - модель предметної області, що складається з переліку взаємопов'язаних понять, що використовуються для опису цієї області, разом з властивостями й характеристиками, класифікацією цих понять, за типами, ситуацій, ознаками в даній області і законів протікання процесів в ній.

При розробці концептуальної моделі ми будемо користуватися наступними поняттями:

а) Сутність - особистості, факти, об'єкти реального світу, що мають відношення до деякої проблемної області.

б) Атрибут - це інформаційне відображення властивостей об'єкта. При реалізації інформаційної моделі на будь-якому носії інформації, атрибут часто

називають елементом даних, полем даних або просто полем.

в) Примірник об'єкта - це один набір значень його елементів даних.

г) Доменом називається набір записів даних одного типу, що відповідають поставленимумовам.

д) Зв'язок - це функціональна залежність між сутностями.

е) Концептуальна модель представляє інтегровані концептуальні вимоги всіх користувачів до бази даних даної предметної області.

є) Концептуальна схема - це графічне представлення даних на концептуальному рівні[1-6].

Побудова семантичної моделі предметної області є початковою стадією проектування системи баз даних, яка базується на аналізі властивостей і природи об'єктів предметної області та інформаційних потреб майбутніх користувачів системи, що розробляється. Цю стадію називають концептуальним проектуванням системи. Її результатом є концептуальна модель предметної області.

Концептуальний рівень відповідає логічному аспекту уявлення даних предметної області в інтегрованому вигляді. Концептуальна модель складається з безлічі екземплярів різних типів даних, структурованих відповідно до вимог СУБД до логічного структурі бази даних[7-10].

Призначення концептуальних моделей визначає і деякі специфічні вимоги до засобів їх подання. Крім згаданої незалежності від середовища (обладнання) та вимоги адекватності відображення предметної області відзначимо наступні:

- формалізованість, що забезпечує можливість автоматизованої обробки, в тому числі, наприклад, автоматичний контроль несуперечності;
- дружність, що забезпечує можливість використання наочних графічних засобів відображення і обробки їх користувачем.

1.2. Проектування концептуальної моделі

MilitaryGram додаток, в якому буде використовуватись база даних, призначений для менеджменту користувачів та їх постів, зберігання нових користувачів з персональними даними та нові пости з фотографіями та їх

коментарями, відповідає за актуальність цих записів. В базі даних повинен зберігатись весь перелік користувачів, їх реєстраційні дані, пости та фотографії, фотографії профілю тощо[11].

Отже, аналізуючи вище описане, можна сформувати та реалізувати сутності до нашої бази даних:

а) Користувачі – порядковий номер, логін, пароль, ім’я, прізвище, зашифрований пароль, дата створення профілю;

б) Пости - порядковий номер, назва, опис, локація фотографії, кількість вподобайок, дата створення;

в) Коментарі – порядковий номер, ім’я користувача, повідомлення коментаря, дата створення;

г) Фотографії – порядковий номер, назва, порядковий номер посту, порядковий номер користувача.

1.3. Опис функціоналу

Веб-додаток, що проектується має ряд вимог. В загальному вигляді опис функціоналу включає опис функцій, які потрібні для спрощення використання соціальної мережі у повсякденному житті.

Додаток MilitaryGram відповідає за менеджмент постів та фотографій користувачів у зручний спосіб через сучасний веб-додаток. В додатку має бути можливість додавати та видаляти пости з фотографіями та коментарями, редагувати свої персональні данні(змінювати фотографію профілю, ім’я, прізвище та біографію, редагувати пост(назву, опис, локацію фотографії), можливість авторизації, реєстрації та можливість виходу зі свого профілю.

Після аналізу вимог, структуризації, конструювання архітектури та відношень між сутностями можна сформувати ряд функціональних вимог до додатку “MilitaryGram”:

1. дозволяти користувачу реєструватись;
2. дозволяти користувачу авторизуватись;
3. дозволяти користувачу додавати, видаляти та редагувати особисту

інформацію в особистому кабінеті;

4. дозволяти користувачу змінювати фотографію профілю;
5. дозволяти користувачу в особистому кабінеті переглядати коментарі та вподобайки від інших користувачів під своїми постами, видаляти коментарі інших користувачів до своїх постів;
6. дозволяти користувачу коментувати пости інших користувачів;
7. дозволяти користувачу залишати вподобайку під постами інших користувачів;
8. дозволяти користувачам переглядати всі пости зареєстрованих користувачів;
9. дозволяти користувачеві виходити зі свого профілю.

З заданим списком функціональних вимог вже можна приступати до проектування таблиць та зв’язками між ними, оскільки з вимог можна чітко зрозуміти, які дії повинні виконуватись над записами таблиці, які сутності повинні бути виділені в окремі таблиці, і які характеристики повинна мати кожна сутність.

1.4. Висновки до розділу 1

У цьому розділі було проведено аналіз вимог до веб-додатку "MilitaryGram", спрямованого на підтримку взаємодії між військовослужбовцями, їхніми родинами та волонтерами. Описано концептуальну модель, яка визначає сутності та зв'язки між ними, а також функціональні вимоги до системи, що включають можливості реєстрації, авторизації, редагування профілю, управління постами та коментарями. Згідно з концептуальною моделлю, для реалізації бази даних необхідно виділити основні сутності: **Користувачі**, **Пости**, **Коментарі**, та **Фотографії**, кожна з яких має власні атрибути, що відображають ключові характеристики об'єктів. Так, наприклад, сутність **Користувачі** містить інформацію про логін, пароль, ім’я, прізвище, а також дату створення профілю, тоді як сутність **Пости** зберігає дані про пост, його опис, фотографії та кількість вподобайок.

Розроблена концептуальна модель створює основу для проектування бази даних і визначає, як ці сутності будуть взаємодіяти між собою для забезпечення необхідного функціоналу. Завдяки детальному аналізу функціональних вимог, визначено, що користувачі повинні мати можливість не тільки взаємодіяти з контентом через додавання постів і коментарів, а й здійснювати персоналізовані налаштування профілю, змінювати фотографії, редагувати свої пости та видаляти коментарі інших користувачів.

Ключовим етапом є подальше проектування таблиць бази даних, що має ґрунтуватися на визначених сутностях та зв’язках. Це дозволить забезпечити правильну організацію даних і ефективну обробку запитів користувачів.

Таким чином, розробка концептуальної моделі та визначення основних функціональних вимог дозволяє сформувати чітке уявлення про структуру майбутнього додатку і є важливим кроком на шляху до створення повноцінної соціальної мережі для військовослужбовців та їхніх родин.

РОЗДІЛ 2
ПРОЕКТУВАННЯ БАЗИ ДАНИХ

2.1. Поняття запиту на мові SQL

Запит SQL (Structured Query Language) — це команда або набір команд, що використовуються для взаємодії з базою даних. Запити дозволяють отримувати, додавати, оновлювати або видаляти дані в базі даних. Запит виконується для виконання цих інструкцій. Окрім повернення результатів, які можна сортувати, групувати або фільтрувати, за допомогою запиту також можна створювати, видаляти, копіювати або змінювати дані. Існує немало різних видів запитів, але самі прості і найбільш використовувані – це запити на вибірку[15]. Отже, можна відокремити такі типи запитів:

1. Запит на вибірку. Витягує дані з однієї або декількох таблиць і результати відображає в об'єкті в режимі таблиці, в якому допускається зміна записів (при деяких обмеженнях).

2. Запит з параметрами - це запит, при запуску якого відкривається діалогове вікно із запрошенням ввести певні відомості, наприклад умови відбору записів або значення для вставки в поле.

3. Перехресний запит відображає результати статистичних розрахунків (такі, як суми, кількість записів і середні значення), виконаних за даними з одного поля.

4. SQL - це запит, який створюється за допомогою інструкції SQL. Прикладами запитів SQL є запит на об'єднання, запит до серверу, керуючий запит і підлеглий запит.

5. Запит на зміни - це запит, який дозволяє, виконавши одну операцію, ввести зміни до багатьох записів. Існує чотири типи запитів на зміну: на створення таблиці, на видалення записів, на додавання і на оновлення записів.

SQL використовується в більшості реляційних баз даних, таких як:

- MySQL
- PostgreSQL
- Oracle

- Microsoft SQL Server
- SQLite

Ці бази даних застосовують SQL для взаємодії з даними, і знання SQL є обов'язковим для роботи з ними.

Переваги SQL:

- Стандартність: SQL є стандартом, що дозволяє переносити знання між різними системами управління базами даних (СУБД).
- Могутність: SQL підтримує складні операції, зокрема агрегації, підзапити, джоіни, транзакції.
- Зрозумілість: SQL є декларативною мовою, що означає, що користувач описує що він хоче отримати (замість того, щоб вказувати як це зробити, як у мовах програмування).

SQL є потужним інструментом для роботи з реляційними базами даних. З його допомогою можна ефективно взаємодіяти з великими обсягами даних, виконуючи різноманітні операції: від простих вибірок до складних маніпуляцій зі структурою таблиць та виконанням агрегацій. Освоєння SQL дає змогу здійснювати гнучку та ефективну роботу з базами даних[13].

Для створення бази даних веб додатку MilitaryGram використано реляційну базу даних MySQL.

2.2. Поняття реляційної моделі бази даних

Реляційна база даних - база даних, основана на реляційній моделі даних, тобто такі, що складаються із таблиць. Для роботи з реляційними базами даних застосовують реляційні СУБД. Інакше кажучи, реляційна база даних - це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеню. Необхідна умова реляційності бази даних - вимога відсутності співпадаючих записів. У реляційній моделі досягається більш високий рівень абстракції даних, ніж в ієрархічній або мережевій.

Реляційна база даних являє собою безліч взаємопов'язаних таблиць, кожна

з яких містить інформацію про об'єкти певного виду. Кожен рядок таблиці містить дані про один об'єкт (наприклад, автомобілі, комп'ютери), а стовпці таблиці містять різні характеристики цих об'єктів - атрибути (наприклад, номер двигуна, марка процесора).

Рядки таблиці називаються записами. Усі записи таблиці мають однакову структуру - вони складаються з полів (елементів даних), в яких зберігаються атрибути об'єкта.

Кожне поле запису містить одну характеристику об'єкта і являє собою заданий тип даних (наприклад, текстовий рядок, число). Для ідентифікації записів використовується первинний ключ. Первинним ключем називається набір полів таблиці, комбінація значень яких однозначно визначає кожний запис у таблиці. Якщо ключ складається з кількох полів, він називається складеним. Ключ повинен бути унікальним і однозначно визначати запис. За значенням ключа можна відшукати єдиний запис. Ключі служать також для впорядкування інформації в базі даних[12].

Основні поняття реляційних баз даних:

1. *Таблиця*: це основна одиниця зберігання даних, яка містить рядки (записи) та стовпці (поля). Кожна таблиця має своє ім'я.
2. *Запис (рядок):* кожен рядок таблиці представляє окремий набір значень для кожного стовпця.
3. Поле (стовпець): кожен стовпець таблиці визначає тип даних для певного виду інформації. Наприклад, "Ім'я", "Дата народження", "Адреса" - це типові стовпці в таблиці користувачів.
4. *Ключ:*
    - *Первинний ключ (Primary Key):* унікальний ідентифікатор кожного запису в таблиці. Він не може містити пустих значень.
    - *Зовнішній ключ (Foreign Key):* використовується для встановлення зв’язків між таблицями. Це поле, яке містить значення з іншої таблиці, що дозволяє здійснювати зв'язки між даними.

5. *Зв'язки між таблицями:* це механізм, який дозволяє пов'язувати дані з різних таблиць. Зв'язки можуть бути:
   - *Один до одного (1:1):* кожен запис в одній таблиці відповідає лише одному запису в іншій.
   - *Один до багатьох (1:M):* один запис в таблиці може бути пов'язаний з кількома записами в іншій.
   - *Багато до багатьох (M:M):* багато записів в одній таблиці можуть бути пов'язані з багатьма записами в іншій.
6. *Нормалізація*: це процес організації даних в базі для зменшення дублювання та забезпечення цілісності. Нормалізація включає декілька етапів (нормальних форм), що допомагають усунути аномалії оновлення, вставки та видалення.

Переваги реляційних баз даних:

- *Цілісність даних*: завдяки використанню ключів і обмежень забезпечується правильність і достовірність даних.
- *Гнучкість запитів*: можливість виконання складних запитів за допомогою SQL (Structured Query Language).
- *Масштабованість і продуктивність*: реляційні бази даних ефективно працюють з великими обсягами даних і дозволяють здійснювати їх швидкий пошук, оновлення та обробку.
- *Безпека*: можливість налаштування прав доступу до різних частин бази даних для різних користувачів.

Реляційні бази даних є важливим інструментом для бізнесу, науки та технологій завдяки своїй ефективності, надійності та зручності в управлінні великими обсягами структурованих даних.

У MySQL реляційній базі даних існує кілька основних типів даних, які використовуються для збереження різних видів інформації. Вони поділяються на кілька категорій залежно від типу даних.

Основні типи даних в MySQL: числові, для зберігання рядків(символьні типи), типи для зберігання дати та часу, логічні типи, для зберігання бінарних даних, спеціальні типи, для зберігання географічних координат[14].

Для створення реляційної бази даних, використано MySQL WorkBanch.

MySQL Workbench - це офіційний графічний інтерфейс для роботи з базами даних MySQL. Це потужний інструмент, який надає зручний користувацький інтерфейс для адміністрування, розробки та проектування баз даних. Він об'єднує кілька корисних функцій для роботи з MySQL:

Основні функції MySQL Workbench:

1. Проектування бази даних:
    - ER-діаграми (Entity-Relationship Diagrams) для візуального проектування схеми бази даних.
    - Створення таблиць, індексів, зовнішніх ключів, обмежень, взаємозв'язків між таблицями та інше через зручний інтерфейс.
2. Управління базами даних:
    - Підключення до різних MySQL серверів.
    - Створення, видалення, збереження, імпорт та експорт баз даних.
    - Взаємодія з таблицями, запитами, процедурами, тригерами та іншими об'єктами бази даних.
    - Відображення та редагування даних у таблицях.
3. SQL-редактор:
    - Можливість написання та виконання SQL-запитів, з автоматичним підсвічуванням синтаксису.
    - Інтерактивне виконання SQL-скриптів.
    - Виконання складних запитів для маніпуляцій з базами даних.
    - Перевірка та профілювання продуктивності запитів (через аналіз виконання запитів).
4. Адміністрування серверу:
    - Налаштування та моніторинг сервера MySQL: перегляд логів, ресурсів, статистики роботи.

- Керування користувачами, привілеями та правами доступу.
- Резервне копіювання та відновлення баз даних.
- Управління реплікацією, налаштуванням сервера, розподіленими середовищами.

5. Міграція та експорт/імпорт даних:
    - Міграція баз даних з інших систем управління базами даних, таких як Microsoft SQL Server, PostgreSQL тощо.
    - Імпорт/експорт даних у різні формати (CSV, SQL, XML).
6. Інтерфейс для адміністраторів:
    - Перегляд і налаштування параметрів сервера.
    - Зручне налаштування облікових записів користувачів і їх прав.
    - Моніторинг продуктивності серверу в реальному часі (використання пам'яті, процесора, з'єднання тощо).

Переваги MySQL Workbench:

- Інтуїтивно зрозумілий інтерфейс: навіть для новачків він є зручним і дозволяє ефективно працювати з базами даних.
- Єдина платформа для розробки і адміністрування: об'єднує багато корисних інструментів у одному пакеті.
- Підтримка кількох версій MySQL: підтримує як старі версії MySQL, так і новітні релізи.
- Інтеграція з іншими інструментами: можна використовувати разом з іншими інструментами для розробки, такими як Git чи сторонні редактори.

MySQL Workbench працює як десктопний додаток, який підключається до вашого MySQL сервера через мережу. Після підключення ви можете виконувати запити, управляти базами даних, моніторити сервер і здійснювати інші операції без необхідності роботи через командний рядок.

MySQL Workbench - це комплексне рішення для розробників і адміністраторів MySQL, яке значно полегшує роботу з базами даних завдяки потужним інструментам для проектування, адміністрування, моніторингу та оптимізації баз

даних. Це популярний вибір для багатьох, хто працює з MySQL, оскільки він дозволяє значно спростити управління та роботу з базами даних.

