Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Вознюка Олега Олеговича

на здобуття ступеня вищої освіти Бакалавра

Система захисту персональних даних при обробці Big Data

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

Шифр КРБКБ.220158.22.01.01 ПЗ

Виконав: студент 3 курсу, групи КБс-22-1 Олег ВОЗНЮК

Керівник: канд. техн. наук, доцент Володимир ДЖУЛІЙ

Нормоконтролер: старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

6 06 2025 р.

Хмельницький, 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Кібербезпеки
Освітній рівень Бакалавр
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
Юрій КЛЬОЦ
15 лютого 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вознюку Олегу Олеговичу

1. Тема роботи Система захисту персональних даних при обробці Big Data ї
Керівник роботи Джулій Володимир Миколайович канд.техн.наук, доцент
Затверджено наказом ректора університету від 07 лютого 2025 № 23
2. Строк подання студентом роботи на кафедру
3. Вихідні дані до проєкту роботи Розробити веб-застосунок "Shog", для генерації великого об'єму даних. Створити скрипт для аналізу даних, отриманих в результаті роботи застосунку. Дослідити наявні способи та технології захисту Big Data. Розробити систему захисту для даних, що обробляються в системі та застосунку. Провести тестування системи та засобів захисту.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз Big Data та постановка задачі. Розробка застосунку та системи обробки інформації. Тестування застосунку.
5. Перелік графічного матеріалу «Схема БД застосунку "Shog"», «Візуальне представлення схеми роботи застосунку"Shog"»

6. Консультанти розділів курсового проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 16 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	
Ознайомлення з предметною областю	Лютий	
Дослідження існуючих рішень та проектування застосунку та системи	Лютий	
Розробка застосунку	Березень	
Розробка системи осбробки та захисту	Квітень	
Тестування застосунку та системи захисту	Травень	
Оформлення пояснювальної записки згідно вимог	Травень	
Оформлення графічної частини	Червень	
Захист ДР	Червень	

Студент Олег ВОЗНЮК

Керівник кваліфікаційної роботи Володимир ДЖУЛІЙ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система захисту персональних даних при обробці Big Data.

Автор роботи: Вознюк О.О.

Керівник роботи: Джулій В.М.

Пояснювальна записка: 72 сторінок , 47 рисунків, 41 джерело

Ця робота присвячена проєктуванню застосунку для генерації великих обсягів даних (Big Data) у структурованому, неструктурованому та напівструктурованому форматах.

В роботі проведено аналіз систем захисту Big Data та впроваджено систему безпеки обробки даних в застосунку, шляхом фільтрування, шифрування, анонімізації та узагальнення потоків даних, що будуть надходити із застосунку. Було створено алгоритми аналізу зібраних даних і проведено тестування застосунку популярними методами атак. В результаті було розроблено застосунок для збору-генерації Big Data, розроблено БД та впровадженно системи захисту під час обробки.

28.05.25 Підпис

ANNOTATION

Course project: Personal data protection system when processing Big Data.

Author of the work: Vozniuk O.O.

Supervisor: Juli V.M.

Volume – pages 72, images 47, sources 41

This work is dedicated to the design of an application for generating large volumes of data (Big Data) in structured, unstructured, and semi-structured formats.

The work analyzes Big Data protection systems and implements a data processing security system within the application, employing filtering, encryption, anonymization, and generalization of data streams from the application. Algorithms for analyzing the collected data were developed, and the application was tested using popular attack methods. As a result, an application for Big Data collection and generation was developed, a database was designed, and security systems for data processing were implemented.

28.05.25 Signature: 

ЗМІСТ

ВСТУП

У сучасному світі, де інформація стала одним із найцінніших ресурсів, обробка великих обсягів даних набуває визначального значення у різних сферах, від науки й медицини, до бізнесу та державного управління. З одного боку, Big Data надає величезні можливості для глибокого аналізу, прогнозування та ухвалення обґрунтованих рішень. З іншого боку, з'являється потреба у сфері захисту персональних даних, оскільки містить в собі значну кількість конфіденційної інформації про користувачів.

Зі стрімким розвитком технологій Big Data питання захисту персональних даних стає одним із центральних аспектів сучасного цифрового простору. Хоч і аналіз великих обсягів даних відкриває широкі можливості для прогресу в різних сферах, це супроводжується значними ризиками, пов'язаними з несанкціонованим доступом, неправомірним використанням та витоком конфіденційної інформації. У зв'язку з цим забезпечення належного рівня захисту персональних даних в умовах аналізу Big Data постає як надзвичайно важливе завдання. Вирішення цього завдання відіграє ключову роль у гарантуванні конфіденційності, безпеки користувачів, а також зменшення фінансових ризиків і втрат компанії, при несанкціонованих діях.

Тому предметом дослідження буде створення джерела Big Data, структуризація, обробка і аналіз цих даних, а також створення системи захисту для цієї інформації.

1 АНАЛІЗ BIG DATA

1.1 Основи Big Data

Big Data є поняттям, яке позначає значні за обсягом та складністю набори даних, обробка яких виходить за межі можливостей традиційних підходів. Вони характеризуються високими темпами збільшення, різноманітністю форматів і джерел походження, що зумовлює необхідність спеціалізованих підходів до їх аналізу й управління[1].

Такий тип даних має п’ять основних характеристик, що відокремлюють його від інших типів[2] (рисунок 1.1):

– об'єм;

– швидкість;

– різноманітність;

– достовірність;

– цінність.

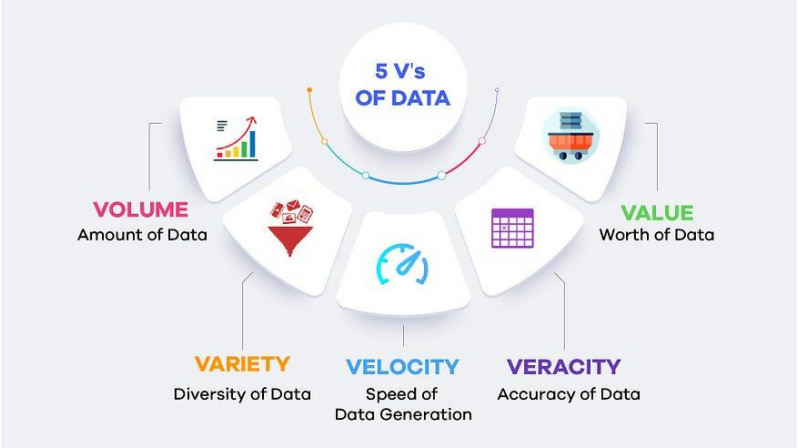


Рисунок 1.1 – Характеристики Big Data[3]

Об'єм є фундаментальною характеристикою Big Data, визначаючи їхню грандіозність. Мова йде не просто про гігабайти, а про масштаби, що вимірюються терабайтами, петабайтами, а нерідко й досягають екзабайтних чи навіть зеттабайтних величин. Цей колосальний обсяг даних не є статичним; він

демонструє невпинне, експоненційне зростання з кожним роком. Така динаміка обумовлена постійним надходженням інформації з дедалі нових і різноманітніших джерел — від сенсорів Інтернету речей (IoT), соціальних мереж та мобільних пристроїв до наукових експериментів, фінансових транзакцій та мультимедійного контенту. Паралельно зі зростанням джерел, стрімкий розвиток технологій для збору, зберігання та обробки цих величезних масивів даних є ключовим фактором. Прогрес в апаратному забезпеченні, поява надзвичайно потужних обчислювальних систем, розвиток розподілених хмарних платформ та впровадження інноваційних підходів до управління даними, таких як горизонтально масштабовані бази даних та файлові системи, роблять можливим не тільки накопичення, а й ефективну роботу з цими безпрецедентними обсягами.

Наступна критична характеристика – Швидкість. Дані в епоху Big Data не просто накопичуються, вони генеруються та надходять з надзвичайно високою, часто неперервною швидкістю. Це створює потік інформації, який вимагає майже миттєвої обробки, щоб отримати актуальні дані та інсайти. Така динамічність означає постійне оновлення інформації, що вимагає від систем Big Data здатності поглинати, обробляти та аналізувати дані практично в режимі реального часу. Традиційні методи пакетної обробки даних, які працюють з фіксованими наборами даних за розкладом, стають недостатніми. Виникає потреба в технологіях потокової обробки, які можуть аналізувати дані "на льоту", дозволяючи оперативно реагувати на події, що відбуваються, будь то зміни на фінансових ринках, поведінка користувачів на вебсайті чи показники промислового обладнання.

Різноманітність також є визначальною рисою Big Data. Цей термін охоплює дані, що існують у безлічі форматів та структур, що істотно ускладнює їх уніфіковану обробку та аналіз. На відміну від традиційних систем, які працюють переважно зі строго структурованими даними у реляційних базах даних (представлених у вигляді таблиць зі заздалегідь визначеними схемами), Big Data включає також напівструктуровані формати, такі як файли JSON або XML, які

мають певну організацію, але не мають жорсткої схеми. Крім того, значна частина Великих Даних є повністю неструктурованою: це текстові документи, електронні листи, зображення, аудіозаписи, відеофайли, дані з соціальних мереж тощо. Ця надзвичайна різноманітність вимагає розробки та застосування спеціалізованих підходів, алгоритмів та інструментів для кожного типу даних. Об'єднання, очищення та аналіз даних з настільки різних джерел та форматів створює значні технічні та методологічні виклики.

Критично важливою є Достовірність даних. Оскільки дані для Big Data надходять з величезної кількості різнорідних джерел, ймовірність того, що вони міститимуть помилки, неточності, дублікати або навіть навмисно спотворену інформацію, є значною. Низька якість даних може призвести до отримання хибних або оманливих результатів аналізу, що, своєю чергою, може стати причиною прийняття невірних рішень з потенційно серйозними наслідками. Тому забезпечення високої якості, точності та надійності даних є одним із найважливіших завдань в управлінні Big Data. Це включає процеси очищення даних, валідації, інтеграції, усунення суперечностей та впровадження політик управління якістю даних. Без належного рівня достовірності навіть найскладніші аналітичні моделі та найпотужніші обчислювальні ресурси не зможуть надати цінних та надійних інсайтів.

Нарешті, центральним аспектом є Цінність Big Data. Самі по собі величезні обсяги, швидке надходження та різноманітність даних не мають значної користі, якщо з них неможливо витягти корисні знання та інсайти. Справжня цінність Big Data полягає у потенціалі, який вони несуть для бізнесу, науки, державного управління та багатьох інших галузей діяльності. Шляхом ретельного аналізу, застосування складних статистичних методів, алгоритмів машинного навчання та штучного інтелекту, ці дані можуть бути трансформовані у значущу інформацію. Належна обробка та аналіз Big Data сприяє ухваленню більш обґрунтованих, даних-орієнтованих рішень, дозволяє виявляти приховані закономірності та нові тенденції розвитку, прогнозувати майбутні події, оптимізувати процеси,

персоналізувати послуги та ефективно вирішувати складні, раніше непідйомні завдання.