Для створення бази даних, використав наступні типи даних:

1. Числові (ці типи використовуються для зберігання чисел):
   - *BIGINT* - велике ціле число (від -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807 зі знаком або від 0 до 18,446,744,073,709,551,615 без знаку);
   - *INT (або INTEGER*) - ціле число. Може бути зі знаком (від -2,147,483,648 до 2,147,483,647) або без знаку (від 0 до 4,294,967,295).
2. Символьні(ці типи використовуються для зберігання текстових значень):
   - *TEXT* - текстовий тип для великих обсягів даних (до 65,535 символів);
   - *VARCHAR* - змінна довжина рядка (від 0 до 65,535 символів). Це більш ефективно для зберігання рядків різної довжини.
3. Типи для зберігання дати та часу(ці типи використовуються для роботи з датами, часом та їх комбінаціями):
   - *DATETIME* - дата та час (рік, місяць, день, година, хвилина, секунда) у форматі YYYY-MM-DD HH:MM:SS.
4. Типи для зберігання бінарних даних(ці типи використовуються для зберігання бінарних даних, таких як зображення, файли тощо.):
   - *LONGBLOB* - для великих бінарних об'єктів (до 4,294,967,295 байт).

Типи даних використовуються для визначення структури таблиць у базі даних, і правильний вибір типу допомагає оптимізувати зберігання даних та виконання запитів[16].

Враховуючи вище сказане, можемо представити базу даних додатку “MilitaryGram” у вигляді схеми бази даних (рис. 2.1).

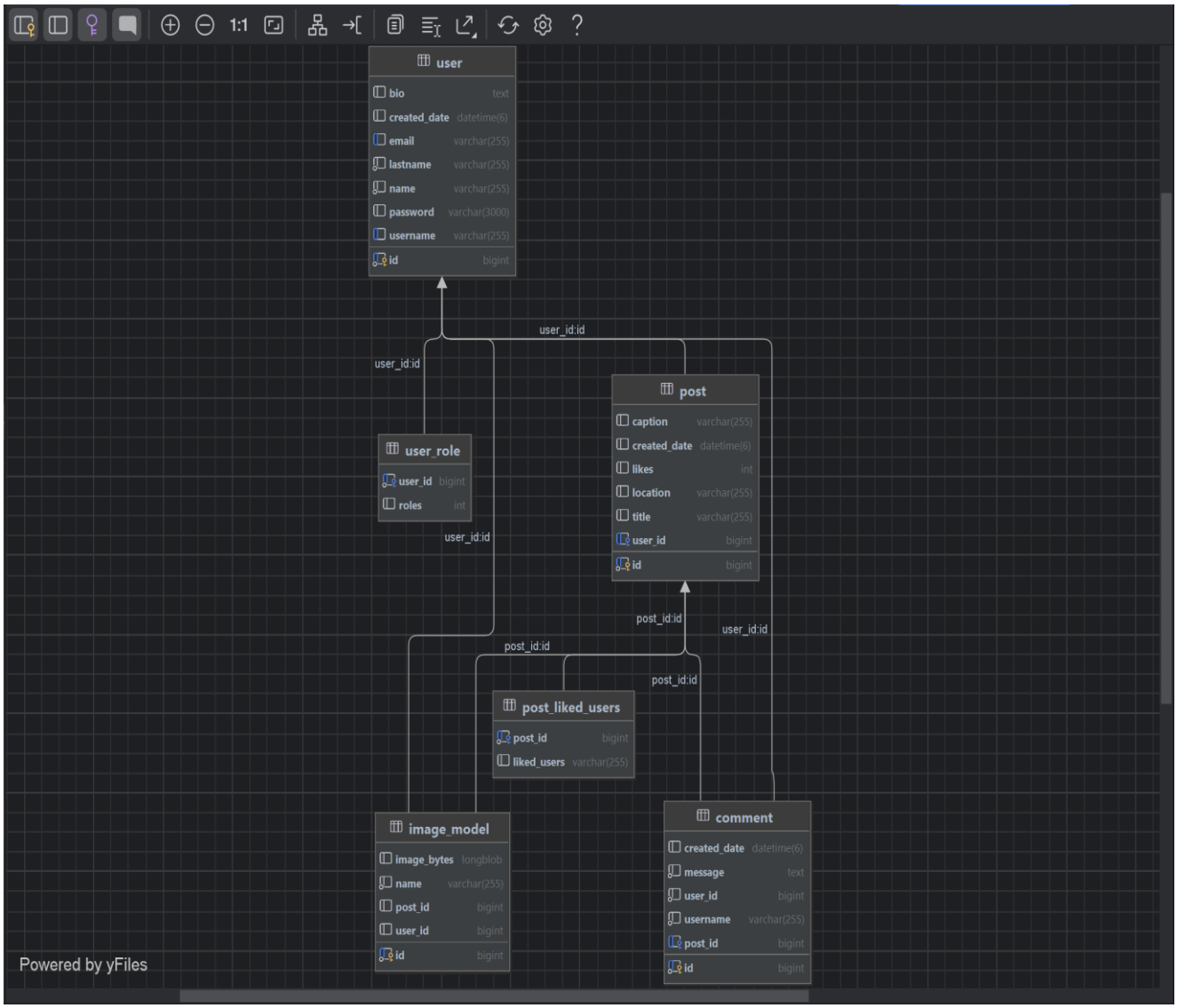
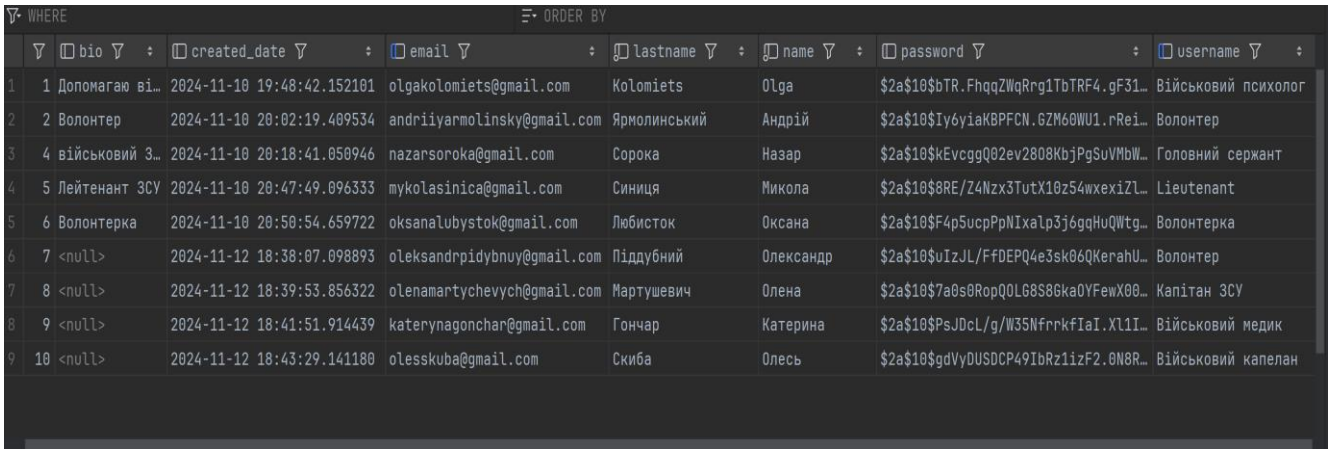


Рис. 2.1 – UML діаграма бази даних «MilitaryGram»

Визначення функціональних вимог та створення схеми бази даних дозволяє перейти до наступного етапу розробки додатку, а саме до проектування бази даних.

В СУБД MySQL створимо наступні таблиці за наступними характеристиками:

Користувач. Таблиця, що містить дані про ідентифікатор користувача, логін, пароль, ім’я та прізвище, біографію, дата створення сторінки (рис. 2.2).

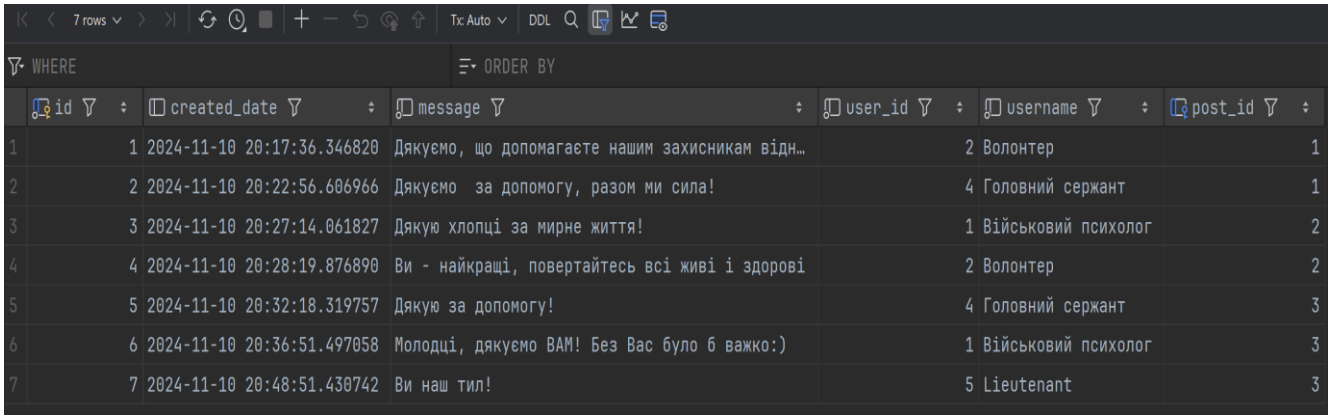


WHERE ORDER BY

	id	bio	created_date	email	lastname	name	password	username
1	1	Допомагаю ві…	2024-11-10 19:48:42.152101	olgakolomiets@gmail.com	Kolomiets	Olga	$2a$10$bTR.FhqqZWqRrg1TbTRF4.gF31…	Військовий психолог
2	2	Волонтер	2024-11-10 20:02:19.409534	andriiyarmolinsky@gmail.com	Ярмолинський	Андрій	$2a$10$Iy6yiaKBPFCN.GZM60WU1.rRei…	Волонтер
3	4	військовий 3…	2024-11-10 20:18:41.050946	nazarsoroka@gmail.com	Сорока	Назар	$2a$10$kEvcggQ02ev28O8KbjPgSuVMbW…	Головний сержант
4	5	Лейтенант ЗСУ	2024-11-10 20:47:49.096333	mykolasinica@gmail.com	Синиця	Микола	$2a$10$8RE/Z4Nzx3TutX10z54wxexiZl…	Lieutenant
5	6	Волонтерка	2024-11-10 20:50:54.659722	oksanalubystok@gmail.com	Любисток	Оксана	$2a$10$F4p5ucpPpNIxalp3j6gqHuQWtg…	Волонтерка
6	7	<null>	2024-11-12 18:38:07.098893	oleksandrpidybnuy@gmail.com	Піддубний	Олександр	$2a$10$uIzJL/FfDEPQ4e3sk06QKerahU…	Волонтер
7	8	<null>	2024-11-12 18:39:53.856322	olenamartychevych@gmail.com	Мартушевич	Олена	$2a$10$7a0s0RopQOLG8S8GkaOYFewX00…	Капітан ЗСУ
8	9	<null>	2024-11-12 18:41:51.914439	katerynagonchar@gmail.com	Гончар	Катерина	$2a$10$PsJDcL/g/W35NfrrkfIaI.Xl1I…	Військовий медик
9	10	<null>	2024-11-12 18:43:29.141180	olesskuba@gmail.com	Скиба	Олесь	$2a$10$gdVyDUSDCP49IbRz1izF2.0N8R…	Військовий капелан

Рис. 2.2 – Структура таблиці «Користувач»

Коментар. Таблиця, що містить ідентифікатор коментаря, меседж коментаря, дату створення, власник коментаря, ідентифікатор користувача, ідентифікатор посту (рис. 2.3).



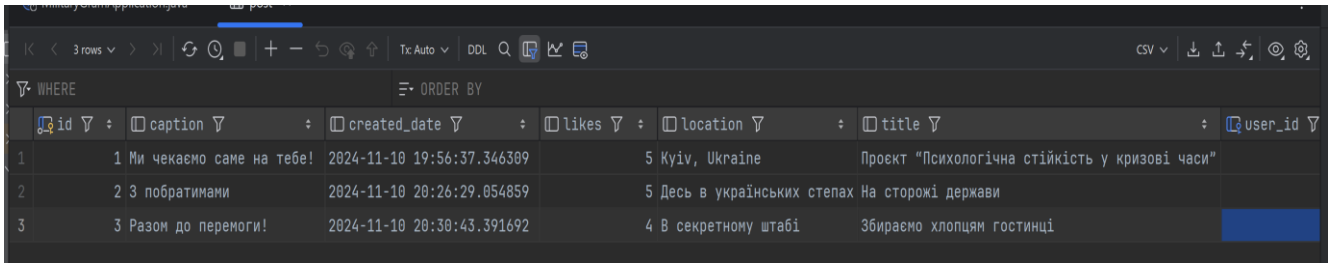
7 rows Tx: Auto DDL

WHERE ORDER BY

	id	created_date	message	user_id	username	post_id
1	1	2024-11-10 20:17:36.346820	Дякуємо, що допомагаєте нашим захисникам відн…	2	Волонтер	1
2	2	2024-11-10 20:22:56.606966	Дякуємо за допомогу, разом ми сила!	4	Головний сержант	1
3	3	2024-11-10 20:27:14.061827	Дякую хлопці за мирне життя!	1	Військовий психолог	2
4	4	2024-11-10 20:28:19.876890	Ви - найкращі, повертайтесь всі живі і здорові	2	Волонтер	2
5	5	2024-11-10 20:32:18.319757	Дякую за допомогу!	4	Головний сержант	3
6	6	2024-11-10 20:36:51.497058	Молодці, дякуємо ВАМ! Без Вас було б важко:)	1	Військовий психолог	3
7	7	2024-11-10 20:48:51.430742	Ви наш тил!	5	Lieutenant	3

Рис. 2.3 – Структура таблиці «Коментар»

Пост. Таблиця, що містить ідентифікатор, опис, кількість вподобайок, локацію, назву, ідентифікатор користувача (рис. 2.4).



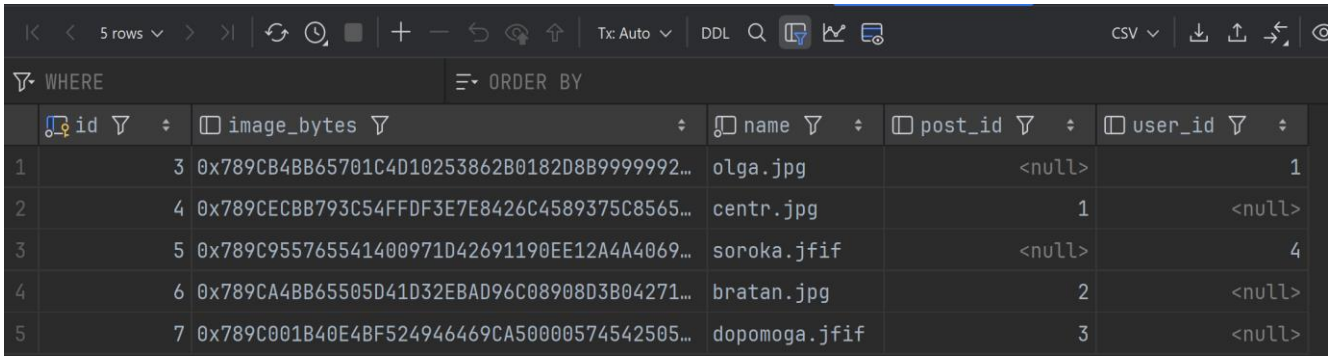
3 rows Tx: Auto DDL CSV

WHERE ORDER BY

	id	caption	created_date	likes	location	title	user_id
1	1	Ми чекаємо саме на тебе!	2024-11-10 19:56:37.346309	5	Kyiv, Ukraine	Проєкт “Психологічна стійкість у кризові часи”	
2	2	З побратимами	2024-11-10 20:26:29.054859	5	Десь в українських степах	На сторожі держави	
3	3	Разом до перемоги!	2024-11-10 20:30:43.391692	4	В секретному штабі	Збираємо хлопцям гостинці	

Рис. 2.4 – Структура таблиці «Пост»

Фотографії. Таблиця, що містить ідентифікатор, назву, файл фотографії, ідентифікатор посту або ідентифікатор користувача (рис. 2.5).



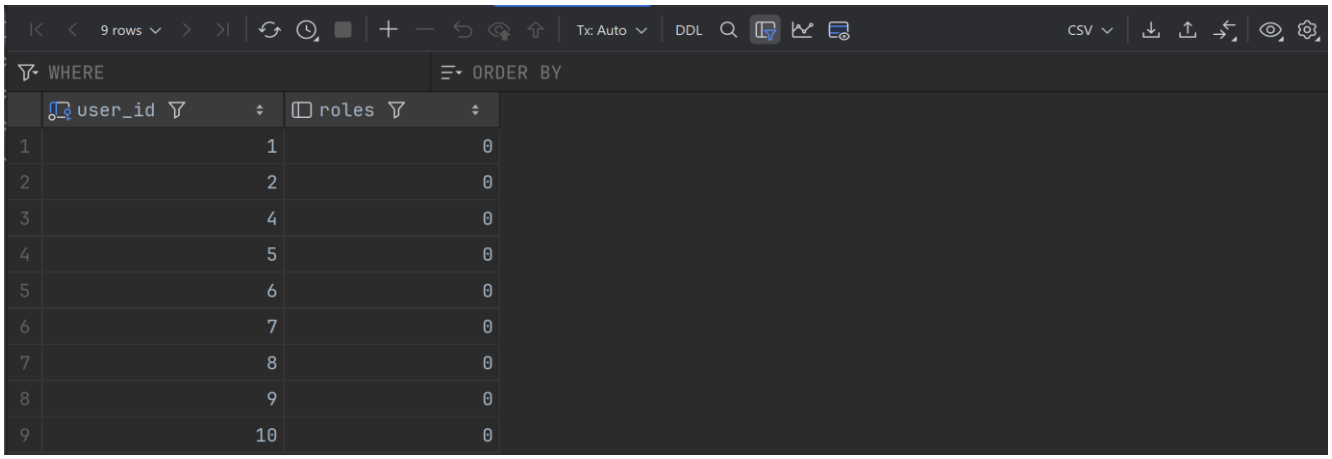
5 rows | Tx: Auto | DDL | CSV

WHERE | ORDER BY

	id	image_bytes	name	post_id	user_id
1	3	0x789CB4BB65701C4D10253862B0182D8B9999992…	olga.jpg	<null>	1
2	4	0x789CECBB793C54FFDF3E7E8426C4589375C8565…	centr.jpg	1	<null>
3	5	0x789C955765541400971D42691190EE12A4A4069…	soroka.jfif	<null>	4
4	6	0x789CA4BB65505D41D32EBAD96C08908D3B04271…	bratan.jpg	2	<null>
5	7	0x789C001B40E4BF524946469CA50000574542505…	dopomoga.jfif	3	<null>

Рис. 2.5 – Структура таблиці «Фотографії»

Роль. Таблиця, що містить ідентифікатор користувача, ідентифікатор ролі (рис. 2.6).



9 rows | Tx: Auto | DDL | CSV

WHERE | ORDER BY

	user_id	roles
1	1	0
2	2	0
3	4	0
4	5	0
5	6	0
6	7	0
7	8	0
8	9	0
9	10	0

Рис. 2.6 – Структура таблиці «Роль»

2.3. Висновки до розділу 2

У цьому розділі було розглянуто основи проектування бази даних для веб-додатку "MilitaryGram”. Розглянуті основи роботи з SQL, реляційною моделлю бази даних та інструментами для їх розробки, зокрема MySQL Workbench.

Розглянуто основні компоненти реляційної бази даних, такі як таблиці, записи, поля, ключі, а також принципи нормалізації для зменшення дублювання та забезпечення цілісності даних.

Визначено типи даних для розуміння, як саме зберігати різні види даних у таблицях. Це важливий аспект проектування, адже правильний вибір типу даних дозволяє оптимізувати роботу з базою даних і підвищити ефективність її функціонування. Розглянуто числові, символьні, часо- та датні типи даних, а також типи для зберігання бінарних даних, таких як зображення.

MySQL Workbench був вибраний як основний інструмент для проектування бази даних завдяки його зручному інтерфейсу, багатофункціональності та інтеграції з MySQL. Цей інструмент дозволяє здійснювати проектування, адміністрування та виконання SQL-запитів, що є важливими етапами в розробці ефективної бази даних для веб-додатку.

РОЗДІЛ 3
ПОБУДОВА СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМИ

3.1. Побудова REST-сервісів за допомогою Spring

REST швидко став фактичним стандартом для створення веб-сервісів в Інтернеті, оскільки їх легко створювати та легко використовувати.

REST слідує канонам вебу, включаючи його архітектуру, переваги та все інше. Це не дивно, враховуючи, що його автор Рой Філдінг брав участь, мабуть, у десятку специфікацій, які визначають, як працює веб.

Мережа та її основний протокол HTTP забезпечують безліч функцій:

- відповідні дії (GET, POST, PUT, DELETE,…);
- кешування;
- перенаправлення та переадресація;
- безпека (шифрування та автентифікація).

Це все є критичними факторами для створення стійких служб. Але це ще не все. Веб побудований з безлічі крихітних специфікацій, отже, він міг легко розвиватися, не заглиблюючись у "стандартні війни".

Розробники можуть спиратись на сторонні набори інструментів, які реалізують ці різноманітні характеристики, і миттєво мають як клієнтську, так і серверну технологію під рукою.

Будуючи поверх HTTP, REST API надають засоби для побудови:

- зворотно-сумісні API;
- еволюційні API;
- масштабовані сервіси;
- безпечні послуги.

Спектр сервісів, які зберігають та не зберігають стан. Важливо усвідомити, що REST, не є стандартом сам по собі, а підходом, стилем та набором обмежень для вашої архітектури, які можуть допомогти вам побудувати веб-масштабні системи. У даній роботі використано Spring Boot, щоб створити сервіси RESTful,

використовуючи безстекові функції REST[18].

Для цього, перейшовши на офіційну сторінку, відкривши вкладку Spring Initializr і додавши до проекту такі залежності: Spring Boot 2.4.1, Spring Data JPA, Spring Data REST, Spring Security, Validation I/O, MySQL Driver, Lombok. В нас сформується архів, розархівувавши, який ми отримуємо готовий зібраний проект на основі засобу автоматизації проектів Maven.

Потім ми створюємо наші сутності (розглянемо на прикладі однієї сутності). @Entity - це анотація JPA, яка робить цей об’єкт готовим до зберігання у сховищі даних на основі JPA. Ідентифікатор, ім'я та роль - це атрибути нашого об'єкта домену. Id позначається анотацією JPA, щоб вказати, що це первинний ключ, і автоматично заповнюється постачальником JPA[17].

Кастомний конструктор створюється, коли нам потрібно створити новий екземпляр, але ще не маючи ідентифікатора.

Завдяки цьому визначенню об’єкта домену, ми тепер можемо звернутися до Spring Data JPA, щоб впоратись з нудною взаємодією з базою даних.

Репозиторії Spring Data JPA - це інтерфейси з методами, що підтримують створення, читання, оновлення та видалення записів із внутрішнього сховища даних. Деякі сховища також підтримують підкачування та сортування даних, де це доречно. Spring Data синтезує реалізації на основі імплементацій, знайдених в іменуванні методів в інтерфейсі. Spring полегшує доступ до даних[20]. Просто оголосивши наступний інтерфейс UserRepository зможемо:

- створити нових юзерів;
- оновити існуючі;
- видалити юзерів;
- пошук юзерів (одного, усіх або пошук за простими або складними властивостями).

Щоб отримати всю цю функціональність автоматично, нам потрібно оголосити інтерфейс, який розширює JpaRepository Spring Data JPA, вказавши тип домену як User, а тип ідентифікатора як Long.

Рішення сховища Spring Data дає змогу обійти специфіку сховища даних і

замість цього вирішити більшість проблем, використовуючи специфічну для домену термінологію.

Додаток Spring Boot - це, як мінімум, public static void основна точка входу та анотація @SpringBootApplication. @SpringBootApplication - це мета-анотація, яка включає сканування компонентів, автоконфігурацію та підтримку властивостей. По суті, вона запустить контейнер сервлетів і обслуговуватиме наш сервіс[21].

HTTP Платформа. Щоб обернути репозиторій веб-шаром, потрібно звернутися до Spring MVC. Завдяки Spring Boot це працює з коробки. @RestController вказує, що дані, що повертаються кожним методом, будуть записані прямо в тіло відповіді, замість того, щоб відображати шаблон. Репозиторій UserRepository інжектиться конструктором в контролер[22].

У нас є маршрути для кожної операції (@GetMapping, @PostMapping, @PutMapping і @DeleteMapping, що відповідає викликам HTTP GET, POST, PUT та DELETE).

За допомогою однієї додаткової бібліотеки та кількох рядків додаткового коду ми додали підтримку гіпермедіа до своєї програми. Але це не єдине, що потрібно, щоб зробити наш сервіс RESTful. Важливою стороною REST є той факт, що він не є ні стеком технологій, ні єдиним стандартом.

REST - це сукупність архітектурних обмежень, які після їх використання роблять вашу програму набагато стійкішою. Ключовим фактором стійкості є те, що коли ви оновлюєте свої послуги, ваші клієнти не страждають від простою.

Іншими словами, оновлення для сервера вимагало оновлення для клієнта. У цей день вік, години чи навіть хвилини простоїв, проведених для оновлення, можуть коштувати мільйонів втрачених доходів.

Сервіси на основі SOAP та CORBA були неймовірно крихкими. Важко було розвернути сервер, який міг би підтримувати як старих, так і нових клієнтів. З практиками на основі REST це набагато простіше. Особливо використовуючи стек Spring.

Як виявляється, REST - це не лише симпатичні URI та повернення JSON

замість XML.

Натомість наступні тактики допомагають зменшити ймовірність того, що наші сервіси зламають поведінку існуючих клієнтів, яких ми можемо контролювати чи не зламають.

@GetMapping («{todoId}») показує, що метод getTodo повинен обробляти GET-запти по шляху / api / todo / {todoId}. Фігурні скобки в даному випадку вказують, що todoId - перехідний шлях(змінна шляху) і може бути використана в аргументах методу. Можливо задати більш строгі правила валідації шляху, додавши після todoId через дві крапки регулярний вираз, який буде використано для валідації шляху.

@PathVariable показує, що значення аргументу todoId повинно братися з запиту шляху. За умовою всі значення є строковими, але при допомозі класів-конвертерів строки перетворюються в об'єкт класу java.util.UUID. Ім'я аргументу відповідає плейсхолдеру, вказаному в @GetMapping, у протилежному випадку нам треба вказати плейсхолдер у властивості name анотації @PathVariable.

Виключна ситуація IncorrectResultSizeDataAccessException виникає в разі, якщо база даних повернула у відповідь на запит кількість строк нерівне 1. Якщо актуальне кількість строк рівна 0, щоб відповідний запис у БД відсутній, і ми можемо повернути відповідь з HTTP-кодом 404 Not Found; якщо строк більше 1, то ми маємо справу з помилкою та можемо вернути відповідь з HTTP-кодом 500 Internal Server Error.

Анотація @PostMapping вказує, що даний метод обробляє всі POST-запити з шляхом / api / todo. Анотація @RequestBody вказує, що позначений аргумент є тілом запиту. Тіло запиту перетворюється в об'єкт нового класу за допомогою конверторів.

Тіло запиту автоматично перетворюється в об'єкт типу TodoPayload за допомогою класу-кодека.

Метод видалення запису повинен бути викликаний при DELETE-запиті за адресою / api / todo / {todoId}; він повинен видалити запис з БД і повернути відповідь із статусом 204 Немає вмісту, або вернути відповідь із статусом 404 *No*

Content, якщо запис не знайдений.

В нашому застосунку є такі сутності як: User, Post, Comment, ImageModel. Контролери: UserController, PostController, CommentController, ImageUploadController, AuthController. Сервіси: UserService, PostService, ImageUploadService, CommentService, CustomUserDetailService. Репозиторії: UserRepository, PostRepository, ImageRepository, CommentRepository.

Враховуючи вище сказане можна вказати ендпоінти для нашого додатку, які потрібно реалізувати.

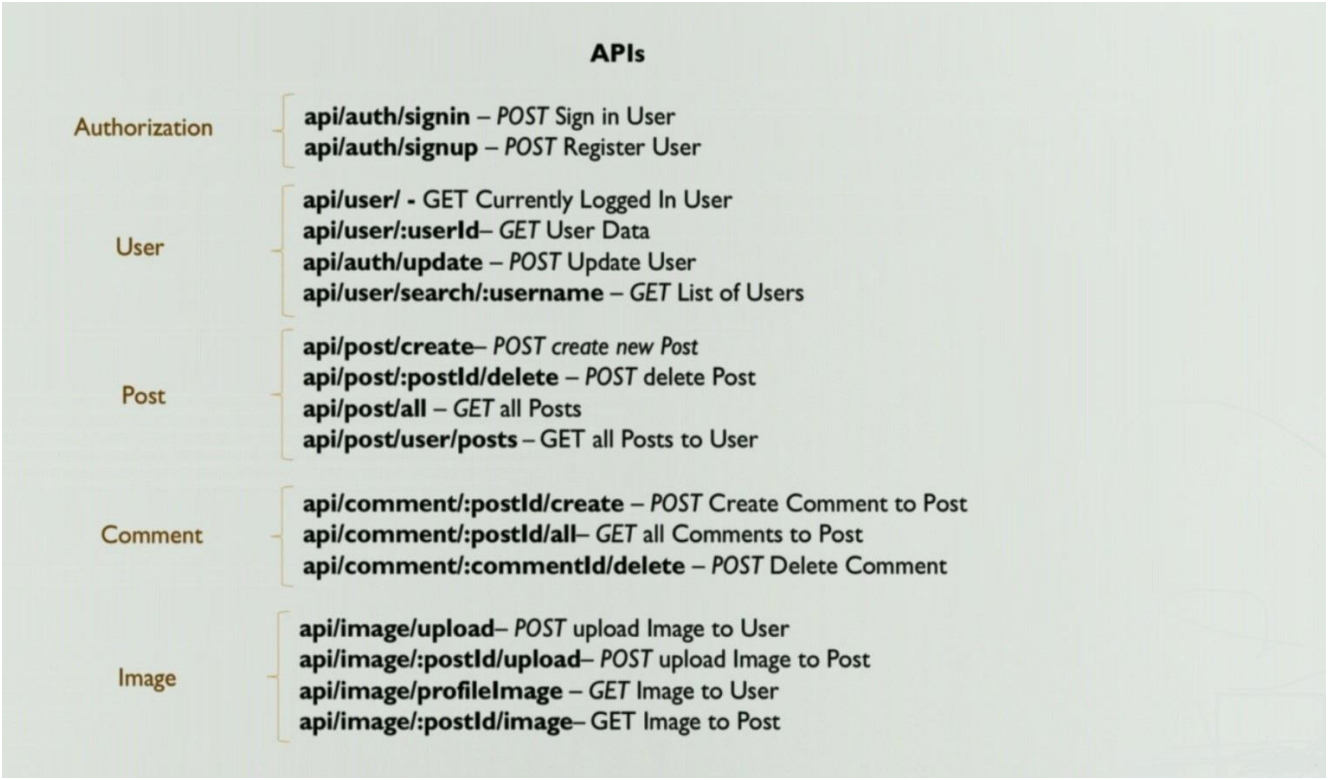


Рис. 3.1 – Енд поінти веб додатку

3.2. Реалізація аутентифікації та авторизації

У сучасному світі, де дані стали ключовим ресурсом, забезпечення їхньої безпеки стає дедалі важливішим. Один з основних аспектів цього – правильне розуміння процесів аутентифікації та авторизації. У даній роботі ми розберемо, яка різниця між авторизацією та автентифікацією і що вони собою являють.

Щоб наочно зрозуміти важливість цих процесів, уявімо собі ситуацію, коли несанкціонований користувач отримує доступ до банківського акаунту через

недостатню аутентифікацію. Це може призвести до фінансових втрат, крадіжки особистих даних і навіть загрози безпеці. Саме тому правильна аутентифікація та авторизація є невід'ємною частиною захисту наших особистих і фінансових інтересів в онлайн-світі.