Тепер розглянемо аналітику великих даних та її сутність. Це не просто обробка інформації; це потужний трансформаційний підхід до розкриття глибинних знань, прихованих у гігантських, часто складних і хаотичних масивах цифрових даних. Аналітика великих даних являє собою багатогранний процес, який має на меті виявити цінні тенденції, стійкі закономірності, неочевидні кореляції та приховані зв’язки серед безпрецедентних обсягів сирої інформації. Кінцева мета це забезпечити міцну основу для ухвалення поінформованих, стратегічно обґрунтованих рішень, які можуть суттєво вплинути на бізнес, науку чи суспільство.

На відміну від традиційних методів статистичного аналізу, які часто обмежувалися меншими, структурованими наборами даних, аналітика великих даних масштабує ці підходи, використовуючи сучасні інструменти та технології. Вона застосовує складні статистичні та математичні методи, такі як кластерний аналіз для групування схожих об'єктів, регресійний аналіз для моделювання взаємозв'язків між змінними, а також різноманітні алгоритми машинного навчання. Однак ключова відмінність полягає в здатності застосовувати ці методи до колосальних обсягів даних, що вимагає значних обчислювальних ресурсів та спеціалізованого програмного забезпечення. Це відкриває можливість для виявлення інсайтів, які були б просто недосяжні за допомогою застарілих підходів.

Цей складний процес розгортається у кілька взаємопов'язаних етапів, які формують цілісний цикл аналізу. Усе починається з першооснови – етапу збору даних. На цьому етапі інформація акумулюється з надзвичайно широкого спектра джерел. Це можуть бути неструктуровані текстові дані з коментарів клієнтів у соціальних мережах чи відгуків на вебсайтах, структуровані дані з результатів маркетингових опитувань, потоки даних з сенсорів та пристроїв Інтернету речей, історичні транзакційні записи, дані про поведінку користувачів на вебсайтах

тощо. Чим різноманітніші та об'ємніші зібрані дані, тим більший потенціал для виявлення глибоких та цінних інсайтів.

Після того, як значний обсяг даних зібрано, настає критично важлива фаза їх очищення та попередньої обробки. Сирі дані рідко бувають ідеальними; вони часто містять пропущені значення, неточності, помилки введення, дублікати або неконсистентні формати. Цей етап є аналогією сортування сировини перед виробництвом або пошуку скарбів, де необхідно відсіяти "сміття" та "зайві" елементи, щоб залишити тільки справді цінні та достовірні дані для подальшого аналізу. Ретельне очищення є запорукою якості кінцевих результатів аналітики.

Далі відбувається етап безпосередньої обробки даних, коли очищена інформація трансформується та форматується у вигляд, який є оптимальним та придатним для застосування аналітичних інструментів та алгоритмів. Це може включати перетворення типів даних, агрегацію, нормалізацію або створення нових ознак на основі існуючих.

Кульмінацією процесу є власне аналіз даних. На цій стадії аналітики та фахівці з обробки даних застосовують арсенал статистичних моделей, математичних алгоритмів та методів машинного навчання. Мета аналізу – виявити значущі закономірності, тренди та кореляції, які не лежать на поверхні. Це може бути виявлення прихованих сегментів клієнтів та їхніх унікальних уподобань, прогнозування ринкових тенденцій, оптимізація бізнес-процесів або ідентифікація аномалій.

Щоб зробити результати цього складного аналізу зрозумілими та доступними для широкого кола зацікавлених сторін, використовується візуалізація великих даних. Комплексні дані та виявлені інсайти часто представляються у вигляді інтуїтивно зрозумілих графіків, діаграм, інформаційних панелей та інтерактивних звітів. Візуалізація дозволяє швидко охопити основні тренди, патерни та аномалії, полегшуючи процес інтерпретації результатів та прискорення ухвалення рішень.

Важливою складовою інфраструктури великих даних є зберігання та управління даними. Після аналізу дані не зникають; вони зберігаються в спеціалізованих сховищах даних, таких як озера даних або сховища даних, для подальшого використання. Належне зберігання забезпечує безпеку даних, відповідність нормативним вимогам та регуляторним актам, а також можливість повторного доступу до інформації для подальшого аналізу або перевірки попередніх результатів.

Слід розуміти, що аналітика великих даних це не одноразова дія, а динамічний, безперервний процес. Це постійне навчання та покращення. Кожен цикл аналізу може виявляти нові питання та гіпотези, які потребують подальшого дослідження. Бізнеси та організації, які успішно інтегрують аналітику великих даних у свою діяльність, отримують значну конкурентну перевагу, оскільки можуть швидше реагувати на зміни, оптимізувати ресурси та пропонувати продукти та послуги, які краще відповідають потребам клієнтів[23].

1.2 Типи даних

Існує три основні типи наборів даних, які обробляються в контексті технологій Big Data.

Першим, і найбільш традиційним, типом є структуровані дані. Ці дані мають чітко визначену, фіксовану форму, яка заздалегідь описана і не змінюється без внесення модифікацій до системи. Зазвичай вони організовані у реляційних базах даних, де інформація представлена у вигляді таблиць. Кожна таблиця має стовпції, що представляють конкретні атрибути або характеристики, такі як ім'я клієнта, дата замовлення, сума транзакції, а кожен рядок містить окремий запис, що відповідає сутності[22]. Завдяки цій суворій організації, структуровані дані надзвичайно зручні для виконання точних запитів за допомогою мови SQL та дозволяють проводити ефективний аналіз на основі чітко визначених параметрів

і зв'язків між даними. Типовими джерелами структурованих даних є системи управління взаємовідносинами з клієнтами (CRM), системи планування ресурсів підприємства (ERP), фінансові системи та операційні бази даних (рисунок 1.2).

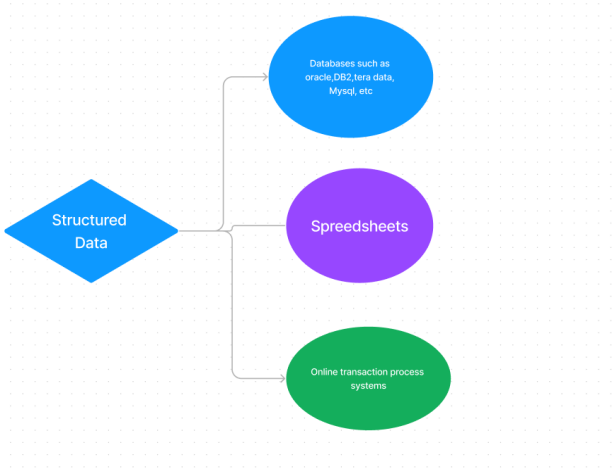


Рисунок 1.2 – Приклади структурованих даних[41]

Другим типом є напівструктуровані дані. Цей формат являє собою своєрідний компроміс, не маючи жорсткої, фіксованої схеми, як у структурованих даних, але при цьому володіючи певною організацією та ієрархією. Найпоширенішими прикладами такого формату є файли XML та JSON, які часто використовуються для обміну даними між різними системами або для зберігання конфігураційної інформації[22] (рисунок 1.3). Напівструктуровані дані містять теги, атрибути або пари ключ-значення, які надають контекст і структуру, але ця структура може бути неповною або змінюватися. Така гнучкість робить їх зручними для представлення різнорідної інформації, яка не вкладається у строгі табличні схеми, але все ж потребує певної впорядкованості для полегшення обробки. Обробка напівструктурованих даних часто вимагає спеціалізованих парсерів та інструментів, здатних інтерпретувати їхню вбудовану організацію.

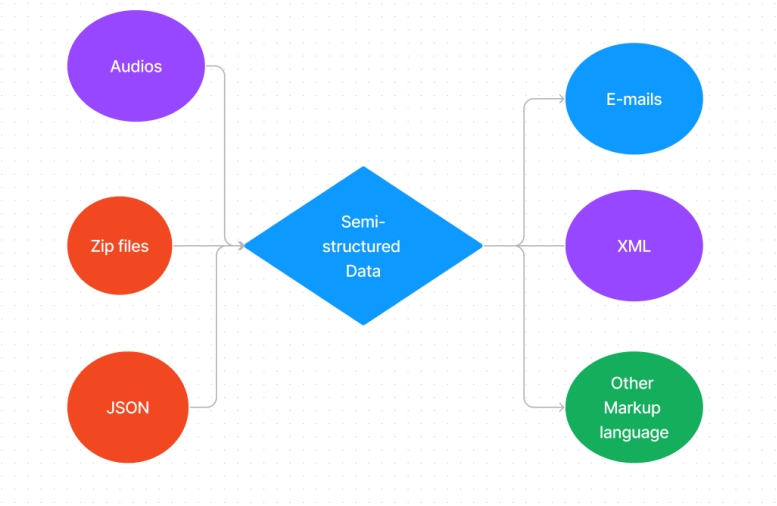


Рисунок 1.3 – Приклади напівструктурованих даних[41]

Третій, і найчисленніший в контексті Big Data, тип – це неструктуровані дані. Як випливає з назви, ці дані не мають жодної заздалегідь визначеної організації, фіксованої структури чи схеми зберігання. Сюди відноситься переважна більшість даних, що генеруються сьогодні, зокрема, тексти, документи, електронні листи, пости в соціальних мережах, зображення, аудіо- та відеофайли (рисунок 1.4). Через відсутність чіткої структури, традиційні методи роботи з базами даних, що базуються на схемах, є малоефективними або взагалі непридатними для їх прямої обробки та аналізу. Витягнення корисної інформації з неструктурованих даних вимагає застосування складних, спеціалізованих підходів та алгоритмів, таких як методи обробки природної мови (NLP) для тексту, комп'ютерний зір для зображень та відео, а також методи розпізнавання мовлення та аудіоаналізу. Прикладами неструктурованих даних є вміст електронних листів, неформальні повідомлення та коментарі в соціальних мережах, фотографії та відеозаписи, зроблені камерами спостереження чи мобільними телефонами, записи розмов кол-центрів, аудіофайли подкастів тощо[22].

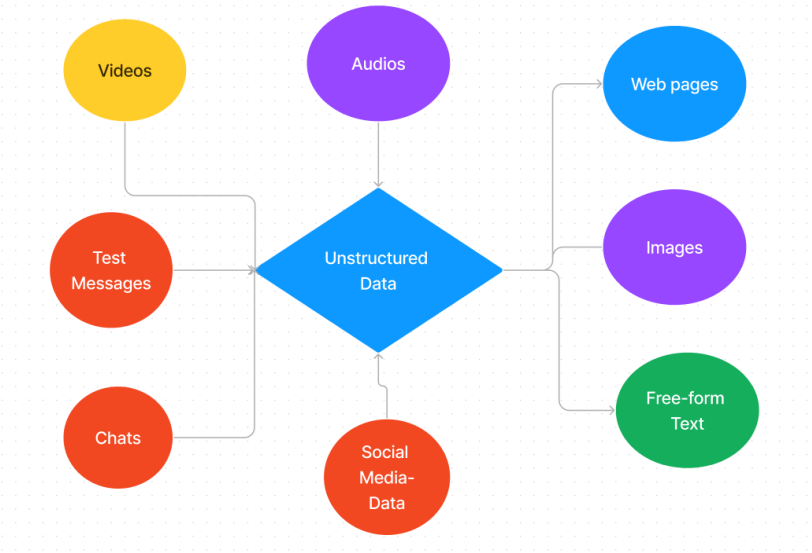


Рисунок 1.4 – Приклад неструктурованих файлів[41]

Таким чином, технології Big Data покликані ефективно працювати з усіма цими трьома типами даних, часто інтегруючи інформацію з різних джерел, що можуть представляти як строго організовані табличні дані, так і гнучкі напівструктуровані формати, і масивні обсяги абсолютно неструктурованого контенту, щоб отримати повніше та глибше розуміння предметної області.