Аутентифікація – це процедура перевірки та підтвердження особи користувача перед наданням доступу до певної системи, додатку або ресурсу. Її мета полягає в тому, щоб переконатися, що користувач, яка намагається отримати доступ, дійсно є тим, за кого він себе видає. Це забезпечує безпеку, запобігає несанкціонованому доступу та захищає конфіденційні дані від несанкціонованого використання.

Процес аутентифікації починається з того, що користувач надає системі унікальну інформацію, яка ідентифікує його, наприклад, логін і пароль. Після введення цих даних система перевіряє їх наявність у базі даних. Якщо введені дані відповідають даним у системі, користувач вважається успішно аутентифікованим і отримує доступ. У разі помилки, користувачеві може бути відмовлено в доступі.

Існує кілька методів такої перевірки, призначених для підвищення безпеки:

1. Введення логіна і пароля: від користувача вимагається ввести унікальний логін і пароль, які пов'язані з його акаунтом. Це один із найпоширеніших способів аутентифікації.
2. Багатофакторна аутентифікація (MFA): користувач повинен надати кілька форм підтвердження своєї особи. Наприклад, після введення логіна і пароля, він може отримати одноразовий код на свій телефон або електронну пошту, який також необхідний для входу.
3. Біометричні дані: цей метод уже використовує унікальні фізіологічні або поведінкові характеристики користувача, такі як відбитки пальців, розпізнавання обличчя або голосу. Має високий рівень безпеки, оскільки біометричні дані складно підробити або вкрасти.
4. Перевірка через соціальні мережі: дає змогу користувачам увійти на сайт або в застосунок, використовуючи свої облікові дані із соціальних мереж, як-от Facebook або Google.

Вибір конкретного методу залежить від рівня безпеки, який потрібен для конкретної системи, і зручності використання для кінцевого користувача.

Авторизація , з іншого боку, пов’язана з визначенням того, до яких ресурсів або функціональності користувач має доступ. Після успішної перевірки система визначає рівень доступу користувача: які файли він може бачити, які дії він може виконувати тощо. Це включає визначення рівнів доступу, дозволів на виконання певних операцій і тимчасових обмежень на доступ до ресурсів.

Цей процес включає в себе такі аспекти:

- Рівні доступу: різні користувачі можуть мати різні рівні доступу в системі. Наприклад, у системі управління проектами звичайні користувачі можуть мати доступ лише до читання даних, тоді як адміністратори мають повний доступ до редагування та видалення інформації. Або, скажімо, у системі електронної медичної документації, лікарі можуть мати вищий рівень доступу, ніж медичні сестри чи адміністративний персонал.
- Дозволи на доступ: перевірка також визначає конкретні дії, які користувач може здійснювати. Це можуть бути дозволи на читання, запис, зміну або видалення певних даних. Наприклад, в онлайн-магазинах, адміністратори мають дозвіл на управління асортиментом товарів, тоді як звичайні користувачі можуть лише переглядати товари та робити замовлення.
- Часові обмеження: Мистецтво про тимчасові рамки доступу. Наприклад, деякі користувачі можуть мати доступ тільки в певний час доби або в певні дні тижня.

Цей процес перевірки не тільки забезпечує безпеку даних, а й допомагає ефективно використовувати ресурси, надаючи користувачам тільки ті можливості, які необхідні для їхньої роботи, і встановлюючи необхідні обмеження для підтримки безпеки системи.

Основна відмінність між обома процесами полягає в тому, що перша підтверджує особу користувача, тоді як друга визначає, що цей користувач може робити після того, як його особу підтверджено. Розуміння цієї відмінності є ключовим, бо без правильної аутентифікації авторизація стає безглуздою – система

не зможе переконатися в тому, що користувач із правильними дозволами справді той, за кого себе видає.

Розглянемо табличку, в якій детально розглядаються основні відмінності та схожості:

Таблиця 3.1 – Порівняння понять авторизація та аутентифікація

Аспект	Аутентифікація	Авторизація
Визначення	Перевірка особистості користувача	Керування доступом користувача до ресурсів після перевірки.
Ціль	Посвідчення, що користувач є тим, за кого він себе видає.	Керування діями та доступом користувача в системі.
Процес	Підтвердження особи через введення даних (логіна, пароля), біометричні дані тощо.	Визначення дозволів і рівня доступу користувача до ресурсів.
Фокус	Посвідчення особи користувача.	Визначення прав доступу та контроль над функціональністю.
Ключові моменти	Логін, пароль, біометричні дані, одноразові коди тощо.	Рівні доступу, дозволи на дії, тимчасові обмеження тощо.
Припускає	Попередню аутентифікацію для визначення особи.	Успішну аутентифікацію для встановлення доступу до ресурсів.
Чому важливо	Запобігає несанкціонованому доступу до системи та даних.	Контролює, що користувач може робити в системі після аутентифікації.

Продовження таблиці 3.1

Аспект	Аутентифікація	Авторизація
Приклад у житті	Введення пароля під час входу в акаунт, використання відбитків пальців або розпізнавання обличчя.	Після входу в пошту, визначення, чи може користувач писати, читати, видаляти листи.
Схожості	Обидва процеси пов’язані з безпекою даних і контролем доступу.	Обидва процеси працюють у парі, визначаючи, хто має доступ до чого.

Правильне розмежування цих процесів підтримує приватність користувачів і відповідність зі стандартами безпеки галузі. Це також забезпечує зручність для кінцевих користувачів, дозволяючи їм отримувати доступ до потрібних ресурсів без надлишкових запитів на підтвердження особи.

Наведемо приклади систем, де аутентифікація та авторизація відіграють ключову роль:

- *Банківські системи*: при доступі до онлайн-банкінгу або мобільних банківських додатків, користувачі спочатку автентифікують себе, вводячи логін і пароль. Після успішної перевірки система визначає, які операції (авторизації) користувач може виконувати, такі, як переказ коштів, оплата рахунків або перегляд балансу.
- *Соціальні мережі*: коли ви входите до свого акаунта в соціальній мережі, ви проходите аутентифікацію, зазвичай вводячи логін і пароль. Потім система визначає, які профілі та повідомлення ви можете бачити або редагувати, а також яку інформацію ви можете публікувати.
- *Електронна пошта:* при вході в поштову скриньку ви проходите аутентифікацію. Потім, на основі авторизації, система дає змогу вам читати, писати, видаляти та надсилати повідомлення.

Чому це важливо в повсякденному житті? Обидві системи забезпечують безпеку особистих даних, підтримують конфіденційність приватної інформації, дають змогу системам контролювати доступ і забезпечують зручність та ефективність у використанні різних сервісів і систем[25].

В веб-додатку “MilitaryGram”, мною було реалізовано аутентифікацію та авторизацію за допомогою використання JSON Web Token на основі фільтрів фреймворку SpringSecurity.

JWT (JSON Web Token) - це стандарт, що використовується для безпечної передачі даних між двома сторонами у вигляді компактного та URL-безпечного токена. У контексті Spring Security JWT часто застосовується для аутентифікації та авторизації, дозволяючи клієнтам аутентифікуватися за допомогою токенів замість традиційних сесій[19].

Основні етапи використання JWT у Spring Security:

1. *Створення JWT*. Коли користувач успішно аутентифікується (наприклад, через логін і пароль), система генерує JWT, який містить дані про користувача (наприклад, його ролі, ID тощо), підписується за допомогою секретного ключа і надсилається клієнту.
2. *Передача JWT клієнту*. Токен передається клієнту у відповіді на запит аутентифікації. Клієнт зберігає цей токен, зазвичай у localStorage або cookies.
3. *Перевірка JWT при кожному запиті*. При кожному запиті, що потребує авторизації, клієнт надсилає JWT в заголовку Authorization (наприклад, як Bearer токен). Сервер перевіряє підпис токена, щоб переконатися у його дійсності та підробленості.
4. *Авторизація з використанням JWT*. На сервері Spring Security перевіряє підпис токена та витягує з нього дані про користувача, щоб дозволити чи заборонити виконання запитуваної операції.

Основні компоненти JWT у Spring Security:

1. *JWT Token Provider*. Це компонент, який генерує JWT і витягує з нього дані.

Він відповідає за створення та перевірку підпису токена.

2. *JWT Filter*. Це фільтр, який обробляє вхідні запити, витягує JWT із заголовка та перевіряє його. Якщо токен валідний, фільтр дозволяє запиту продовжити виконання, в іншому випадку - відхиляє його.
3. *Налаштування Spring Security для використання JWT*. Щоб Spring Security використовував JWT для аутентифікації, потрібно налаштувати фільтр JWT і додати його до ланцюга фільтрів.

Переваги використання JWT у Spring Security.

1. *Безпека*. JWT використовує криптографічний підпис, що забезпечує цілісність даних і захист від підробки токенів. Підпис токена гарантує, що дані в ньому не були змінені після його створення. Алгоритми підпису, як-от HS256 (HMAC з SHA-256), або асиметричні алгоритми (RSA, ECDSA), дозволяють забезпечити надійний захист.
2. *Масштабованість*. Оскільки JWT не вимагає зберігання сесій на сервері, його легко використовувати у масштабованих розподілених системах. Токен містить всю необхідну інформацію про користувача.
3. *Гнучкість*: JWT дозволяє передавати будь-які дані у вигляді "payload" і використовувати їх для різних цілей (авторизація, аудит тощо).

Недоліки JWT.

1. *Складність відкликання токенів*. Тому що токен є самодостатнім, його відкликання (наприклад, при виході користувача) може бути складним. Це вимагає додаткових механізмів, як от чорний список токенів.
2. *Розмір токенів*. JWT може бути великим, особливо якщо містить багато даних, що може впливати на ефективність передачі через мережу[23].

В таблиці 3.1 ми можемо побачити, як зберігаються дані про паролі в базі даних, які зберігаються у вигляді токенів, і без ключів, які додаток формує на основні даних користувача, дані будуть надійно захищені навіть після взлому БД.

	id	bio	created_date	email	lastname	name	password	username
1	1	Допомагаю ві…	2024-11-10 19:48:42.152101	olgakolomiets@gmail.com	Kolomiets	Olga	$2a$10$bTR.FhqqZWqRrg1TbTRF4.gF31…	Військовий психолог
2	2	Волонтер	2024-11-10 20:02:19.409534	andriiyarmolinsky@gmail.com	Ярмолинський	Андрій	$2a$10$Iy6yiaKBPFCN.GZM60WU1.rRei…	Волонтер
3	4	військовий 3…	2024-11-10 20:18:41.050946	nazarsoroka@gmail.com	Сорока	Назар	$2a$10$kEvcggQ02ev28O8KbjPgSuVMbW…	Головний сержант
4	5	Лейтенант ЗСУ	2024-11-10 20:47:49.096333	mykolasinica@gmail.com	Синиця	Микола	$2a$10$8RE/Z4Nzx3TutX10z54wxexiZl…	Lieutenant
5	6	Волонтерка	2024-11-10 20:50:54.659722	oksanalubystok@gmail.com	Любисток	Оксана	$2a$10$F4p5ucpPpNIxalp3j6gqHuQWtg…	Волонтерка
6	7	<null>	2024-11-12 18:38:07.098893	oleksandrpidybnuy@gmail.com	Піддубний	Олександр	$2a$10$uIzJL/FfDEPQ4e3sk06QKerahU…	Волонтер
7	8	<null>	2024-11-12 18:39:53.856322	olenamartychevych@gmail.com	Мартушевич	Олена	$2a$10$7a0s0RopQOLG8S8GkaOYFewX00…	Капітан ЗСУ
8	9	<null>	2024-11-12 18:41:51.914439	katerynagonchar@gmail.com	Гончар	Катерина	$2a$10$PsJDcL/g/W35NfrrkfIaI.Xl1I…	Військовий медик
9	10	<null>	2024-11-12 18:43:29.141180	olesskuba@gmail.com	Скиба	Олесь	$2a$10$gdVyDUSDCP49IbRz1izF2.0N8R…	Військовий капелан

Рис. 3.2 – Приклад роботи JWT в таблиці бази даних

3.3. Висновки до розділу 3

У даному розділі було розглянуто важливі аспекти побудови серверної частини програми з використанням технології Spring Boot, а також реалізацію механізмів аутентифікації та авторизації користувачів у веб-додатках.

Починаючи з побудови REST-сервісів, було досліджено основні принципи REST-архітектури, яка є важливою складовою сучасних веб-систем. REST забезпечує простоту, масштабованість, зворотну сумісність та безпеку веб-сервісів. Використання Spring Boot, Spring Data JPA, Sprin Security та інших інструментів дозволяє автоматизувати процеси роботи з даними, що робить розробку ефективною та зручною. Розробка таких сервісів дозволяє створювати надійні API, що можуть бути інтегровані з іншими системами або використовуватися як окремі модулі.

У реалізації аутентифікації та авторизації ми акцентували увагу на важливості захисту даних та контролю доступу до ресурсів. Розглянуто процес аутентифікації за допомогою JWT (JSON Web Token), що є одним із найбільш поширених методів аутентифікації в сучасних веб-додатках[24]. Цей підхід дозволяє забезпечити безпеку передачі даних, оскільки використовуються криптографічні підписи для підтвердження достовірності інформації. Крім того, ми розглянули значення авторизації в контексті розподілу прав доступу, що є важливою частиною управління безпекою системи.

Використання JWT надає ряд переваг, таких як безпека, масштабованість та гнучкість, але також вимагає певних додаткових налаштувань для забезпечення

ефективного управління токенами, наприклад, механізмів відкликання токенів або налаштування додаткових фільтрів.

Загалом, даний розділ показав, як можна побудувати надійний сервер для веб-додатків, що забезпечує зручний доступ до ресурсів за допомогою REST API, а також як забезпечити належний рівень безпеки, контролюючи доступ користувачів та гарантуючи цілісність і конфіденційність даних.

РОЗДІЛ 4

ПОБУДОВА ІНТЕРФЕЙСУ ПРОГРАМИ

Для зручного використання програми було розроблено простий інтерфейс, який дає можливість користувачу швидко додати новий пост, редагувати або видалити пост, редагувати персональні дані та коментувати пости інших користувачів. Спочатку користувач має створити свій обліковий запис, для чого була створена форма Register, а також, якщо користувач вже зареєстрований, то він заповнює форму Login та заходить у свій персональний кабінет.

Для програмної реалізації було вирішено використатиAngular та Bootstrap.

Angular – крос-платформений фреймворк від компанії Google для створення mobile-first та desktop-first додатків. Він розрахований на створення SPA-рішень (Single Page Application), тобто односторінкових додатків. Angular дозволяє створювати великі і складні у частині бізнес-логіки додатки. У якості мови програмування Angular використовує TypeScript[26].

Базовий будівельний блок Angular додатків – це Angular компоненти, які організовані у Ng-модулі. Кожний додаток моє один головний модуль, який завантажується першим, а також декілька модулів, що надають додатку конкретний функціонал. Компоненти визначають представлення, які містять набір екранних елементів. Компоненти використовують сервіси, які виконують специфічний функціонал, не пов’язаний з представленнями. Постачальники сервісів можуть бути вбудовані (injected) у компоненти у якості підлеглих модулів (dependencies), що робить код модульним, ефективним і цей код можна використовувати повторно[27].

Модулі, компоненти і сервіси є класами, що використовують так звані декоратори. Декоратори позначають свій тип і надають метадані, які підказують Angular, як їх використовувати. Метадані для класу-компоненту пов’язують його з шаблоном, який визначає представлення. Шаблон поєднує звичайний

HTML з директивами та маркуванням, що дозволяє Angular видозмінювати HTML перед виводом на екран. Метадані для класу-сервісу дозволяють використовувати його у компонентах через впровадження залежностей (dependency injection)[28].

Компоненти програми зазвичай визначають багато представлень, що впорядковані ієрархічно. Angular також має сервіс роутера (router), який надає навігаційні адреси представленням.

Ng-модулі Angular відрізняються від і доповнюють JavaScript модулі. Кожний Angular додаток має кореневий модуль, що традиційно називається AppModule, який запускається при старті додатку. Додаток зазвичай має багато функціональних модулів[29].

Як і JavaScript модулі, Ng-модулі можуть імпортувати функціональність інших модулів та дозволяють експортувати власну функціональність для використання у інших модулях.

Організація коду у окремі функціональні модулі допомагають розробляти складні додатки та використовувати код повторно. Також ця техніка дозволяє використовувати завантаження за вимогою, тобто завантажувати лише мінімально необхідний код при старті додатку.

Кожний додаток Angular має принаймні один компонент, що називається корінним, який поєднує компонентну ієрархію з об’єктною моделлю документа (DOM). Кожний компонент визначає клас, який містить дані та логіку додатку, та якому відповідає HTML шаблон, який визначає представлення, яке буде показане у кінцевому середовищі[30].

Декоратор @Component() визначає клас як компонент і надає йому шаблон і метадані. Декоратори – це функції, що модифікують JavaScript класи. Angular визначає низку декораторів, які наділяють класи специфічними метаданими[31].

Шаблон поєднує HTML з маркуванням Angular і модифікує HTML елементи перед їхнім показом. Шаблонні директиви є джерелом програмної логіки, а пов’язуюча розмітка (binding markup) поєднує дані додатку і об’єктну модель (DOM). Існує два види зв’язування даних:

- зв’язування через подію, дозволяє додатку відповідати на запити користувача, оновлюючи дані додатку;
- зв’язування через властивість, дозволяє включати значення, які підраховані додатком, у HTML[32].

Перш ніж показати представлення, Angular аналізує директиви і зв’язки у шаблоні, щоб змінювати елементи HTML та DOM відповідно до даних і логіки програми. Angular підтримує двостороннє зв’язування. Це означає, що зміни у DOM, зумовлені діями користувача, також відображаються у даних додатку[33].

Шаблони можуть використовувати конвеєр (pipes), підвищуючи зручність шляхом трансформації значень. Наприклад, pipes використовуються для показу дат і грошових одиниць, які підходять для користувача.

Для даних і логіки, які не асоціюються з певним представленням та які використовуються у декількох компонентах, створюється клас-сервіс. Класу-сервісу передує декоратор @Injectable(). Декоратор містить метадані, що дозволяють впроваджувати залежності у клас[34].

Впровадження залежностей (dependency injection) дозволяє класам-компонентам мати невеликий розмір і бути продуктивними. Такі компоненти одночасно не отримують дані з серверу, не перевіряють введені користувачем дані, не виводять інформацію у консоль. Вони делегують ці завдання сервісам[35].

Роутер-модуль Angular є сервісом, завдяки якому можна задати навігаційні шляхи, які відповідають різним станам роботи додатка та ієрархіям представлень. Модуль заснований на знайомих правилах навігації у браузері:

- введіть URL у рядку адреси і браузер перейде до відповідної сторінки;
- натисніть на посилання і браузер перейде на нову сторінку;
- натисніть кнопки «вперед» і «назад» і браузер перейде вперед і назад по історії сторінок, які ви відвідували раніше.

Роутер встановлює співвідношення між URL-подібними шляхами та представленнями, а не сторінками. Коли користувач здійснює дію, таку як натискання на посилання, роутер перехвачує її і показує чи приховує представлення (view)[36].

Якщо роутер визначить, що стан веб-додатку вимагає наявності певної функціональності, а модуль, який надає цю функціональність, не був завантажений, роутер може завантажити цей модуль за потребою (lazy-load).

Роутер інтерпретує URL посилання відповідно до навігаційних правил додатку і стану даних. Можна відображати нові представлення, коли користувач натискає на посилання чи обирає значення з випадаючого меню, або у відповідь на іншу дію іншого походження. Роутер документує свою діяльність у історії браузера, тому кнопки «вперед» і «назад» працюють.

Щоб визначити навігаційні правила, необхідно пов’язати навігаційні шляхи з компонентами. Шлях використовує URL-подібний синтаксис, який інтегрує дані програми подібно до того, як шаблонний синтаксис об’єднує представлення і дані додатку. Можна застосувати програмну логіку щоб обирати, які представлення відображати чи приховати у відповідь на дії користувача чи правила доступу[37].

Крім основних компонентів в додатку ми також можемо визначати допоміжні компоненти, які керують певними ділянками розмітки html. Крім того в додатку на сторінці може бути ряд різних блоків з якимось певним завданням. І для кожного такого блоку можна створити окремий компонент, щоб спростити управління блоками на сторінці. В нашому додатку задекларовані такі компоненти: AppComponent, LoginComponent, RegisterComponent, NavigationComponent, IndexComponent, ProfileComponent, UserPostsComponent, EditUserComponent, AddPostComponent[38].

Створивши всі необхідні таблиці в базі данних, всі класи та структуру серверної частини на Java, всі класи та структуру на стороні клієнту за допомогою Angular, маємо таку архітектуру додатку:

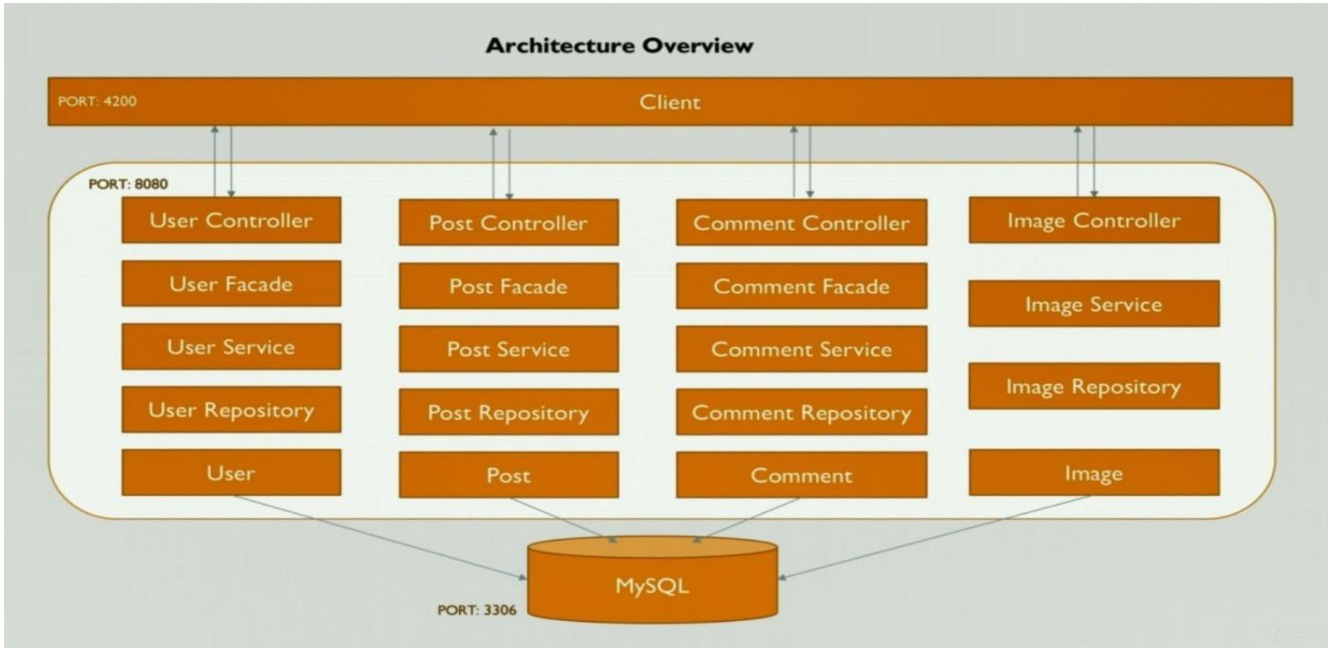


Рис. 4.1 – Архітектура додатку

Тепер безпосередньо можемо перейти до демонстрації інтерфейсу та функцій, які були реалізовані в додатку “MilitaryGram”. Для цього запустимо сервер на http://localhost:8080 і перейдемо на сторінку з такою URL http://localhost:4200/login.

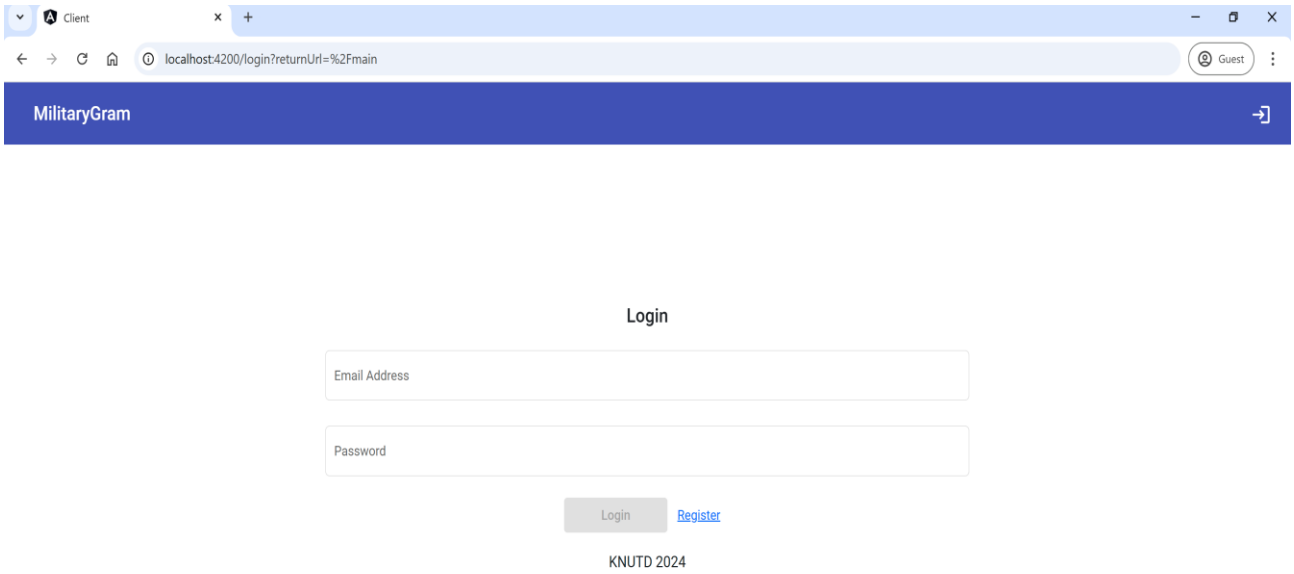


Рис. 4.2 – Сторінка логування

На стартовій сторінці ми можемо увійти до персонального кабінету, заповнивши логін(електронна адреса) та пароль. Якщо акаунт не створено, то спочатку треба перейти на сторінку реєстрації натиснувши на кнопку Register і наша адреса зміниться на таку URL http://localhost: 4200/register.

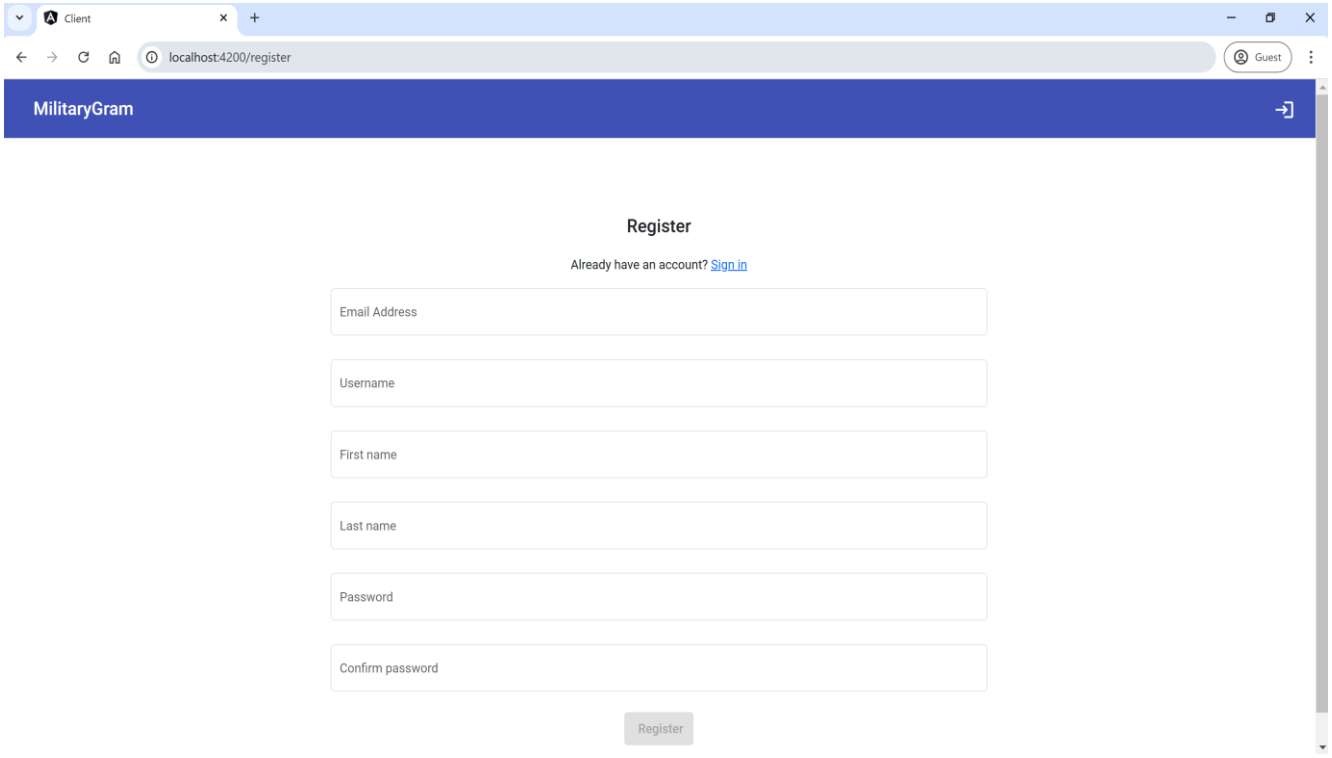


Рис. 4.3 – Сторінка реєстрації

Після того як ми увійшли під акаунтом ми потрапляємо на головну сторінку додатку за адресою URL http://localhost:8080/main, де ми бачимо всі пости користувачів, ми можемо коментувати пости, робити вподобайки під постами. А також перейти в особистий кабінет натиснувши на значок біля свого ім’я на верхній панелі з права.

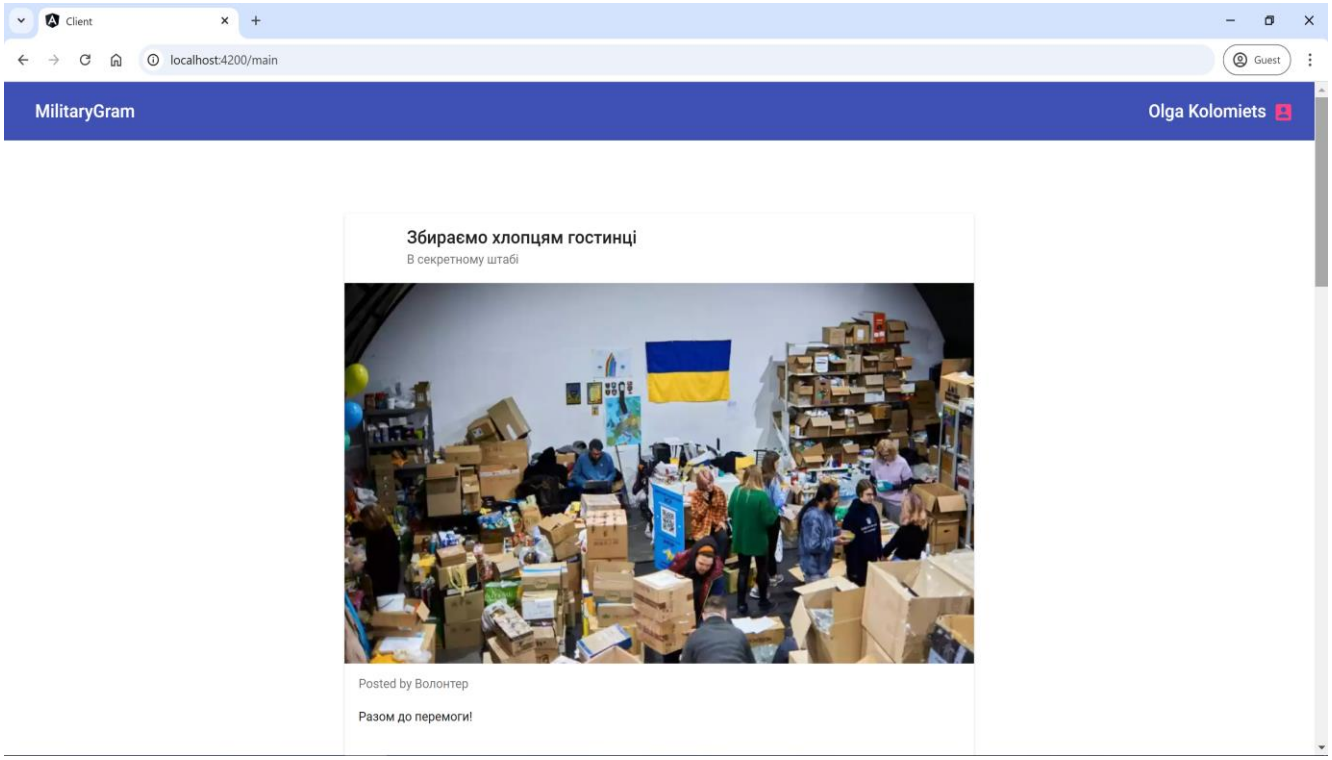


Рис. 4.4 – Стартова сторінка користувача

На стартовій сторінці користувача відразу бачимо, пости від інших користувачів від найсвіжішого від дати опублікування до найстарішого. Побачити хто запостив, коментарі до посту та вподобайки.