1.3 Технології та інструменти

Зростаючі обсяги, швидкість генерації, надзвичайна різноманітність форматів, а також критична потреба у забезпеченні достовірності та вилученні цінності – ці характеристики визначають сутність Великих Даних. Для ефективної роботи з такими складними масивами інформації розроблено та активно застосовується широкий спектр технологій та інструментів, які, працюючи спільно, формують потужну екосистему для Big Data. Ці технології можна класифікувати за їхніми функціональними призначеннями, розуміючи, що часто вони тісно інтегровані між собою[24].

Однією з фундаментальних складових є розподілені файлові системи. Вони відіграють ключову роль у забезпеченні надійного та масштабованого зберігання даних, розподіляючи їх між численними вузлами (комп'ютерами) у кластері. Така архітектура дозволяє не тільки зберігати колосальні обсяги інформації, що вимірюються терабайтами та петабайтами, але й забезпечувати високу пропускну здатність для одночасного доступу та обробки даних багатьма обчислювальними процесами. Завдяки розподіленню даних і можливості паралельної обробки безпосередньо на вузлах, де зберігаються дані (принцип обчислювальної локальності), значно підвищується швидкість та ефективність роботи з великими масивами даних. Це особливо критично для завдань, де необхідна інтенсивна пакетна обробка великих обсягів інформації. Одним із найбільш відомих і широко застосовуваних прикладів таких систем є Apache Hadoop Distributed File System (HDFS), яка відома своєю стійкістю до збоїв завдяки реплікації даних, горизонтальною масштабованістю та здатністю підтримувати високопродуктивні розподілені обчислення, що є однією з ключових переваг використання екосистеми Hadoop загалом, як це може бути проілюстровано відповідними схемами [5].

Системи управління кластером виконують важливу роль у забезпеченні злагодженої та ефективної роботи всіх компонентів, які входять до складу кластера. Їх основна функція полягає у координації роботи розподіленої файлової системи, а також інших модулів і служб у межах кластерної інфраструктури. Це дозволяє досягти оптимального виконання завдань, пов’язаних із обробкою великих обсягів даних, забезпечуючи стабільність, масштабованість та продуктивність системи. Одним із прикладів таких систем є Apache Hadoop YARN, який надає можливість ефективно управляти ресурсами та розподіляти обчислювальні навантаження між вузлами[6].

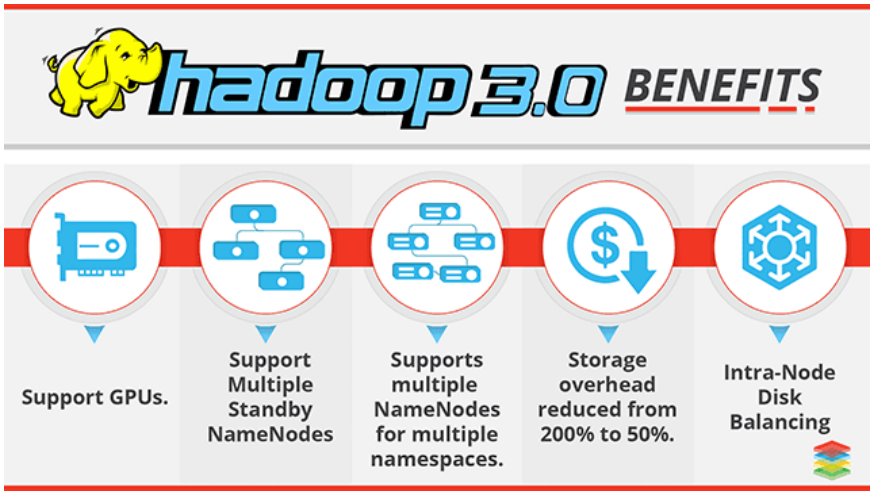


Рисунок 1.5 – Переваги використання Hadoop[6]

Для безпосереднього виконання аналітичних завдань та обробки даних використовуються обчислювальні фреймворки. Вони надають розробникам високорівневі абстракції та готові інструменти, що значно спрощують процес створення та виконання складних розподілених обчислень над великими обсягами даних. Завдяки цим фреймворкам, розробники можуть зосередитись на логіці обробки даних, не заглиблюючись у технічні деталі паралельного виконання, управління помилками та розподілу завдань. Це суттєво знижує складність роботи з Big Data та підвищує продуктивність розробки та виконання обчислювальних процесів. Серед найпопулярніших фреймворків виділяється Apache Spark, який, на відміну від більш ранніх підходів, часто виконує обчислення в оперативній пам'яті, що робить його надзвичайно швидким та масштабованим для широкого спектру завдань, включаючи інтерактивний аналіз, потокову обробку та машинне навчання, як це відображає його архітектура на відповідних схемах [7].

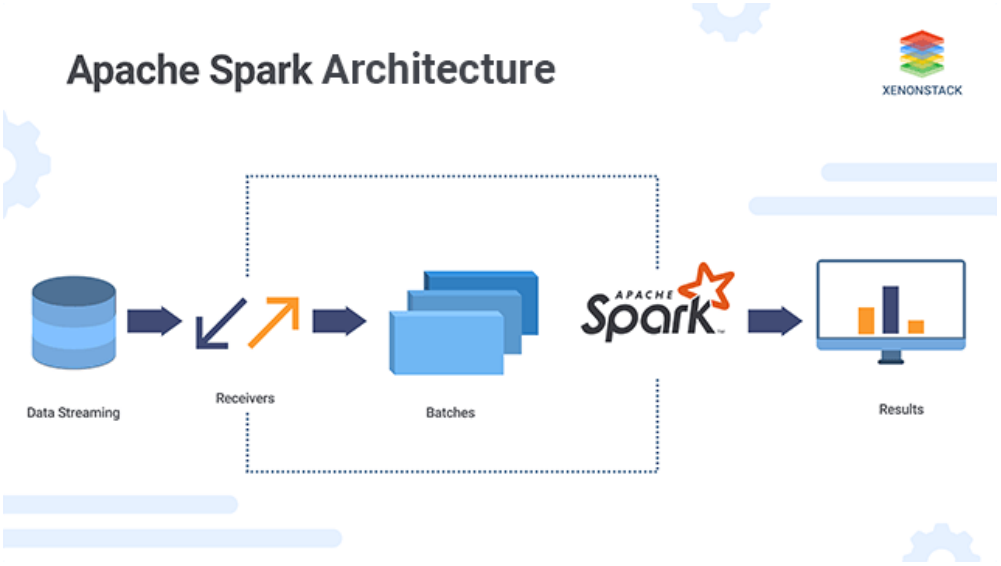


Рисунок 1.6 – Візуальне представлення архітектури Spark[7]

Окреме місце в екосистемі Big Data займають бази даних NoSQL. На відміну від традиційних реляційних баз даних, які вимагають суворої схеми та орієнтовані переважно на структуровані дані, NoSQL бази даних, призначені для ефективного зберігання та швидкого доступу до даних, які можуть бути неструктурованими або напівструктурованими. Вони розроблені з фокусом на горизонтальну масштабованість, гнучкість схеми та високу доступність, що робить їх ідеальним вибором для обробки великих обсягів різнорідних даних, таких як документи, лог-файли, дані соціальних мереж, сенсорні дані тощо. Існує кілька типів NoSQL баз даних, наприклад, документоорієнтовані, колонково-орієнтовані, Key-Value сховища, графові. Кожен з цих типів оптимізований під певні види навантажень та моделей даних.

Для написання коду, що реалізує складні алгоритми обробки та аналізу даних, використовуються спеціалізовані мови програмування. Ці мови забезпечують гнучкість та набір бібліотек, необхідних для збору, очищення, трансформації, дослідження, моделювання та візуалізації великих обсягів інформації. Вони дозволяють спеціалістам з аналізу даних реалізовувати індивідуальні підходи та застосовувати широкий спектр статистичних методів та алгоритмів машинного навчання. Серед найбільш популярних мов для роботи з

Big Data варто відзначити Python, який має багату екосистему бібліотек для аналізу даних (таких як Pandas, NumPy, Scikit-learn), R, що є стандартом для статистичних обчислень та візуалізації, а також Scala, яка часто використовується у зв'язці з Apache Spark завдяки своїй продуктивності на віртуальній машині Java (JVM).

Результати аналізу Big Data часто є складними та багатогранними, тому для їх ефективної комунікації та розуміння критично важливими є інструменти візуалізації даних. Вони дозволяють перетворити числові дані та результати аналізу у наочні графічні форми: графіки, діаграми, гістограми, теплові карти, інтерактивні дашборди. Візуалізації допомагають швидко виявити ключові тенденції, аномалії, залежності та закономірності, які можуть бути неочевидними у сирих даних або табличних звітах. Ефективне використання цих інструментів не тільки полегшує інтерпретацію складних результатів фахівцями, але й робить інсайти доступними для ширшої аудиторії, включаючи керівництво компаній та нетехнічних спеціалістів. Приклади провідних інструментів у цій галузі включають Tableau, відомий своєю інтуїтивністю та можливістю створення інтерактивних візуалізацій, та Microsoft Power BI, що пропонує глибоку інтеграцію з екосистемою Microsoft та широкі можливості для бізнес-аналітики та звітності.

Платформи машинного навчання надають користувачам набір інструментів, алгоритмів та бібліотек для створення моделей машинного навчання різної складності. Вони дозволяють автоматизувати процес аналізу великих обсягів даних, створювати прогнозні моделі, вирішувати задачі класифікації, кластеризації та багато інших питань. Окрім цього, такі платформи підтримують оптимізацію процесу навчання моделей, щоб забезпечити точніші результати. Приклади поширених платформ: TensorFlow, яка пропонує великі можливості для роботи з нейронними мережами; PyTorch, що відома своєю простотою використання та популярністю серед дослідників і розробників.

Хмарні технології в сфері Big Data відіграють ключову роль, забезпечуючи можливість доступу до високопродуктивних та масштабованих обчислювальних ресурсів, а також сучасних сервісів, призначених для зберігання, обробки та аналізу великих обсягів даних у хмарному середовищі. Завдяки використанню хмарних платформ організації отримують інструменти, що дозволяють ефективно управляти Big Data без необхідності в значних капіталовкладеннях у власну інфраструктуру. Серед найпопулярніших постачальників хмарних рішень, які підтримують роботу з великими даними, виділяються такі платформи: Amazon Web Services (AWS) зі своїми потужними сервісами для аналітики і обробки даних, Microsoft Azure, що надає інтегровані інструменти для роботи з Big Data в поєднанні з іншими корпоративними рішеннями[8], а також Google Cloud Platform (GCP), яка пропонує сучасні сервіси для високошвидкісної обробки та аналізу великих обсягів інформації. Кожен із цих провайдерів надає унікальні можливості для розвитку бізнесу за рахунок ефективного використання потужностей хмарних технологій.

1.4 Застосування Big Data

Технології Big Data є не просто інструментами для обробки великих масивів інформації; вони є рушійною силою трансформації, що охоплює надзвичайно широкий спектр галузей та сфер діяльності, істотно впливаючи на їхній розвиток, оптимізацію та загальну ефективність.