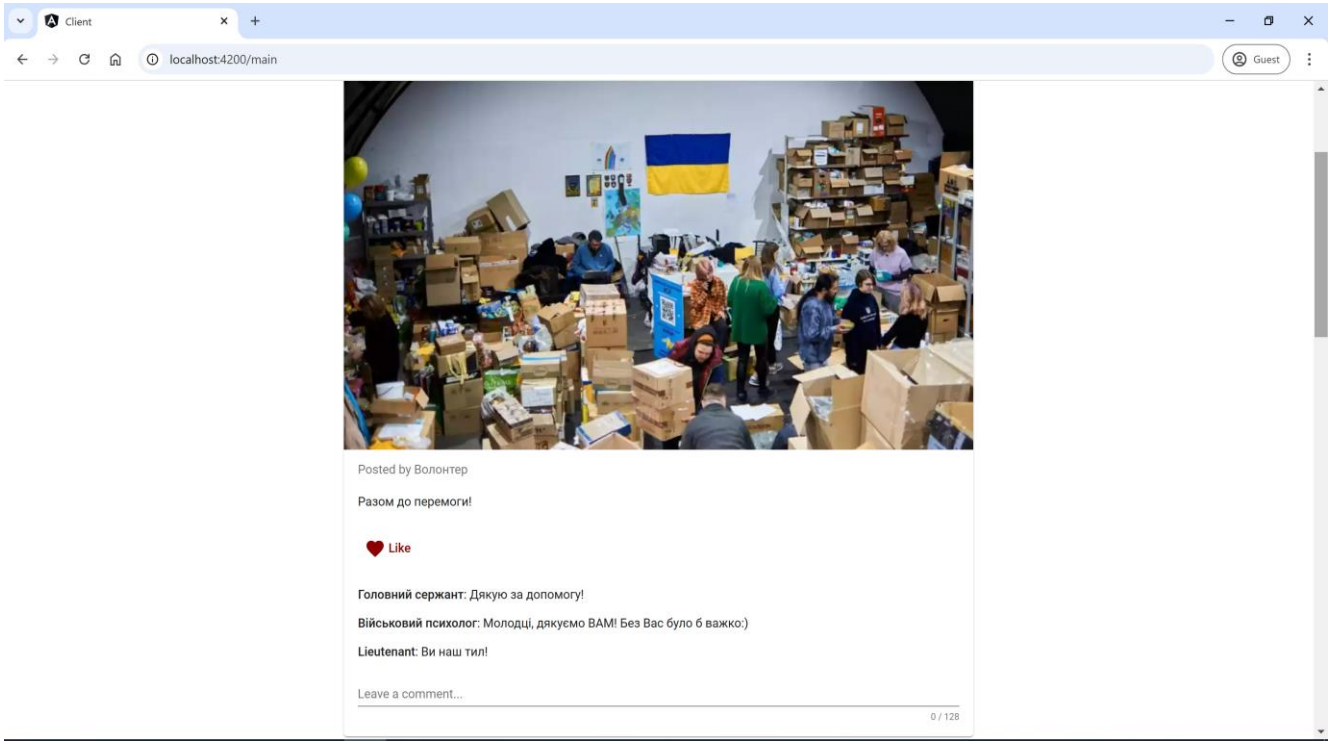


Рис. 4.5 – Пост стартової сторінки користувача

В особистому кабінеті користувача ми можемо редагувати ім’я, прізвище, біографію, фотографію профілю, додавати та видаляти пости, видаляти коментарі під своїми постами від інших користувачів, бачити хто залишив вподобайку під нашим постом та зробити logout з нашого акаунту.

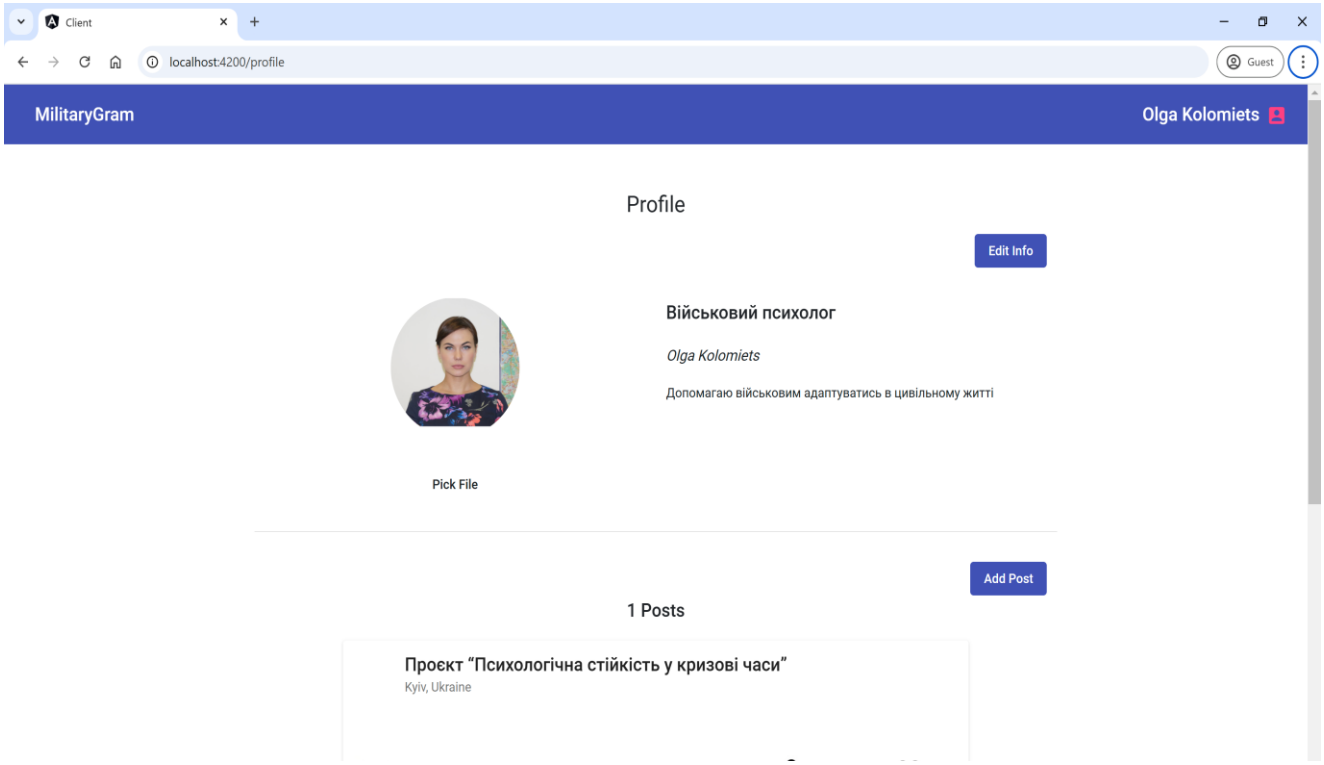


Рис. 4.6 – Особистий кабінет користувача

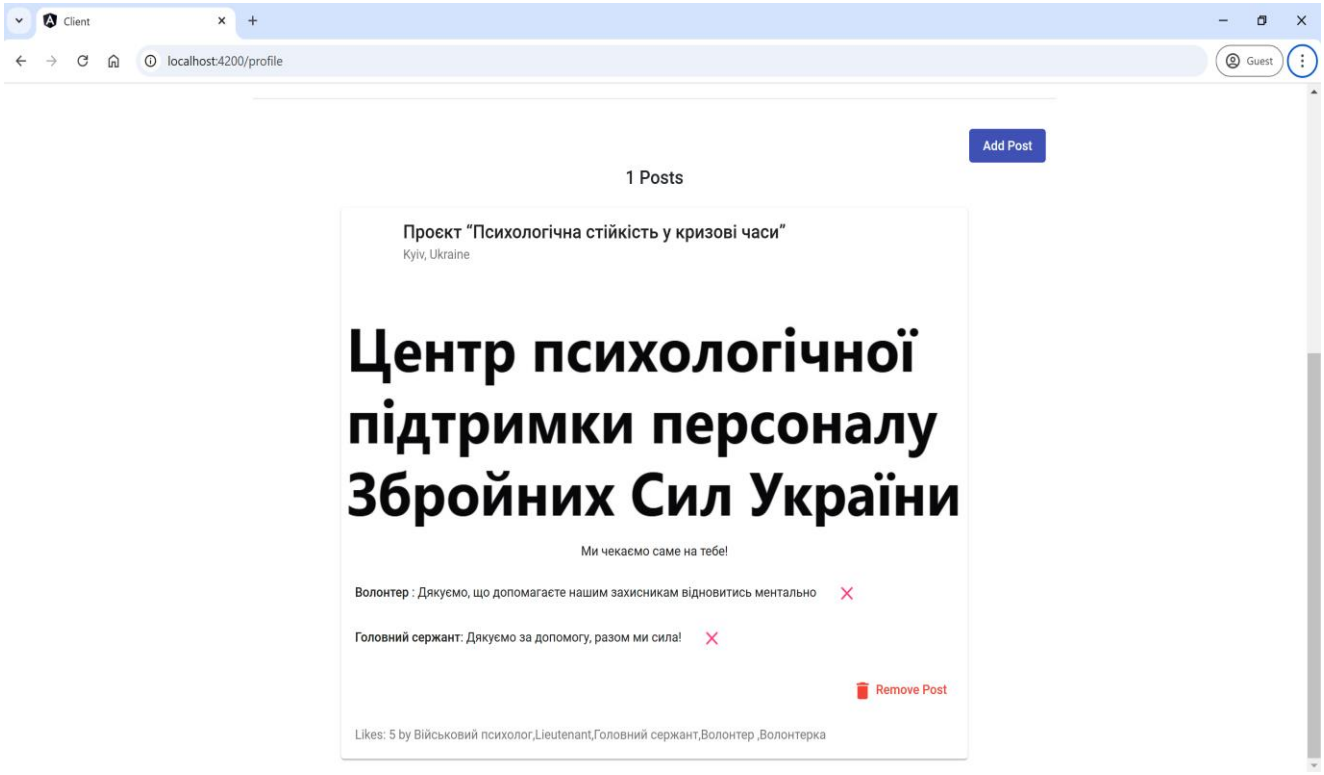


Рис. 4.7 – Пости користувача в особистому кабінеті

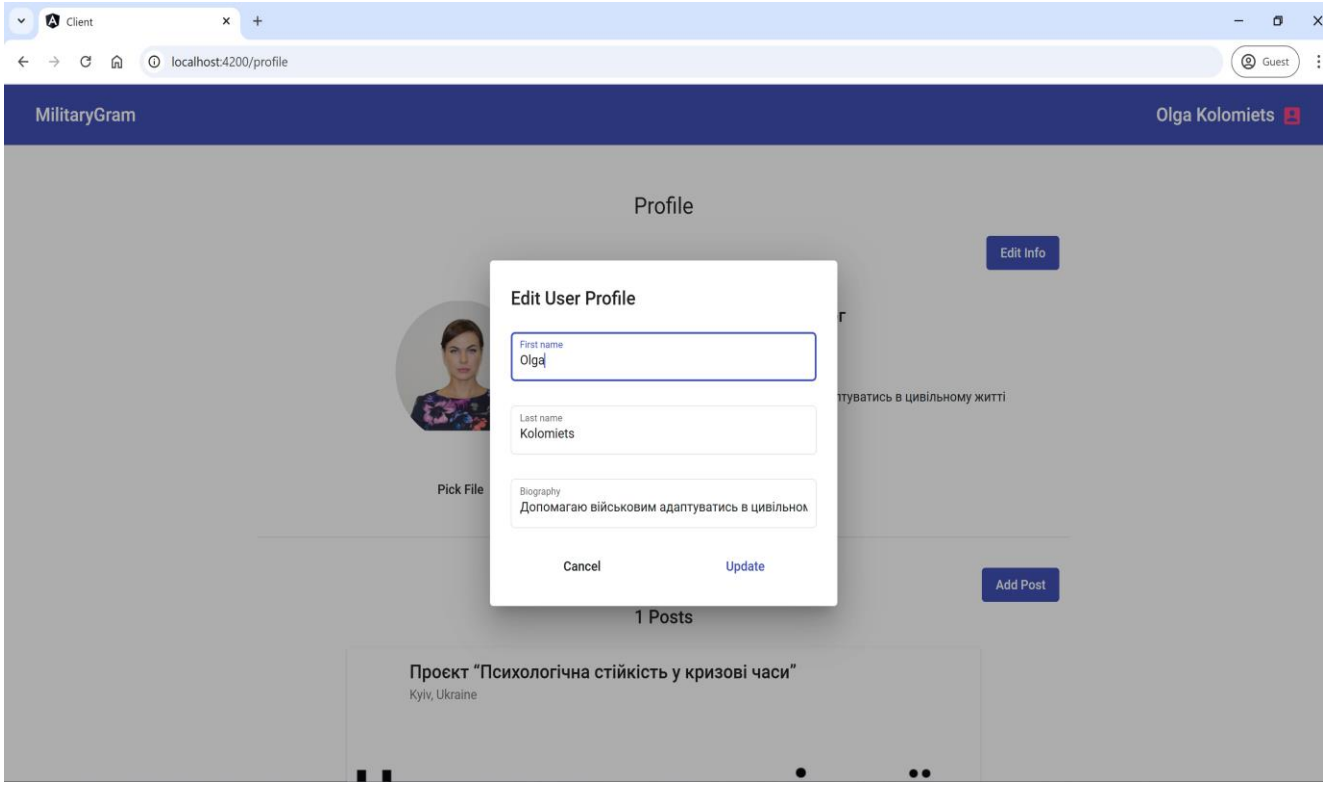


Рис. 4.8 – Редагування профайлу в особистому кабінеті

4.1. Висновки до розділу 4

У цьому розділі було розглянуто створення та реалізацію інтерфейсу користувача для програми "MilitaryGram", використовуючи технології Angular та Bootstrap. Метою було забезпечити зручний і функціональний користувацький інтерфейс, що дозволяє користувачам швидко реєструватися, входити до системи, редагувати свої персональні дані, додавати та редагувати пости, а також взаємодіяти з контентом інших користувачів.

Однією з основних особливостей було використання Angular, який дозволяє розробляти односторінкові додатки (SPA) з великим обсягом бізнес-логіки. Завдяки компонентному підходу Angular, ми змогли організувати інтерфейс у вигляді окремих модулів та компонентів, що сприяло чистоті коду, його модульності та зручності повторного використання. Використання директив і зв’язування даних дозволило ефективно взаємодіяти з DOM і оновлювати інтерфейс на основі змін у даних додатка. Роутер Angular допоміг організувати навігацію між різними сторінками додатку, забезпечивши плавний та інтуїтивно зрозумілий процес переміщення між сторінками.

У результаті інтеграції Angular з Bootstrap було забезпечено адаптивний дизайн, що дозволяє коректно відображати інтерфейс як на мобільних пристроях, так і на десктопах. Завдяки цьому, користувачі можуть зручно працювати з додатком на будь-якому пристрої.

Процес створення та редагування постів, коментування, взаємодія з іншими користувачами, а також можливість керувати власним профілем значно підвищують зручність і функціональність додатку. Завдяки організації відповідних компонентів для кожної з цих функцій (додавання постів, коментування, редагування профілю), було досягнуто високої ступені інтерактивності та зручності для користувачів.