У сфері маркетингу та продажу застосування технологій Big Data радикально змінює підхід до взаємодії з клієнтами. Аналіз величезних обсягів даних про поведінку споживачів – їхні покупки, перегляди вебсайтів, взаємодію в соціальних мережах, історію запитів – дозволяє компаніям отримати глибоке розуміння потреб, вподобань та інтересів своєї цільової аудиторії на безпрецедентному рівні деталізації. Це уможливлює точну сегментацію ринку,

виявлення мікро-ніш та створення високоперсоналізованих маркетингових кампаній та пропозицій. Від точкових рекомендацій товарів на основі попередніх покупок до динамічного ціноутворення та прогнозування відтоку клієнтів – Big Data дозволяє будувати міцніші зв'язки зі споживачами та значно підвищувати конверсію та лояльність[25].

Для фінансової галузі аналітика великих даних є критично важливою для управління ризиками та забезпечення безпеки операцій. Банки, інвестиційні компанії та інші фінансові установи використовують Big Data для виявлення та мінімізації потенційних ризиків, таких як кредитні ризики, ризики ринку та операційні ризики. Аналізуючи величезну кількість структурних та неструктурних даних, включаючи дані про транзакції, ринкові показники, новини та соціальні медіа, можна будувати складні прогностичні моделі для оцінки ризиків з високою точністю. Крім того, швидка обробка та аналіз потоку транзакцій в режимі реального часу є надзвичайно ефективним механізмом для боротьби з шахрайством, дозволяючи вчасно виявляти підозрілу активність та запобігати фінансовим втратам[26].

В галузі охорони здоров'я Big Data відкривають фундаментально нові можливості. Аналіз інтегрованих масивів медичних даних, що включають історії хвороби пацієнтів, результати лабораторних досліджень, медичні зображення, генетичну інформацію, дані клінічних випробувань та наукові публікації, дозволяє виявляти приховані закономірності та взаємозв'язки. Це сприяє підвищенню точності діагностики захворювань на ранніх стадіях, персоналізації лікування на основі індивідуальних особливостей пацієнта (так звана точна медицина), розробці більш ефективних терапевтичних підходів та прискоренню процесу відкриття та розробки нових лікарських препаратів. Також Big Data допомагають у прогнозуванні розвитку хвороб та управлінні епідеміями[27].

У виробничій сфері технології Big Data значно сприяють оптимізації та підвищенню ефективності виробничих процесів. Збір та аналіз даних із сенсорів обладнання, систем управління виробництвом, ланцюгів постачань та систем

контролю якості дозволяє отримати повну картину роботи підприємства. Це дає можливість не тільки ефективніше планувати та управляти ресурсами, оптимізувати витрати енергії та сировини, але й реалізувати концепцію "передбачувального обслуговування" (predictive maintenance). Аналізуючи дані про стан обладнання, можна прогнозувати ймовірні поломки ще до того, як вони стануться, дозволяючи провести технічне обслуговування завчасно. Це дозволяє уникнути несподіваних простоїв, мінімізувати втрати від поломок та суттєво знизити витрати на ремонти та простої[28].

В галузі транспорту Big Data стають незамінним інструментом для планування та управління складними транспортними системами. Аналіз величезних обсягів даних про дорожній трафік, використання громадського транспорту, погодні умови, інформацію з GPS-трекерів та мобільних пристроїв, часто в режимі реального часу, дозволяє оптимізовувати маршрути перевезень, керувати світлофорами для зменшення заторів, прогнозувати час прибуття транспорту та підвищувати загальну ефективність логістичних ланцюгів. Це сприяє не тільки скороченню часу в дорозі та витрат на паливо, але й зменшенню викидів та покращенню екологічної ситуації в містах[29].

Особливо значущим є взаємозв'язок між Інтернетом речей (IoT) та технологіями Big Data. Мільярди підключених пристроїв IoT, від промислових сенсорів та "розумних" побутових приладів до носимих гаджетів, які генерують безпрецедентні обсяги даних з надзвичайно високою швидкістю. Технології Big[30] Data є ключовими для збору, обробки, зберігання та аналізу цього потоку інформації. Поєднання даних з IoT з іншими джерелами дозволяє створювати інтелектуальні системи, що забезпечують автоматизацію процесів, оптимізацію споживання ресурсів, покращення безпеки та підвищення загальної продуктивності у найрізноманітніших секторах, від "розумних міст" та промисловості 4.0, до точного землеробства та персоналізованого моніторингу здоров'я.

1.5 Захист Big Data

Зважаючи на специфіку Big Data, що характеризуються величезними обсягами, високою швидкістю надходження та значною різноманітністю, забезпечення їхнього захисту під час обробки стає надзвичайно важливим та складним завданням. Необхідно застосовувати багатогранний підхід, що охоплює всі етапи життєвого циклу даних, від моменту їх збору до зберігання та аналізу, для гарантування конфіденційності, цілісності та доступності інформації.

Фундаментальним аспектом є правове забезпечення та відповідність нормам. Будь-яка обробка даних, особливо якщо вона стосується персональної інформації, має неухильно відповідати вимогам чинного національного законодавства та міжнародних стандартів, таких як європейський Загальний регламент про захист даних (GDPR)[12]. За останні три десятиліття понад два десятки держав ухвалили закони про захист персональних даних, що встановлюють дієві правові механізми для регулювання обігу цієї інформації[11]. Це вимагає ретельного аналізу внутрішніх політик та процедур обробки даних на їх відповідність правовим нормам, для чого може знадобитися залучення кваліфікованих юристів. Крім того, надзвичайно важливо обирати та використовувати лише ті інструменти, платформи та технології для обробки даних, які самі відповідають встановленим стандартам захисту та безпеки.

Невід'ємною частиною захисту є контроль доступу та управління ідентифікацією. Необхідно впроваджувати та підтримувати надійні механізми для автентифікації користувачів, які намагаються отримати доступ до даних, та авторизації, що визначає, які саме дії вони можуть виконувати. Критично важливим є чітке розмежування прав доступу на основі ролей та функціональних обов'язків співробітників, реалізуючи принцип найменших привілеїв, щоб кожен користувач мав доступ лише до тієї інформації, яка необхідна для виконання його роботи[13].

Для захисту даних від несанкціонованого перегляду як під час їх зберігання, так і під час передачі мережею, слід використовувати шифрування даних. Сучасні та криптографічно стійкі алгоритми шифрування мають застосовуватися для захисту даних "at rest", тобто коли вони зберігаються на дисках чи в базах даних, а також "in transit", під час їх передачі між системами чи користувачами. Ефективне управління ключами шифрування також є обов'язковою складовою цього процесу[31].

Безпека інфраструктури, в якій обробляється Big Data , вимагає комплексного підходу. Це включає захист мережевих периметрів за допомогою брандмауерів, використання систем виявлення та запобігання вторгненням (IDS/IPS) для моніторингу та блокування шкідливого трафіку, а також захист серверів та систем зберігання даних від різноманітних зовнішніх та внутрішніх загроз[15]. При використанні хмарних платформ необхідно приділяти особливу увагу питанням безпеки, ретельно аналізуючи механізми захисту, що пропонуються провайдером, та розуміючи модель розподілу відповідальності за безпеку. Варто зазначити, що індивідуально розроблені системи можуть потенційно надати організації більший контроль над захистом критично важливої інформації порівняно зі стандартними комерційними рішеннями`.

Проактивний захист забезпечується через моніторинг та аудит. Необхідно здійснювати постійний моніторинг систем обробки Big Data для своєчасного виявлення будь-якої підозрілої активності, аномалій у поведінці систем чи користувачів, що можуть свідчити про потенційні загрози безпеці. Паралельно слід вести детальні та захищені від модифікації журнали подій (аудит), які фіксують усі важливі операції з даними. Ці журнали є незамінними для розслідування інцидентів безпеки, якщо такі стануться[15].

Для зменшення ризиків, пов'язаних із обробкою персональних даних, доцільно застосовувати техніки анонімізації та псевдонімізації. Ці методи дозволяють видалити або замінити прямі ідентифікатори особи (анонімізація) або замінити їх на унікальні псевдоніми (псевдонімізація), таким чином

знеособлюючи дані перед їх подальшим аналізом. Це особливо корисно у випадках, коли для аналітичних цілей не потрібна пряма ідентифікація суб'єктів даних[32].

Ефективне управління ризиками є безперервним процесом, що включає регулярну ідентифікацію, аналіз та оцінку потенційних загроз і вразливостей, пов'язаних з обробкою Big Data. На основі цієї оцінки розробляються та впроваджуються відповідні заходи для мінімізації ризиків. Це вимагає глибокого розуміння потоків даних в організації: які дані збираються, з якою метою, де вони зберігаються, хто має до них доступ, і який рівень захисту забезпечується на кожному етапі[15]. Важливо також враховувати людський фактор, забезпечуючи захист від внутрішніх загроз шляхом ретельного відбору та перевірки персоналу та контрагентів, що мають доступ до чутливих даних.

Таким чином, захист Big Data при обробці вимагає комплексного, багатошарового підходу, що інтегрує правові, технічні та організаційні заходи для забезпечення надійного захисту цінної інформації на всіх етапах її життєвого циклу.

1.6 Постановка задачі

Для генерації великих масивів даних (Big Data) та їх подальшої аналітичної обробки необхідно створити комплексний програмний продукт, який би в процесі своєї роботи природним чином продукував значні та різноманітні інформаційні потоки. Оптимальним вибором для цієї мети стала розробка соціальної мережі, оскільки подібні платформи є потужними генераторами даних завдяки безперервній активності користувачів: створення контенту, взаємодії, комунікації та метадані, що супроводжують ці процеси.

2 РОЗРОБКА ЗАСТОСУНКУ ТА СИСТЕМИ ОБРОБКИ

2.1 Вибір стеку та пошук готових рішень

Реалізація задуманої соціальної мережі буде здійснена у вигляді сучасного веб-застосунку. Для розробки цього амбітного проєкту було обрано потужний та гнучкий технологічний стек, який об'єднує передові JavaScript-технології для створення повноцінних, масштабованих та високоінтерактивних додатків. В основі цієї архітектури лежать три ключові компоненти: MySQL для керування даними, React для побудови клієнтського інтерфейсу та Node.js для реалізації серверної логіки. Такий вибір створює чітку трирівневу архітектуру (клієнт-сервер-база даних), де кожен елемент виконує свою спеціалізовану функцію, забезпечуючи надійність та ефективність усієї системи.

Клієнтська частина (Frontend), з якою безпосередньо взаємодіють користувачі, буде розроблена з використанням бібліотеки React. Це дозволить створити динамічний, швидкий та чутливий користувацький інтерфейс, що є критично важливим для соціальних мереж, де контент постійно оновлюється. Завдяки компонентній архітектурі React, інтерфейс розбивається на незалежні, перевикористовувані блоки (наприклад, компонент профілю, стрічки новин, коментаря), що значно спрощує розробку, тестування та подальшу підтримку. Використання віртуального DOM забезпечує оптимальну продуктивність, оновлюючи лише ті частини сторінки, які зазнали змін, що створює плавний та безперервний досвід для користувача.