Весь процес від реєстрації користувача до персонального кабінету і взаємодії з контентом демонструє ефективну реалізацію сучасного веб-додатку, який має чітко організовану архітектуру, зручний інтерфейс і можливість масштабування. Таким

чином, цей розділ показує, як за допомогою Angular та Bootstrap можна створити потужний і адаптивний інтерфейс, який задовольняє всі вимоги до сучасних веб-додатків.

ВИСНОВКИ

В результаті проведеної роботи над дипломним проектом було розроблено веб-додаток, який може стати надійною основою для створення повноцінної соціальної мережі, орієнтованої на військових, волонтерів, психологів та інших категорій користувачів, залучених до державних ініціатив і проектів. Розроблена система відповідає всім вимогам, зазначеним у технічному завданні, а також вимогам предметної області. Особливу увагу було приділено формуванню каталогу завдань і запитів, що дозволяє ефективно організувати робочі процеси для всіх учасників.

Одним із ключових аспектів даного проекту є те, що розроблений додаток має високу гнучкість та можливість подальшого розширення. Це забезпечується правильно побудованою архітектурою програми, що дозволяє впроваджувати нові функціональні можливості без значних змін у структурі існуючого коду. Зокрема, система проектується з урахуванням подальшого вдосконалення, що дозволяє адаптувати її до нових вимог і умов.

Особливу увагу було приділено базі даних, яка була розроблена з урахуванням принципів нормалізації, включаючи першу, другу та третю форми нормалізації. Завдяки цьому база даних є високоякісною, зручною для використання, а також легко модифікується у разі необхідності. Це забезпечує підтримку стабільної роботи системи, а також дозволяє ефективно зберігати та обробляти великий обсяг даних, що характерно для сучасних веб-застосунків.

У процесі розробки додатку було використано ряд сучасних інструментів і технологій, зокрема Angular, Bootstrap, Spring Boot, Spring Security та MySQL. Робота з цими фреймворками значно покращила мої навички в розробці веб-застосунків, оскільки кожна з цих технологій має свої специфічні можливості та

методи реалізації. Зокрема:

- Angular дозволив створити динамічний, зручний і сучасний фронтенд, що забезпечує високий рівень інтерактивності користувацького інтерфейсу.
- Bootstrap надав можливість швидко створювати адаптивний дизайн, що дозволяє додатку коректно працювати на різних пристроях.
- Spring Boot забезпечив простоту в налаштуванні серверної частини та інтеграцію з різними сервісами, що дозволяє легко масштабувати систему.
- Spring Security забезпечив високий рівень безпеки додатку, а саме аутентифікацію та авторизацію користувачів через сучасні протоколи (зокрема через JWT).
- MySQL була обрана як реляційна СУБД для зберігання даних, завдяки своїй надійності, масштабованості та підтримці транзакцій.

У процесі розробки було особливо важливо не лише реалізувати функціональні можливості, але й забезпечити зручність і простоту в користуванні. Для цього велику увагу було приділено проектуванню інтуїтивно зрозумілого інтерфейсу користувача, що відповідає вимогам зручності та доступності. Використання принципів адаптивного дизайну дозволило створити інтерфейс, який добре працює на різних типах пристроїв, включаючи мобільні телефони, планшети та десктопи.

Розробка документації стала важливою частиною дипломної роботи, що дозволило систематизувати знання про використані технології та методи, а також створити базу для майбутніх удосконалень системи. Важливим аспектом стало написання технічної та користувацької документації, що є невід'ємною частиною розробки сучасного програмного забезпечення.

Підготовка дипломної роботи допомогла глибше опанувати різноманітні аспекти розробки веб-додатків, від початкового проектування до реалізації кінцевих функціональних можливостей. Вивчення та використання різних фреймворків і технологій дозволило отримати досвід у розробці масштабованих і безпечних систем. Крім того, у ході роботи над дипломом було вдосконалено навички в роботі з базами даних, вивчено методи оптимізації запитів та ефективної

організації даних.

В цілому, реалізація даного проекту дала змогу значно покращити мої професійні навички в розробці програмного забезпечення, а також підготувала до вирішення складних задач у галузі програмної інженерії. Це є важливим кроком у моєму професійному розвитку та готовності до роботи над великими проектами в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Building Social Web Applications: Working with Java, Rails, and PHP. – Sebastopol: O'Reilly Media, 2011. – 270 с.
2. Social Media and Networking: A Guide for Beginners. – Berlin: Springer, 2020. – 312 с.
3. The Social Network Business Plan: 18 Steps to Building Your Company’s Billion-Dollar Strategy. – Irvine: Entrepreneur Press, 2011. – 198 с.
4. Legrand T., Fernandes A. Building Social Web Applications: A New Era of Web-based Social Networking. – Sebastopol: O'Reilly Media, 2013. – 340 с.
5. Makice K. Twitter API: Up and Running: Learn How to Access and Build Twitter Apps. – Sebastopol: O'Reilly Media, 2010. – 256 с.
6. Kirkpatrick D. The Facebook Effect: The Inside Story of the Company That Is Connecting the World. – New York: Simon & Schuster, 2010. – 400 с.
7. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston: Addison-Wesley, 2020. – 395 с
8. Freeman E., Robson E., Bates B., Sierra K. Head First Java. – Sebastopol: O'Reilly Media, 2021. – 720 с.
9. Freeman E., Freeman E. Head First Design Patterns. – Sebastopol: O'Reilly Media, 2004. – 694 с.
10. Bates B. Java: The Complete Reference. – New York: McGraw-Hill, 2005. – 1056 с.
11. Ковальчук О. М. Програмування на мові Java: підручник. – Київ: Видавництво КНЕУ, 2021. – 480 с.
12. Garcia-Molina H., Ullman J. D., Widom J. Database Systems: The Complete Book. – Boston: Pearson, 2008. – 1248 с.
13. Winand M. SQL Performance Explained. – Self-published, 2012. – 216 с.
14. MySQL 8.0 Reference Manual. – Redwood Shores: Oracle, 2020. – 398 с.
15. Forta B. SQL in 10 Minutes, Sams Teach Yourself. – Indianapolis: Sams Publishing, 2011. – 240 с.

16. Dyer R. J. T. Learning MySQL and MariaDB: Get Started with MySQL 5.7, MariaDB 10.2, and Percona Server. – Sebastopol: O'Reilly Media, 2017. – 408 с.
17. Spring Boot [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-boot
18. Building a RESTful Web Service [Електронний ресурс] – Режим доступу: https://spring.io/guides/gs/rest-service/
19. Spring Security [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-security
20. Walls C. Spring Boot in Action. – Shelter Island: Manning Publications, 2015. – 264 с.
21. Heckler M. Spring Boot: Up and Running. – Sebastopol: O'Reilly Media, 2021. – 328 с.
22. Carnell J. Spring Microservices in Action. – Shelter Island: Manning Publications, 2016. – 456 с.
23. Spilca L. Spring Security in Action. – Shelter Island: Manning Publications, 2021. – 576 с.
24. Knutson M. Spring Security: Authentication and Access Control. – Birmingham: Packt Publishing, 2018. – 366 с.
25. Szermer J. Spring Security 3.x Cookbook. – Birmingham: Packt Publishing, 2014. – 354 с.
26. Introduction to the Angular Docs [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs
27. Introduction to Angular Concepts [Електронний ресурс] – Режим доступу: https://angular.io/guide/architecture
28. Dayley B., Dayley B. Learning Angular. – Sebastopol: O'Reilly Media, 2016. – 352 с.
29. Freeman A. Pro Angular. – New York: Apress, 2017. – 776 с.
30. Awais M. Angular 8: Building Web Applications. – Birmingham: Packt Publishing, 2020. – 400 с.
31. Rozentals N. Mastering Angular. – Birmingham: Packt Publishing, 2020. – 412 с.

32. Moreto S. Bootstrap 4 By Example. – Birmingham: Packt Publishing, 2018. – 366 c.
33. Lett J. Bootstrap 4 Quick Start: A Beginner's Guide to Building Responsive Layouts with Bootstrap 4. – Toronto: BootstrapBay, 2017. – 140 c.
34. Shenoy A. Bootstrap 4 For Beginners: A Book for Non-Developers. – Independently published, 2020. – 232 c.
35. Seshadri S. Angular: Up and Running. – Sebastopol: O'Reilly Media, 2018. – 320 c.
36. Colburn M. S. Pro Bootstrap 4. – New York: Apress, 2017. – 288 c.
37. Lambert M. Learning Bootstrap 4. – Birmingham: Packt Publishing, 2018. – 330 c.
38. Uluca D. Angular for Enterprise-Ready Web Applications. – New York: Apress, 2020. – 760 c.

Додаток А

В програмі розглянуто 4 сутності(User, Post, Comment, Image). Код програми буде представлено на прикладі однієї сутності User.

Код сутності User:

```
@Data
@NoArgsConstructor
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String name;
    @Column(unique = true, updatable = false)
    private String username;
    @Column(nullable = false)
    private String lastname;
    @Column(unique = true)
    private String email;
    @Column(columnDefinition = "text")
    private String bio;
    @Column(length = 3000)
    private String password;
    @ElementCollection(targetClass = ERole.class)
    @CollectionTable(name = "user_role",
            joinColumns = @JoinColumn(name = "user_id"))
    private Set<ERole> roles = new HashSet<>();
    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY, mappedBy =
"user", orphanRemoval = true)
    private List<Post> posts = new ArrayList<>();
```

```
@JsonFormat(pattern = "yyyy-mm-dd HH:mm:ss")
@Column(updatable = false)
private LocalDateTime createdDate;
@Transient
private Collection<? extends GrantedAuthority> authorities;
public User(Long id,
        String username,
        String email,
        String password,
        Collection<? extends GrantedAuthority> authorities) {
   this.id = id;
   this.username = username;
   this.email = email;
   this.password = password;
   this.authorities = authorities;
}

@PrePersist
protected void onCreate() {
   this.createdDate = LocalDateTime.now();
}
@Override
public String getPassword() {
   return password;
}
@Override
public boolean isAccountNonExpired() {
   return true;
}
@Override
```

```
    public boolean isAccountNonLocked() {
        return true;
    }
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    @Override
    public boolean isEnabled() {
        return true;
    }
}

@Data
public class UserDTO {
    private Long id;
    @NotEmpty
    private String firstname;
    @NotEmpty
    private String lastname;
    private String username;
    private String bio;
}

public enum ERole {
    ROLE_USER,
    ROLE_ADMIN
}

@Component
public class UserFacade {
    public UserDTO userToUserDTO(User user) {
```

```
        UserDTO userDTO = new UserDTO();
        userDTO.setId(user.getId());
        userDTO.setFirstname(user.getName());
        userDTO.setLastname(user.getLastname());
        userDTO.setUsername(user.getUsername());
        userDTO.setBio(user.getBio());
        return userDTO;
    }
}

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findUserByUsername(String username);
    Optional<User> findUserByEmail(String email);
    Optional<User> findUserById(Long id);
}

@Service
public class UserService {
    public static final Logger LOG = LoggerFactory.getLogger(UserService.class);

    private final UserRepository userRepository;
    private final BCryptPasswordEncoder passwordEncoder;

    @Autowired
    public UserService(UserRepository userRepository, BCryptPasswordEncoder
passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }
```

```
    public User createUser(SignupRequest userIn) {
        User user = new User();
        user.setEmail(userIn.getEmail());
        user.setName(userIn.getFirstname());
        user.setLastname(userIn.getLastname());
        user.setUsername(userIn.getUsername());
        user.setPassword(passwordEncoder.encode(userIn.getPassword()));
        user.getRoles().add(ERole.ROLE_USER);

        try {
            LOG.info("Saving User {}", userIn.getEmail());
            return userRepository.save(user);
        } catch (Exception e) {
            LOG.error("Error during registration. {}", e.getMessage());
            throw new UserExistException("The user " + user.getUsername() + " already
exist. Please check credentials");
        }
    }

    public User updateUser(UserDTO userDTO, Principal principal) {
        User user = getUserByPrincipal(principal);
        user.setName(userDTO.getFirstname());
        user.setLastname(userDTO.getLastname());
        user.setBio(userDTO.getBio());

        return userRepository.save(user);
    }

    public User getCurrentUser(Principal principal) {
        return getUserByPrincipal(principal);
```

```
    }

    private User getUserByPrincipal(Principal principal) {
        String username = principal.getName();
        return userRepository.findUserByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("Username not found
with username " + username));

    }

    public User getUserById(Long id) {
        return userRepository.findById(id).orElseThrow(() -> new
UsernameNotFoundException("User not found"));
    }
}
@RestController
@RequestMapping("api/user")
@CrossOrigin
public class UserController {

    @Autowired
    private UserService userService;
    @Autowired
    private UserFacade userFacade;
    @Autowired
    private ResponseErrorValidation responseErrorValidation;

    @GetMapping("/")
    public ResponseEntity<UserDTO> getCurrentUser(Principal principal) {
```

```
        User user = userService.getCurrentUser(principal);
        UserDTO userDTO = userFacade.userToUserDTO(user);

        return new ResponseEntity<>(userDTO, HttpStatus.OK);
    }

    @GetMapping("/{userId}")
    public ResponseEntity<UserDTO> getUserProfile(@PathVariable("userId") String
userId) {
        User user = userService.getUserById(Long.parseLong(userId));
        UserDTO userDTO = userFacade.userToUserDTO(user);

        return new ResponseEntity<>(userDTO, HttpStatus.OK);
    }

    @PostMapping("/update")
    public ResponseEntity<Object> updateUser(@Valid @RequestBody UserDTO
userDTO, BindingResult bindingResult, Principal principal) {
        ResponseEntity<Object> errors =
responseErrorValidation.mapValidationService(bindingResult);
        if (!ObjectUtils.isEmpty(errors)) return errors;

        User user = userService.updateUser(userDTO, principal);

        UserDTO userUpdated = userFacade.userToUserDTO(user);
        return new ResponseEntity<>(userUpdated, HttpStatus.OK);
    }
}

@ResponseStatus(HttpStatus.BAD_REQUEST)
```

```
public class UserExistException extends RuntimeException {
    public UserExistException(String message) {
        super(message);
    }
}
```

Код обробки Response:

```
@Getter
public class InvalidLoginResponse {
    private String username;
    private String password;
    public InvalidLoginResponse() {
        this.username = "Invalid Username";
        this.password = "Invalid Password";
    }
}
```

```
Data
@AllArgsConstructor
public class JWTTokenSuccessResponse {
   private boolean success;
   private String token;
}
```

```
@Data
@AllArgsConstructor
public class MessageResponse {
   private String message;
}
```

Код обробки Request:

```
@Data
```

```
public class LoginRequest {
    @NotEmpty(message = "Username cannot be empty")
    private String username;
    @NotEmpty(message = "Password cannot be empty")
    private String password;
}
```

```
@Data
@PasswordMatches
public class SignupRequest {
    @Email(message = "It should have email format")
    @NotBlank(message = "User email is required")
    @ValidEmail
    private String email;
    @NotEmpty(message = "Please enter your name")
    private String firstname;
    @NotEmpty(message = "Please enter your lastname")
    private String lastname;
    @NotEmpty(message = "Please enter your username")
    private String username;
    @NotEmpty(message = "Password is required")
    @Size(min = 6)
    private String password;
    private String confirmPassword;
}
```