Серверна частина (Backend), яка є ядром застосунку, буде побудована на платформі Node.js, ймовірно, з використанням фреймворку Express.js для структурування коду та обробки запитів. Node.js ідеально підходить для соціальних мереж завдяки своїй асинхронній, керованій подіями архітектурі. Це дозволяє серверу ефективно обробляти тисячі одночасних з'єднань — наприклад, коли безліч користувачів одночасно переглядають стрічку, ставлять лайки, коментують та надсилають повідомлення — без блокування та затримок. Бекенд

буде відповідати за всю бізнес-логіку: автентифікацію користувачів, обробку запитів від клієнта, взаємодію з базою даних та формування API (Application Programming Interface), через який фронтенд отримуватиме та надсилатиме дані.

Для збереження, керування даними, та надійне зберігання всієї інформації, буде реалізовано за допомогою системи керування реляційними базами даних MySQL. Це перевірене часом, стабільне та потужне рішення, яке чудово підходить для зберігання структурованих даних. Інформація про користувачів, їхні зв'язки, пости, коментарі та інші сутності будуть організовані в чітко визначених таблицях зі зв'язками між ними. Це забезпечує цілісність даних та транзакційну надійність (ACID), що є вкрай важливим для збереження акаунтів та контенту користувачів.

Обраний стек технологій, хоч і складається з різних елементів, забезпечує значну однорідність середовища розробки, оскільки основна мова програмування як для фронтенду, так і для бекенду це JavaScript. Це дозволяє розробникам легше переключатися між завданнями та сприяє кращій співпраці в команді, що в кінцевому підсумку знижує час і загальну складність створення та подальшої підтримки проєкту. Крім того, кожен з цих компонентів має величезну та активну спільноту розробників, що гарантує доступ до великої кількості документації, готових рішень, бібліотек та форумів для вирішення будь-яких проблем.

2.2 Розробка застосунку

Зазвичай, повний цикл розробки веб-застосунків охоплює етапи проектування, дизайну, розробки та розгортання. Однак, з метою оптимізації часових ресурсів та прискорення розробки застосунку, було прийнято рішення відійти від традиційного послідовного процесу і використати готовий шаблон дизайну та фронтенд-компонентів. Це дозволило значно скоротити фазу

проектування клінієнту для користувачів та переключитися безпосередньо до імплементації функціоналу.

Обраний шаблон виявився комплексним рішенням, що вже включав в себе значну частину реалізованих фронтенд-компонентів та базових сторінок. Як ілюструють рисунок 2.1 та рисунок 2.2, це надало міцну основу, що охоплює основні елементи інтерфейсу користувача та попередньо визначені макети сторінок. Такий підхід не тільки економить час, що зазвичай витрачається на розробку UI/UX з нуля, але й забезпечує певну узгодженість у візуальному стилі та поведінці компонентів.

Для забезпечення повноцінного функціонування клієнтської частини цього веб-застосунку, першим кроком є ініціалізація проекту. Цей процес здійснюється за допомогою команди yarn. Виконання цієї команди запускає автоматичне завантаження та інсталяцію всіх необхідних залежностей та пакетів, які визначені у файлі конфігурації проекту, зазвичай це package.json. Це гарантує, що всі бібліотеки, фреймворки та інструменти, необхідні для компіляції та виконання фронтенду, будуть доступні в локальному середовищі розробки.

Після успішної ініціалізації проекту та встановлення всіх залежностей, для запуску самого клієнта використовується команда yarn start. Ця команда активує локальний сервер розробки, який завантажує та відображає веб-застосунок у браузері.

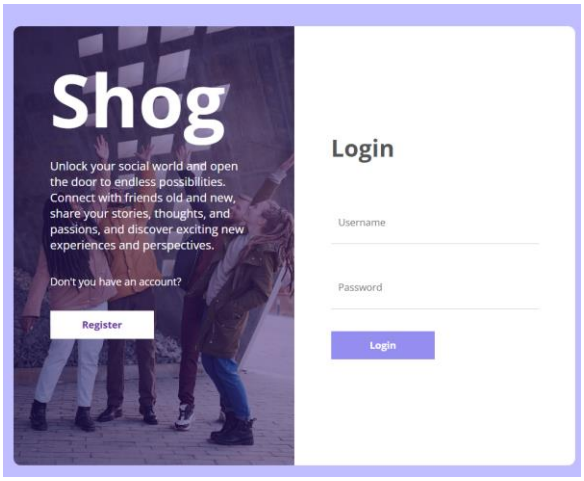


Рисунок 2.1 – Форма логіну

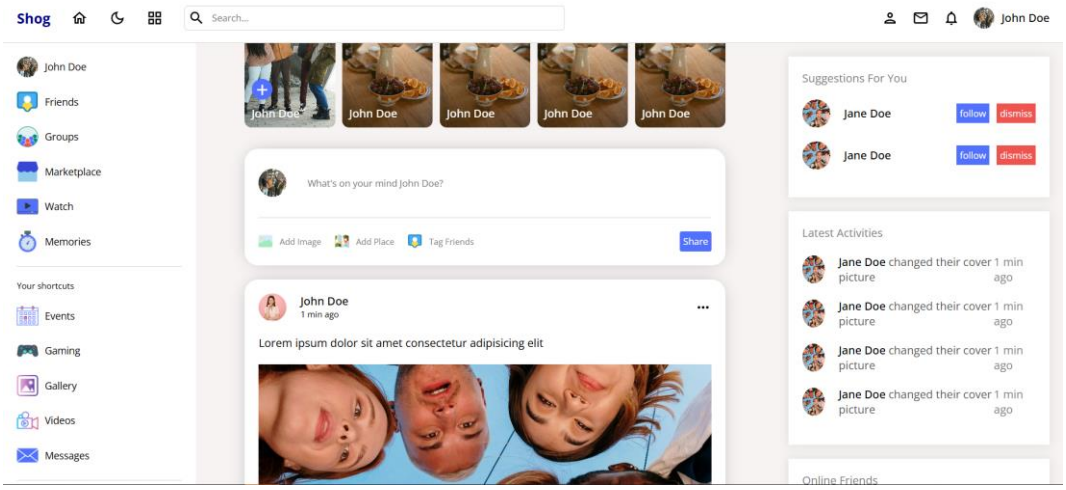


Рисунок 2.2 – Головна сторінка

Форми логіну та реєстрації, які входять до складу готового шаблону, наразі не містять вбудованих функцій для перевірки валідності введених даних. Це становить потенційний ризик безпеці та призводить до надлишкових запитів на сервер з некоректною інформацією. Для вирішення цих проблем критично важливо імплементувати механізми клієнтської валідації. Це дозволить відсіювати невірні дані на етапі введення користувачем, підвищить безпеку застосунку та значно зменшить навантаження на сервер.

Оскільки клієнтська частина розроблена на React, найбільш ефективним підходом для ініціації перевірки є використання обробника події onChange. Цей обробник буде спрацьовувати щоразу, коли користувач вносить зміни до поля введення, і викликати функцію checkField.

Клієнтська частина розроблена на React, тому найбільш найбільш ефективним підходом для ініціації перевірки є використання обробника події onChange, який буде спрацьовувати щоразу, коли користувач вносить зміни до поля введення, і викликати функцію checkField.

Функція checkField буде відповідати за перевірку коректності даних у відповідному полі. Її логіка буде базуватися на атрибуті name поля введення, що дозволить проводити специфічні перевірки для різних полів, наприклад перевірка формату email, довжини пароля, тощо.

У разі, якщо перевірка виявить некоректні дані, поле введення буде візуально підсвічуватись червоним кольором, а під формою реєстрації з'явиться блок помилки, що чітко повідомлятиме причину проблеми[33].. Це надасть користувачеві миттєвий зворотний зв'язок і допоможе йому виправити помилки перед відправкою форми.

Першим етапом у ланцюжку валідаційних перевірок буде визначення, чи є у полі введення взагалі якийсь вміст. Це базова, але важлива перевірка, яка дозволяє відсіяти порожні поля. Для цього буде використано функцію, яка аналізує наявність даних (рисунок 2.3, рисунок 2.4).

```
const isInputEmpty = (target) => {
  const check = target.value.trim() !== ""
  check ? setErr("") : setErr(target.name + " is empty")
  setErrClassName(target, check)
};
```

Рисунок 2.3

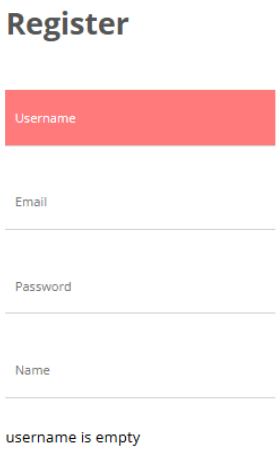


Рисунок 2.4 – Результат виконання isInputEmpty

Після базової перевірки на наявність вмісту у полях введення, наступним критично важливим етапом валідації є перевірка формату електронної пошти. Це дозволяє гарантувати, що введена адреса відповідає загальноприйнятим

стандартам email, що є важливим для коректної доставки повідомлень та функціонування системи. Для реалізації цієї перевірки буде використовуватися спеціалізована функція, представлені на рисунку 2.5.

```
const isValidEmailFormat = (target) => {
  const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
  const check = emailRegex.test(target.value);
  check ? setErr("") : setErr(target.name + " wrong format")
  setErrClassName(target, check)
};
```

Рисунок 2.5 – Код функції isValidEmailFormat

Функція isValidEmailFormat приймає об'єкт target і починає свою роботу з визначення регулярного виразу emailRegex. Цей вираз є потужним інструментом для перевірки синтаксису електронної пошти, забезпечуючи відповідність адреси типовим правилам форматування, а саме наявність імені користувача, символу @, доменного імені та домену верхнього рівня. Далі, за допомогою методу test() регулярного виразу, функція перевіряє значення поля введення target.value на відповідність цьому шаблону, зберігаючи результат у змінній check.

На основі значення check, функція умовно оновлює стан повідомлення про помилку: якщо формат коректний check є true, то повідомлення очищається за допомогою setErr(""), якщо формат некоректний check є false, то встановлюється відповідне повідомлення про помилку, що включає ім'я поля та вказівку на "wrong format". Завершальним кроком є виклик функції setErrClassName(target, check), яка відповідає за візуальне відображення статусу валідації, змінюючи стиль поля, підсвічуючи його червоним у разі виявлення помилки.

Таким чином, ця функція забезпечує миттєвий зворотний зв'язок користувачеві, підвищує якість введених даних і зменшує навантаження на серверну частину застосунку (рисунок 2.6).

Рисунок 2.6 – Результат виконання isValidEmailFormat

Після перевірки наявності вмісту та формату електронної пошти, третім, і не менш важливим етапом клієнтської валідації є перевірка складності пароля. Це критично важливо для забезпечення безпеки облікових записів користувачів, оскільки слабкі паролі є однією з найпоширеніших вразливостей. Для реалізації цих вимог буде використана функція isValidPassword, імплементація якої детально представлена на рисунку 2.7.

```
const isValidPassword = (target) => {
  const errors = [];
  if (!/[A-Z]/.test(target.value)) {
    errors.push("Must contain at least one uppercase letter.\n");
  }
  if (!/[0-9]/.test(target.value)) {
    errors.push("Must contain at least one digit.\n");
  }
  if (!/[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?]/.test(target.value)) {
    errors.push("Must contain at least one special character (!@#$%^&*).\n");
  }
  const check = errors.length === 0
  check ? setErr("") : setErr(errors)
  setErrClassName(target, check)
};
```

Рисунок 2.7 – Код функції isValidPassword

Функція isValidPassword приймає об'єкт target, який представляє собою поле введення пароля і починає свою роботу зі створення порожнього масиву errors, який буде зберігати всі виявлені проблеми з паролем. Далі, функція

послідовно виконує три окремі перевірки за допомогою регулярних виразів. Перша перевірка if (!/[A-Z]/.test(target.value)) визначає, чи містить пароль хоча б одну велику літеру, якщо ні, до масиву errors додається відповідне повідомлення "Must contain at least one uppercase letter.". Друга перевірка if (!/[0-9]/.test(target.value)) аналогічно шукає наявність щонайменше однієї цифри, додаючи "Must contain at least one digit.", у разі її відсутності. Третя перевірка if (!/[!@#$%^&*()_+-=\[\]{};':"\\|,.<>\/?]/.test(target.value)) є найскладнішою і перевіряє наявність хоча б одного спеціального символу з визначеного набору, якщо такий символ відсутній, додається повідомлення "Must contain at least one special character (!@#$%^&*).". Після виконання всіх трьох перевірок, функція обчислює змінну check, яка дорівнює true, якщо масив errors порожній, тобто пароль пройшов усі перевірки і false в іншому випадку. На основі значення check функція оновлює стан повідомлення про помилку, якщо check є true, повідомлення очищається setErr(""). Якщо check є false, у setErr передається весь масив errors, що дозволяє відобразити користувачеві всі виявлені невідповідності пароля.

Насамкінець, викликається функція setErrClassName(target, check), яка візуально відображає результат валідації на елементі форми (рисунок 2.8). Таким чином, ця комплексна функція забезпечує надійну перевірку складності пароля на клієнтській стороні, сприяючи підвищенню безпеки облікових записів.

Рисунок 2.8 – Результат виконання isValidPassword

Після успішної реалізації комплексних перевірок для форми реєстрації, які включають валідацію наявності вмісту, формату електронної пошти та складності пароля, логічним і необхідним кроком є застосування аналогічних механізмів валідації до форми логіну. Важливо зазначити, що форма логіну, на відміну від форми реєстрації, містить лише поля для імені користувача та пароля, тобто перевірка складності електронної пошти та паролю, яка була реалізована для реєстрації, не буде застосовуватися для форми логіну. При логіні достатньо перевірити лише наявність вмісту у полі електронної пошти та наявність вмісту у полі пароля.

Після завершення імплементації та вдосконалення клієнтської частини веб-застосунку, зокрема, форм реєстрації та логіну з їхньою валідацією, наступним логічним і критично важливим етапом у циклі розробки є створення та конфігурація бази даних. Саме база даних є фундаментом для зберігання всіх необхідних даних застосунку – інформації про користувачів, їхні публікації, взаємодії та багато іншого.

Наступним критично важливим етапом у циклі розробки є створення та налаштування серверної інфраструктури, зокрема бази даних. База даних є фундаментом будь-якого динамічного веб-застосунку, оскільки саме в ній зберігаються всі користувацькі дані, інформація про контент та інші суттєві відомості, що дозволяють застосунку функціонувати.

Перед безпосереднім створенням бази даних необхідно виконати ряд підготовчих кроків, а саме необхідно встановити MySQL та інструменти для роботи з нею. Це включає в себе як основний серверний програмний пакет, так і клієнтські утиліти, які дозволяють взаємодіяти з базою даних. Одним з найбільш зручних та поширених інструментів для візуального проектування, розробки та адміністрування баз даних MySQL є MySQL Workbench. Цей графічний інтерфейс надає широкі можливості для моделювання даних, виконання SQL-запитів, управління користувачами та моніторингу продуктивності.

У програмі MySQL Workbench підключимось до серверу, підключення вимагає вказання параметрів доступу до серверу, таких як адреса хоста, у нашому випадку це localhost, порт, ім'я користувача та пароль. Через контекстне меню створимо нову БД, з назвою social[34], у якій ми надалі будемо проектувати таблиці та визначати зв'язки між ними, для подальшої взаємодії з даними.

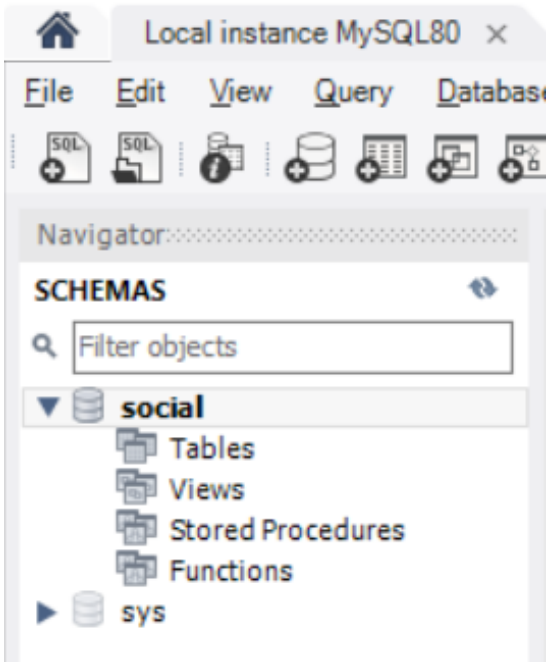


Рисунок 2.9 – Створенна БД

Наступним фундаментальним етапом у розробці бекенду є проектування структури таблиць, а також визначення зв'язків між ними. Це є ключовим кроком для ефективного зберігання, управління та взаємодії з даними в рамках веб-застосунку. Правильно спроектована схема бази даних забезпечує цілісність даних, оптимізує продуктивність запитів та спрощує подальший розвиток функціоналу.

Першою і найважливішою таблицею, яку ми створимо, буде таблиця для користувачів, оскільки всі інші сутності (публікації, сторіс, вподобання, коментарі, підписки) безпосередньо пов'язані з користувачами. Таблиця users буде містити всю необхідну інформацію про кожного зареєстрованого

користувача. Детальна структура цієї таблиці, як показано на рисунку 2.10, включає наступні поля [35]:

– id;
– usrename;
– email;
– pasword;
– name;
– coverPic;
– profilePic;
– sity;
– website.

Поле id буде виступати як первинний ключ таблиці users. Воно забезпечуватиме унікальний числовий ідентифікатор для кожного користувача, що є критично важливим для однозначної ідентифікації записів та встановлення зв'язків з іншими таблицями.

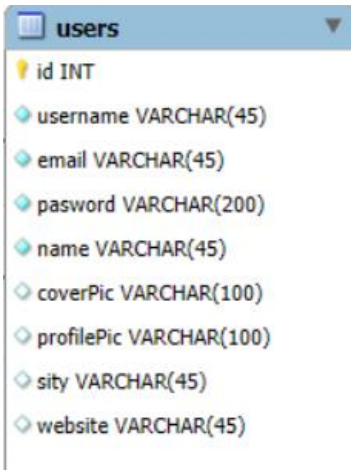


Рисунок 2.10 – Таблиця users

Після створення базової таблиці users, логічним наступним кроком є визначення структури таблиці для зберігання основного контенту соціального застосунку, тобто публікацій (posts). Ця таблиця є центральною для відображення динамічного контенту користувачів і як показано на рисунку 2.10, міститиме всі

необхідні дані для кожної окремої публікації. Таблиця posts буде складатися з наступних полів:

– id;
– desc;
– img;
– userId;
– createdAt.

Поле userId поле є зовнішнім ключем, який встановлює прямий зв'язок між таблицею posts та таблицею users. Воно посилається на id користувача з таблиці users, який створив дану публікацію. Це ключовий елемент для забезпечення реляційної цілісності: кожна публікація повинна належати конкретному користувачеві. Через це поле можна легко отримати інформацію про автора будь-якої публікації.

Поле createdAt автоматично фіксуватиме дату і час створення публікації. Використання типу даних DATETIME дозволяє зберігати як дату, так і час з високою точністю. Це поле є важливим для хронологічного відображення публікацій, наприклад, у стрічці новин, сортування та аналізу активності користувачів.

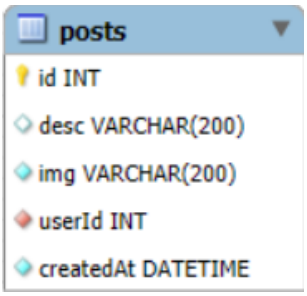


Рисунок 2.11 – Таблиця публікацій

Після створення таблиць для користувачів та їх публікацій, наступним кроком є визначення структури таблиці для зберігання коментарів (comments). Коментарі є невід'ємною частиною інтерактивності соціальних мереж,

дозволяючи користувачам обговорювати та взаємодіяти з контентом. Як показано на рисунку 2.12, таблиця comments буде містити ключову інформацію про кожен залишений коментар. Таблиця comments буде складатися з наступних полів:

– id;
– desc;
– createdAt;
– userId;
– postId.

Поле userId є зовнішнім ключем, який встановлює зв'язок між таблицею comments та таблицею users. Воно посилається на id користувача з таблиці users, який залишив даний коментар. Цей зв'язок є критично важливим для ідентифікації автора коментаря та відображення його імені або аватара поруч із коментарем.

Поле postId також є зовнішнім ключем, який встановлює зв'язок між таблицею comments та таблицею posts. Воно посилається на id публікації з таблиці posts, до якої відноситься даний коментар. Цей зв'язок забезпечує, що кожен коментар чітко асоційований з конкретною публікацією, дозволяючи відображати всі коментарі під відповідним дописом.

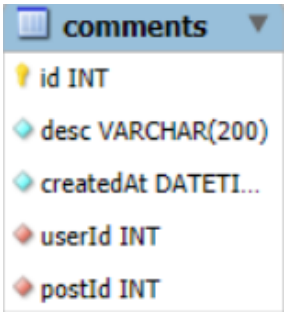


Рисунок 2.12 – Таблиця коментарів

Продовжуючи розбудову структури бази даних для соціального веб-застосунку, важливим елементом є реалізація функціоналу історій (stories), які стали надзвичайно популярним форматом обміну контентом. Для зберігання цих короткочасних візуальних публікацій буде створена окрема таблиця stories. Як

видно на рисунку 2.13, ця таблиця матиме спрощену, але ефективну структуру, що дозволяє легко асоціювати історії з їхніми авторами.

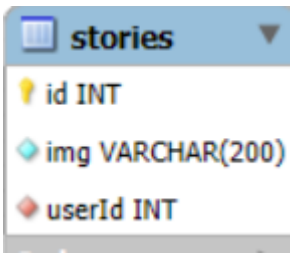


Рисунок 2.13 – Таблиця сторіс

У соціальних мережах можливість встановлювати зв'язки між користувачами, іншими словами підписки, є фундаментальним функціоналом такого типу застосунків. Для реалізації цього механізму в базі даних створимо спеціальну таблицю relationships (рисунок 2.14). Ця таблиця є класичним прикладом таблиці зв'язку (junction table), що використовується для реалізації зв'язку багато-до-багатьох між сутностями, в даному випадку між користувачами. Таблиця relationships буде складатися з наступних полів:

– id;
– followedUserId;
– followedUserId.