Код імплементації Security:

```
@Component
public class JWTAuthenticationEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest httpServletRequest,
```

```
HttpServletResponse httpServletResponse, AuthenticationException e) throws
IOException, ServletException {
        InvalidLoginResponse loginResponse = new InvalidLoginResponse();
        String jsonLoginResponse = new Gson().toJson(loginResponse);
        httpServletResponse.setContentType(SecurityConstants.CONTENT_TYPE);
        httpServletResponse.setStatus(HttpStatus.UNAUTHORIZED.value());
        httpServletResponse.getWriter().println(jsonLoginResponse);
    }
}

public class JWTAuthenticationFilter extends OncePerRequestFilter {
    public static final Logger LOG =
LoggerFactory.getLogger(JWTAuthenticationFilter.class);
    @Autowired
    private JWTTokenProvider jwtTokenProvider;
    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, FilterChain filterChain) throws
ServletException, IOException {
        try {
            String jwt = getJWTFromRequest(httpServletRequest);
            if (StringUtils.hasText(jwt) && jwtTokenProvider.validateToken(jwt)) {
                Long userId = jwtTokenProvider.getUserIdFromToken(jwt);
                User userDetails = customUserDetailsService.loadUserById(userId);
                UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(
                        userDetails, null, Collections.emptyList()
```

```
            );
            authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(httpServletRequest));
            SecurityContextHolder.getContext().setAuthentication(authentication);
          }
       } catch (Exception ex) {
          LOG.error("Could not set user authentication");
       }

       filterChain.doFilter(httpServletRequest, httpServletResponse);
    }

    private String getJWTFromRequest(HttpServletRequest request) {
       String bearToken = request.getHeader(SecurityConstants.HEADER_STRING);
       if (StringUtils.hasText(bearToken) &&
bearToken.startsWith(SecurityConstants.TOKEN_PREFIX)) {
          return bearToken.split(" ")[1];
       }
       return null;
    }
}

@Component
public class JWTTokenProvider {
    public static final Logger LOG =
LoggerFactory.getLogger(JWTTokenProvider.class);

    public String generateToken(Authentication authentication) {
       User user = (User) authentication.getPrincipal();
       Date now = new Date(System.currentTimeMillis());
```

```
        Date expiryDate = new Date(now.getTime() +
SecurityConstants.EXPIRATION_TIME);

        String userId = Long.toString(user.getId());

        Map<String, Object> claimsMap = new HashMap<>();
        claimsMap.put("id", userId);
        claimsMap.put("username", user.getEmail());
        claimsMap.put("firstname", user.getName());
        claimsMap.put("lastname", user.getLastname());

        return Jwts.builder()
                .setSubject(userId)
                .addClaims(claimsMap)
                .setIssuedAt(now)
                .setExpiration(expiryDate)
                .signWith(SignatureAlgorithm.HS512, SecurityConstants.SECRET)
                .compact();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser()
                    .setSigningKey(SecurityConstants.SECRET)
                    .parseClaimsJws(token);
            return true;
        }catch (SignatureException |
                MalformedJwtException |
                ExpiredJwtException |
                UnsupportedJwtException |
```

```
            IllegalArgumentException ex) {
        LOG.error(ex.getMessage());
        return false;
      }
    }

    public Long getUserIdFromToken(String token) {
      Claims claims = Jwts.parser()
            .setSigningKey(SecurityConstants.SECRET)
            .parseClaimsJws(token)
            .getBody();
      String id = (String) claims.get("id");
      return Long.parseLong(id);
    }
}

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
      securedEnabled = true,
      jsr250Enabled = true,
      prePostEnabled = true
)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private JWTAuthenticationEntryPoint jwtAuthenticationEntryPoint;
    @Autowired
    private CustomUserDetailsService customUserDetailsService;
```

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable()
.exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                .antMatchers(SecurityConstants.SIGN_UP_URLS).permitAll()
                .anyRequest().authenticated();
        http.addFilterBefore(jwtAuthenticationFilter(),
UsernamePasswordAuthenticationFilter.class);
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
auth.userDetailsService(customUserDetailsService).passwordEncoder(bCryptPasswor
dEncoder());
    }
    @Override
    @Bean(BeanIds.AUTHENTICATION_MANAGER)
    protected AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
    @Bean
    BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public JWTAuthenticationFilter jwtAuthenticationFilter() {
        return new JWTAuthenticationFilter();
```

```
    }
}

public class SecurityConstants {
    public static final String SIGN_UP_URLS = "/api/auth/*";
    public static final String SECRET = "SecretKeyGenJWT";
    public static final String TOKEN_PREFIX = "Bearer ";
    public static final String HEADER_STRING = "Authorization";
    public static final String CONTENT_TYPE = "application/json";
    public static final long EXPIRATION_TIME = 600_000; //10min
}
```

Код імплементації Validation:

```
public class EmailValidator implements ConstraintValidator<ValidEmail, String> {
    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-+]+(.[_A-Za-z0-9-
]+)*@" + "[A-Za-z0-9-]+(.[A-Za-z0-9]+)*(.[A-Za-z]{2,})$";
    @Override
    public void initialize(ValidEmail constraintAnnotation) {
    }
    @Override
    public boolean isValid(String email, ConstraintValidatorContext
constraintValidatorContext) {
        return (validateEmail(email));
    }
    private boolean validateEmail(String email) {
        Pattern pattern = Pattern.compile(EMAIL_PATTERN);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }
}
```

```
public class PasswordMatchesValidator implements
ConstraintValidator<PasswordMatches, Object> {
    @Override
    public void initialize(PasswordMatches constraintAnnotation) {

    }
    @Override
    public boolean isValid(Object obj, ConstraintValidatorContext
constraintValidatorContext) {
        SignupRequest userSignupRequest = (SignupRequest) obj;
        return
userSignupRequest.getPassword().equals(userSignupRequest.getConfirmPassword());
    }
}
```

Service

```
public class ResponseErrorValidation {
    public ResponseEntity<Object> mapValidationService(BindingResult result) {
        if (result.hasErrors()) {
            Map<String, String> errorMap = new HashMap<>();
            if (!CollectionUtils.isEmpty(result.getAllErrors())) {
                for (ObjectError error : result.getAllErrors()) {
                    errorMap.put(error.getCode(), error.getDefaultMessage());
                }
            }
            for (FieldError error : result.getFieldErrors()) {
                errorMap.put(error.getField(), error.getDefaultMessage());
            }
            return new ResponseEntity<>(errorMap, HttpStatus.BAD_REQUEST);
        }
```

```
        return null;
    }
}

@Target({ElementType.TYPE, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = PasswordMatchesValidator.class)
@Documented
public @interface PasswordMatches {
    String message() default "Password do not match";
    Class<?>[] groups() default{};
    Class<? extends Payload>[] payload() default {};
}

@Target({ElementType.TYPE, ElementType.FIELD,
ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = EmailValidator.class)
@Documented
public @interface   ValidEmail {
    String message() default "Invalid Email";
    Class<?>[] groups() default{};
    Class<? extends Payload>[] payload() default {};
}

@CrossOrigin
@RestController
@RequestMapping("/api/auth")
@PreAuthorize("permitAll()")
public class AuthController {
    @Autowired
    private JWTTokenProvider jwtTokenProvider;
```

```
    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private ResponseErrorValidation responseErrorValidation;
    @Autowired
    private UserService userService;

    @PostMapping("/signin")
    public ResponseEntity<Object> authenticateUser(@Valid @RequestBody
LoginRequest loginRequest, BindingResult bindingResult) {
        ResponseEntity<Object> errors =
responseErrorValidation.mapValidationService(bindingResult);
        if (!ObjectUtils.isEmpty(errors)) return errors;
        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                loginRequest.getUsername(),
                loginRequest.getPassword()
        ));
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = SecurityConstants.TOKEN_PREFIX +
jwtTokenProvider.generateToken(authentication);
        return ResponseEntity.ok(new JWTTokenSuccessResponse(true, jwt));
    }
    @PostMapping("/signup")
    public ResponseEntity<Object> registerUser(@Valid @RequestBody
SignupRequest signupRequest, BindingResult bindingResult) {
        ResponseEntity<Object> errors =
responseErrorValidation.mapValidationService(bindingResult);
        if (!ObjectUtils.isEmpty(errors)) return errors;
        userService.createUser(signupRequest);
```

```
        return ResponseEntity.ok(new MessageResponse("User registered
successfully!"));
    }
}
```

Додаток Б

ПРЕЗЕНТАЦІЯ ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

На тему:
"Розроблення WEB-додатку соціальна мережа "MilitaryGram " для ЗСУ"

Студент: **РемезенкоСергій Віталійович**
Керівник: **Чупринка Наталія Вікторівна**

МгІТ-23

КНУТД-2024

Актуальність дипломної роботи

полягає в розробці та впровадженні соціальної мережі, орієнтованої на потреби військовослужбовців, їхніх родин, волонтерів та фахівців, що працюють у сфері безпеки та підтримки. В умовах сучасних інформаційних викликів та війни, потреба в ефективних інструментах для комунікації, обміну інформацією та підтримки морального духу є надзвичайно важливою. Створення безпечної та зручної платформи для спілкування в таких умовах допоможе значно підвищити оперативність обміну інформацією і підтримку бойового духу серед військовослужбовців та інших зацікавлених сторін.

Мета дипломної роботи

розробка веб-додатку для ефективної комунікації та підтримки зв'язків між військовослужбовцями, їхніми родинами, волонтерами та психологами шляхом створення спеціалізованої соціальної мережі з урахуванням специфічних потреб цієї аудиторії.

Об'єкт дослідження

розробка і впровадження веб-додатку для військових, що включає функції соціальної мережі, орієнтованої на забезпечення ефективної комунікації та підтримки користувачів в умовах війни.

Предмет дослідження:

технології та методи створення соціальної мережі для військових, яка включає такі функції як створення профілів, обмін фотографіями та постами, коментування та підтримка взаємодіїкористувачів

Існуючі популярні соціальні мережі

FaceBook

Instagram

LinkedIn

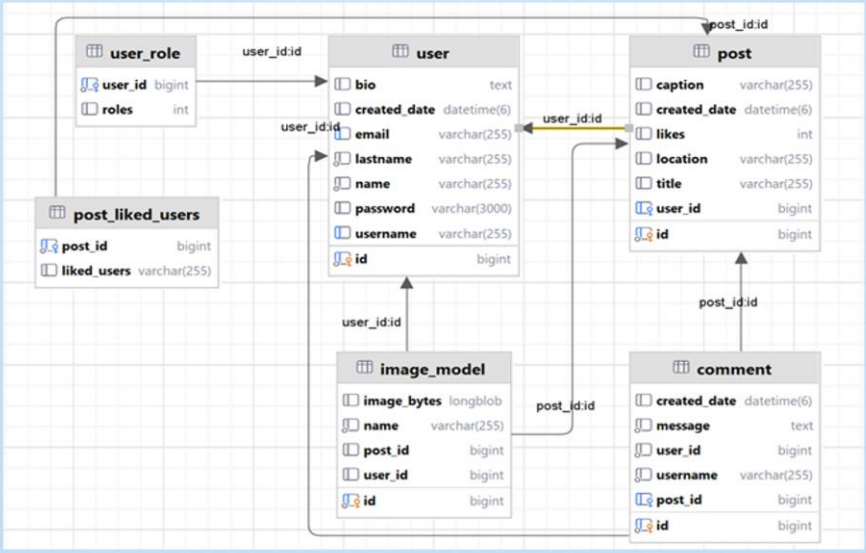
YouTube

X(Twitter)

Функції веб додатку

- реєстрація користувачів соціальної мережі;
- авторизація користувачів;
- редагування ім'я та прізвища;
- редагування спеціальності;
- додавання/видалення біографії користувача;
- додавання/редагування фотографії користувача;
- додавання/видалення постів користувача з можливістю прикріпити фото і коментар;
- додавання коментарів та вподобайок до постів інших користувачів;
- видалення коментарів інших користувачів зі свого посту;
- перегляд вподобайок.

Проектування бази даних на основі сутностей і їх функцій :



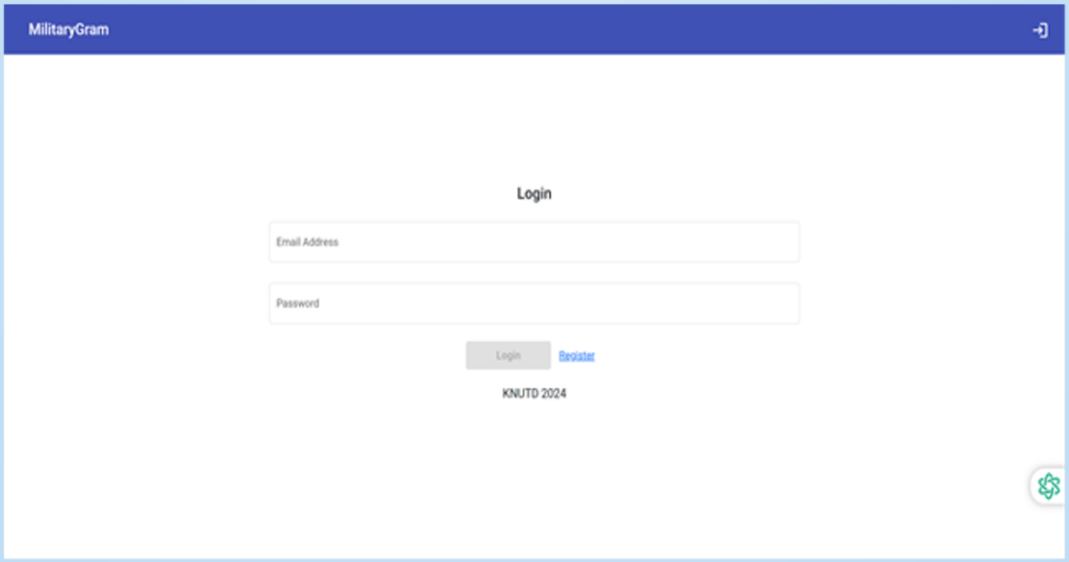
UML діаграма бази даних

Кодування сутностей на основі бази даних на мові програмування java за допомогою фреймворку spring:

```java
import com.example.demo.entity.enums.ERole;
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.*;

@Entity
@Data
@NoArgsConstructor
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String name;
    @Column(unique = true, updatable = false)
    private String username;
    @Column(nullable = false)
    private String lastname;
    @Column(unique = true)
    private String email;
    @Column(columnDefinition = "text")
    private String bio;
    @Column(length = 3000)
    private String password;
```

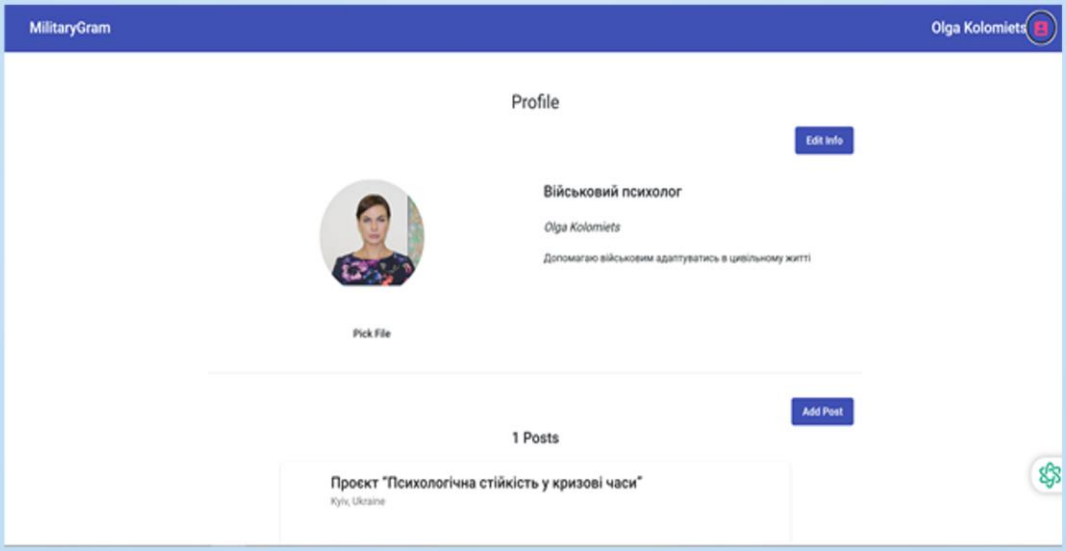
Основний інтерфейс сутності “Користувач”

Інтерфейс програми:



Стартова сторінка

Інтерфейс програми:



Сторінка профайлу

ВИСНОВКИ

В результаті проведеної роботи над дипломним проектом було розроблено веб-додаток, який може стати надійною основою для створення повноцінної соціальної мережі, орієнтованої на військових, волонтерів, психологів та інших категорій користувачів, залучених до державних ініціатив і проектів. Розроблена система відповідає всім вимогам, зазначеним у технічному завданні, а також вимогам предметної області. Особливу увагу було приділено формуванню каталогу завдань і запитів, що дозволяє ефективно організувати робочі процеси для всіх учасників.

Одним із ключових аспектів даного проекту є те, що розроблений додаток має високу гнучкість та можливість подальшого розширення. Це забезпечується правильно побудованою архітектурою програми, що дозволяє впроваджувати нові функціональні можливості без значних змін у структурі існуючого коду. Зокрема, система проектується з урахуванням подальшого вдосконалення, що дозволяє адаптувати її до нових вимог і умов.

Дякую за увагу