Поле followedUserId є зовнішнім ключем, який вказує на id користувача з таблиці users, який ініціював підписку (тобто, той, хто підписується). Цей зв'язок є критично важливим для розуміння, хто саме є "фоловером" у даній парі відносин. Воно посилається на id у таблиці users.

Поле followedUserId це поле також є зовнішнім ключем, який вказує на id користувача з таблиці users, на якого підписалися. Цей зв'язок необхідний для ідентифікації підписника в парі відносин. Воно також посилається на id у таблиці users.

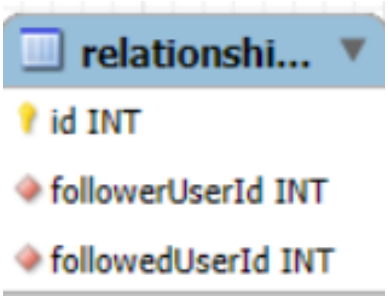


Рисунок 2.14 – Таблиця підписок

Після завершення проектування та імплементації структури бази даних, наступним фундаментальним етапом у розробці серверної частини веб-застосунку є створення API (Application Programming Interface). API дозволяє клієнтській частині взаємодіяти з базою даних та логікою на сервері, обмінюючись даними. Для побудови ефективного та масштабованого API буде використана популярна та гнучка бібліотека для Node.js під назвою Express.js.

Express.js надає мінімалістичний та швидкий фреймворк для створення веб-серверів та RESTful API. Його ключовою концепцією є маршрутизація, яка дозволяє визначати, як сервер реагує на різні HTTP-запити (GET, POST, PUT, DELETE) за певними URL-шляхами. У контексті нашого соціального веб-застосунку, нам необхідно створити окремі набори маршрутів для кожного типу сутностей, з якими ми працюємо: користувачів, публікацій, коментарів, вподобань та автентифікації.

Кожен маршрут у Express.js складається з визначеного HTTP-методу наприклад router.get(), та URL-шляху і одного або кількох обробників запитів. Як показано на рисунку 2.15.

```
server > routes > users.js > default
1   import Express from "express"
2   import { getUser } from "../controllers/user";
3
4   const router = Express.Router();
5
6   router.get('/getUser/:userId' , getUser)
7
8   export default router
```

Рисунок 2.15 – Приклад маршрута users

Для забезпечення чистоти та модульності коду, кожен набір маршрутів буде розміщуватися в окремому файлі у директорії server/routes. Кожен такий файл експортуватиме свій Express.Router(), який потім імпортуватиметься у головний файл сервера та буде імплементований за допомогою app.use(), що дозволить організувати API за логічними блоками.

Цей підхід з використанням Express.js та розділенням на маршрути й контролери є стандартною та рекомендованою практикою для побудови RESTful API, забезпечуючи чіткість, масштабованість та легкість у підтримці серверної частини веб-застосунку [36].

Кожен маршрут має асоційований з ним скрипт-обробник, який буде виконаний при отриманні відповідного запиту. Ці скрипти відповідають за виконання всіх необхідних операцій. Зазвичай, логіка цих скриптів організовується за допомогою контролерів. Як видно на прикладі з users.js, функція getUser імпортується з окремого файлу контролера, що сприяє кращій структуризації коду та його підтримці.

Контролери є центральним місцем, де реалізується бізнес-логіка. Вони отримують інформацію із запиту, наприклад userId з параметрів URL, дані з тіла POST запиту. Також вони взаємодіють з базою даних використовуючи, у нашому випадку, бібліотеки для роботи з MySQL (бібліотека mysql12), і залежно від результату цих операцій, формують відповідь для клієнта. Контролер може, наприклад, отримати дані про користувача з бази даних, обробити їх, і потім відправити їх назад у вигляді JSON-об'єкта.

Після визначення структури бази даних та налаштування маршрутизації за допомогою Express.js, ключовим етапом у розробці серверної логіки є створення контролерів. Контролери є мозком бекенду, відповідаючи за обробку вхідних запитів, взаємодію з базою даних та формування відповідей для клієнта. Розпочнемо з реалізації контролера авторизації, який буде включати функції для реєстрації, логіну та розлогіну користувачів.

Функція реєстрації нового користувача є першою та однією з найважливіших, оскільки вона дозволяє новим користувачам створювати облікові записи в системі. Її логіка, як показано на рисунку 2.16, складається з кількох послідовних кроків, спрямованих на забезпечення безпеки та унікальності даних.

Рисунок 2.16 – Функція реєстрації

Функція реєстрації нового користувача розпочинає свою роботу з перевірки наявності користувача в базі даних. Вона формує SQL запит SELECT FROM users WHERE username = ?, де плейсхолдер замінюється на ім'я користувача, отримане з вхідного запиту. Якщо під час виконання цього запиту виникає системна помилка, функція негайно повертає HTTP-статус 500 та відповідне JSON-повідомлення про помилку, інформуючи про збій на сервері. У випадку, коли запит до бази даних виявляє існуючого користувача з таким самим іменем, функція негайно надсилає клієнту HTTP-статус 409 разом із повідомленням, що такий користувач вже є в системі.

Якщо ж користувача з таким іменем не виявлено, процес переходить до етапу безпечного зберігання пароля. Тут застосовується бібліотека bcrypt для хешування пароля, що є обов'язковою практикою безпеки. Спочатку генерується унікальна salt за допомогою bcrypt.genSaltSync(10), яка додає випадковість до процесу хешування і захищає від атак за попередньо обчисленими хешами. Отриманий з вхідного запиту пароль потім хешується разом із цією сіллю за

допомогою bcrypt.hashSync, створюючи безпечне представлення пароля, яке буде збережене у базі даних замість відкритого тексту.

Нарешті, після успішної перевірки на унікальність та надійного хешування пароля, функція переходить до створення нового користувача та запису його даних у БД. Формується SQL-запит INSERT INTO users ('username', 'email', 'password', 'name') VALUES (?), а значення для вставки беруться з вхідного запиту та щойно згенерованого хешованого пароля. Цей запит відправляється до бази даних. Якщо під час операції вставки виникає будь-яка помилка, сервер знову повертає HTTP-статус 500 (Internal Server Error) з відповідним повідомленням. У протилежному випадку, коли новий користувач успішно доданий до бази даних, функція завершується відправкою HTTP-статусу 200 (OK) та підтверджувального повідомлення "User has been created", що сигналізує клієнту про успішну реєстрацію облікового запису. Також функція реєстрації, окрім своєї основної задачі, інтегрує два важливі механізми для захисту інформації.

По-перше, застосовується хешування паролів за допомогою бібліотеки bcrypt. Це забезпечує, що паролі користувачів ніколи не зберігаються у відкритому вигляді. Оскільки хешування є односторонньою функцією, навіть у разі витоку даних з бази даних, зловмисник не зможе відновити реальні паролі користувачів, значно підвищуючи безпеку.

По-друге, при формуванні SQL-запитів для взаємодії з базою даних використано особливості синтаксису JavaScript для роботи з рядками та функціонал бібліотеки, що забезпечує захист від SQL-ін'єкцій. Замість прямої конкатенації рядків, використовуються плейсхолдери, а значення передаються окремо. Це дозволяє бібліотеці автоматично екранувати дані, нейтралізуючи будь-які потенційно шкідливі символи або фрагменти SQL-коду. Цей метод, відомий як параметризовані запити, є стандартом безпеки. Важливо, що цей спосіб буде застосовуватися в усіх інших функціях контролерів для забезпечення послідовного захисту.

Для перевірки коректності реалізованої функції реєстрації, ми використаємо сервіс Postman. Як показано на рисунку 2.17, буде сформований POST-запит до ендпоінту реєстрації, з необхідними даними користувача у тілі запиту. Це дозволить наочно перевірити логіку функції, її взаємодію з базою даних та коректність повернення відповідних HTTP-статусів та повідомлень (рисунок 2.18).

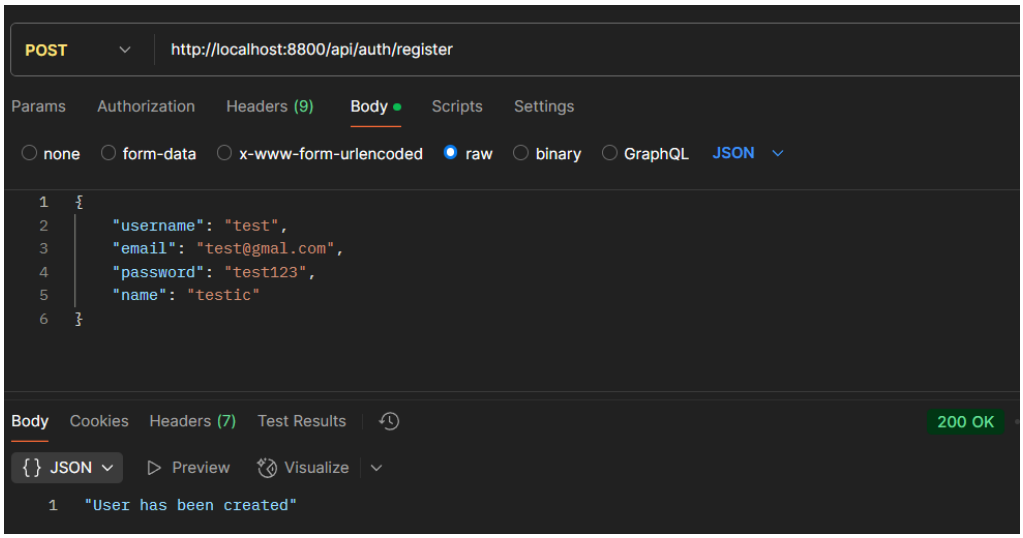


Рисунок 2.17 – Запит реєстрації користувача

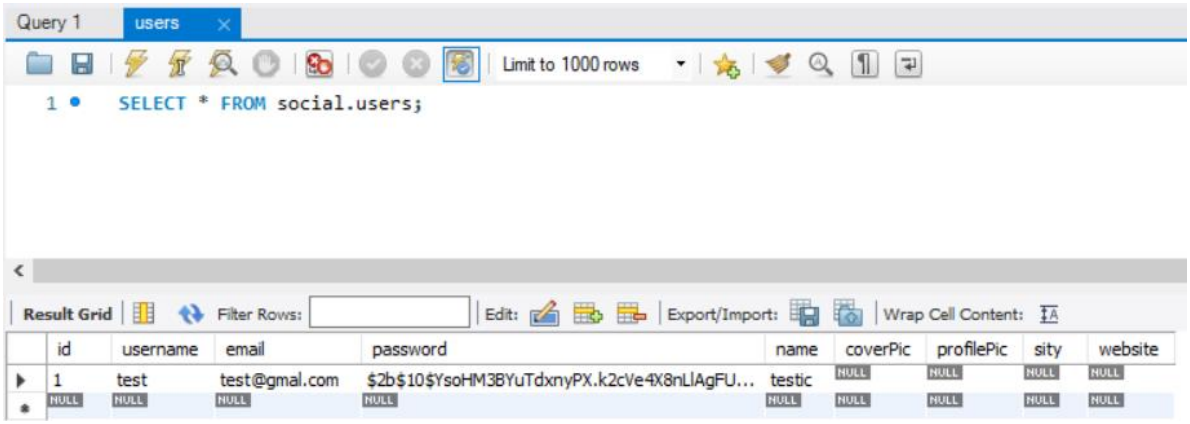


Рисунок 2.18 – Запис тестового користувача у таблиці

Далі створимо функцію логіну, ця функція не лише повинна давати змогу вже існуючим користувачам увійти в систему, а й імплементує сучасні механізми безпеки, зокрема використання JSON Web Tokens, для ефективного контролю

доступу та автентифікації[38]. Як показано на рисунку 2.18, процес логіну є багатоетапним і включає в себе перевірку облікових даних та генерацію токена.

```
export const login = (req, res)=>{
    const query = "SELECT * FROM users WHERE username = ?"
    // User exist check
    db.query(query, [req.body.username], (err, data) =>{
        if(err) return res.status(500).json(err.code)
        if(data.length === 0) return res.status(400).json("Invalid username or password")

        const checkPassword = bcrypt.compareSync(req.body.password, data[0].password)

        if (!checkPassword) return res.status(400).json("Invalid username or password")

        const {password, ...userData} = data[0]
        const token = jwt.sign({id: data[0].id}, process.env.SEKRET_KEY)

        res.cookie("accessToken", token, {
            httpOnly: true,
        }).status(200).json(userData)
    })
}
```

Риунок 2.19 – Код функції логіну

Процес логіну розпочинається з того, що сервер отримує запит від клієнта, що містить ім'я користувача та пароль. Першим етапом у функції логіну є перевірка наявності користувача з вказаним іменем у базі даних. Це здійснюється за допомогою SQL-запиту, аналогічного тому, що використовувався у функції реєстрації, для пошуку відповідного запису в таблиці users. Якщо під час виконання цього запиту виникає помилка, сервер негайно повертає HTTP-статус 500, сигналізуючи про внутрішню проблему. Якщо ж користувача з таким ім'ям не знайдено, функція повертає HTTP-статус 400? з повідомленням "Invalid username or password", уникаючи розкриття конкретної причини, для підвищення безпеки.

У разі, якщо користувача знайдено, відбувається ключовий етап співставлення паролів. Введений користувачем, пароль хешується за допомогою тієї ж бібліотеки bcrypt, яка використовувалася при реєстрації. Отриманий хеш потім порівнюється зі збереженим у базі даних хешем пароля користувача за допомогою функції bcrypt.compareSync(). Це порівняння є безпечним, оскільки воно не розкриває сам пароль, а лише підтверджує його відповідність. Якщо хеші

не співпадають, це означає невірний пароль, і функція знову повертає HTTP-статус 400 з повідомленням "Invalid username or password".

Після успішної перевірки пароля, система вважає користувача автентифікованим. На цьому етапі генерується JWT токен. JWT токен – одни із стандаартів токену, на основі JSON, що використовується для підтверження особи користувача. Він містить закодовану інформацію про користувача, таку як його унікальний ідентифікатор, а також може включати інші дані, наприклад, ролі, термін дії токена та інші потрібні параметри. Важливою частиною токена є його підпис, згенерований за допомогою секретного ключа сервера. Цей підпис гарантує цілісність токена: будь-яка спроба його підробки або зміни буде виявлена при перевірці підпису.

Згенерований JWT токен потім відправляється назад клієнту. Зазвичай, це відбувається у вигляді HTTP-кукі. Також було використано додатковий параметр httpOnly, із значенням true, що для кукі є важливою мірою безпеки, оскільки це запобігає доступу JavaScript на стороні клієнта до цього кукі, зменшуючи ризик XSS-атак. Зберігання токена в кукі дозволяє браузеру автоматично надсилати його з кожним наступним запитом до сервера. Крім того, функція повертає решту даних користувача (userData), за винятком пароля, для використання на клієнтській стороні.

Після успішної реалізації функцій автентифікації та авторизації, включно з безпечним логіном за допомогою JWT токенів, наступним етапом у розробці серверної частини є створення контролерів для маніпуляції даними, що зберігаються в базі даних. Це включає операції отримання, додавання, редагування та видалення інформації, зокрема для постів, коментарів, лайків, та інших сутностей.

Почнемо з контролерів для постів, оскільки вони є центральним елементом вмісту у соціальному застосунку. Перед тим, як дозволити будь-яку операцію з базою даних, критично важливо переконатися, що користувач авторизований. Це

ключовий аспект безпеки, який запобігає несанкціонованому доступу та маніпуляціям з даними.

Зображений на рисунку 2.19, приклад коду контролеру users, містить функцію getPosts, яка розпочинається з перевірки автентифікації. Вона намагається отримати JWT токен з кукі запиту, якщо токен відсутній, це означає, що користувач не залогінений, і функція негайно повертає HTTP-статус 401 з повідомленням "Not logged in!".

```
export const getPost = (req, res) => {
    const userId = req.query.userId;
    const token = req.cookies.accessToken;
    if (!token) return res.status(401).json("Not logged in!");

    jwt.verify(token, "secretkey", (err, userInfo) => {
        const query = "SELECT p.*, u.id AS userId, name, profilePic FROM posts AS p JOIN users AS u ON (u.id = p.userId)"
        const values = userId !== "undefined" ? [userId] : [userInfo.id, userInfo.id];

        db.query(query, values, (err, data) => {
            if (err) return res.status(500).json(err);
            return res.status(200).json(data);
        });
    })

}
```

Рисунок 2.20 – Код функціїї отримання даних

Якщо токен присутній, відбувається його верифікація за допомогою бібліотеки jsonwebtoken та секретного ключа сервера, цей крок є важливим для забезпечення кофіденційності та цілісності інформації.

Якщо токен недійсний, наприклад підроблений, змінений або прострочений, jwt.verify поверне помилку і функція негайно поверне HTTP-статус 403, оскільки користувач не має прав доступу.

Якщо токен успішно верифікований, з нього витягується інформація про користувача userInfo, зокрема його id. Цей id буде використовуватися для фільтрації даних та забезпечення, що користувач бачить лише той контент, до якого має доступ, наприклад пости від друзів або власні пости.

Далі формується SQL-запит для отримання постів та супутньої до них інформації. Запит виконує об'єднання таблиць posts та users. Це дозволяє отримати не тільки дані про сам пост, але й додаткову інформацію про його

автора, таку як його ім'я, ідентифікатор користувача та посилання на зображення профілю цього користувача.

Запит також адаптується залежно від того, чи запитуються пости конкретного користувача. Якщо userId передано і воно не є "undefined", то пости фільтруються за цим userId. В іншому випадку, якщо запитуються пости для стрічки новин, використовуються дані userInfo.id з верифікованого токена, щоб показати пости, релевантні поточному користувачу, наприклад пости, що належать йому самому або пости від тих, на кого він підписаний.

Цей підхід до імплементації контролерів, з незмінним акцентом на обов'язкову автентифікацію та авторизацію перед будь-яким доступом або маніпуляцією з даними, є фундаментальним для побудови безпечного та функціонального веб-застосунку. Після реалізації цих операцій для постів, аналогічні функції будуть впроваджені й для інших контролерів, що працюють з різними сутностями бази даних, такими як коментарі та лайки. Таким чином, буде забезпечено повноцінний функціонал як для клієнтської, так і для серверної частини, дозволяючи системі ефективно працювати з великими обсягами даних і надавати користувачам розширені можливості взаємодії.

Також критично важливим кроком у процесі налаштування бази даних буде забезпечення належного розмежування прав доступу, для забезпечення конфіденційності, доступності та цілісності даних. Замість використання одного акаунта з повними привілеями, наприклад стандартного root, ми створимо набір спеціалізованих адміністративних акаунтів. Кожен з цих акаунтів буде відповідальним за конкретні задачі, отримуючи при цьому лише ті привілеї, які необхідні для виконання його функцій. Такий підхід реалізує принцип найменших привілеїв (principle of least privilege), значно зменшуючи ризик випадкових помилок або зловмисних дій.

Для реалізації цього завдання ми використаємо вбудований графічний інструмент MySQL Workbench. Він надає зручний та інтуїтивно зрозумілий

інтерфейс для управління користувачами та їхніми правами доступу безпосередньо до сервера MySQL.

Створимо підключення до серверу БД, далі перейдемо у вкладку Servers > Users and Privileges, та створимо новий акаунт. Почнемо з адміністратора користувачів, який буде відповідальний за створення, зміну, видалення облікових записів користувачів БД, надання та відкликання привілеїв іншим користувачам. Основні, обрані в селекторі, привілеї ролі це CREATE USER, DROP USER, ALTER USER, RENAME USER, SELECT (для системнх таблиць), RELOAD. Заповнимо поля для авторизації цього адміністратора, а також оберемо зберігання паролю у хешованому алгоритмом функції SHA256 вигляді (рисунок 2.21).

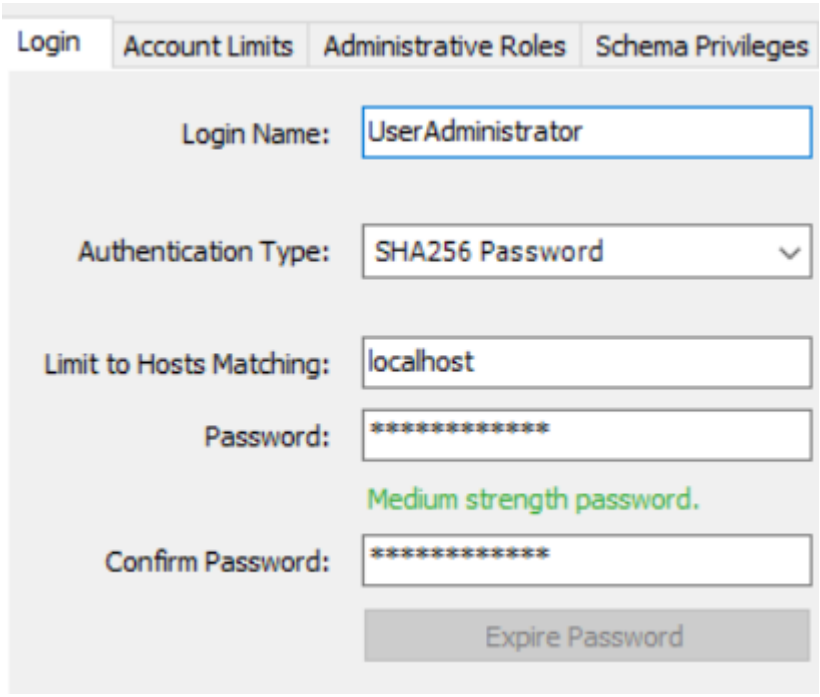


Рисунок 2.21

Створимо адміністратора баз даних, ця рольбуде відповідальна за зміну та видалення баз даних (схем), таблиць, індексів, подань (views), збережених процедур та функцій, тригерів. Основні, обрані в селекторі, привілеї ролі це CREATE DATABASE, ALTER ROUTINE, CREATE ROUTINE, CREATE TEMPORALY TABLES, CREATE VIEW, EVENT, INDEX, TRIGGER, CREATE,

ALTER, DROP, SELECT (для вибраних таблиць). Для цієї ролі нам підійде пресет з назвою DBDesigner (риисунок 2.23, рисунок 2.24).

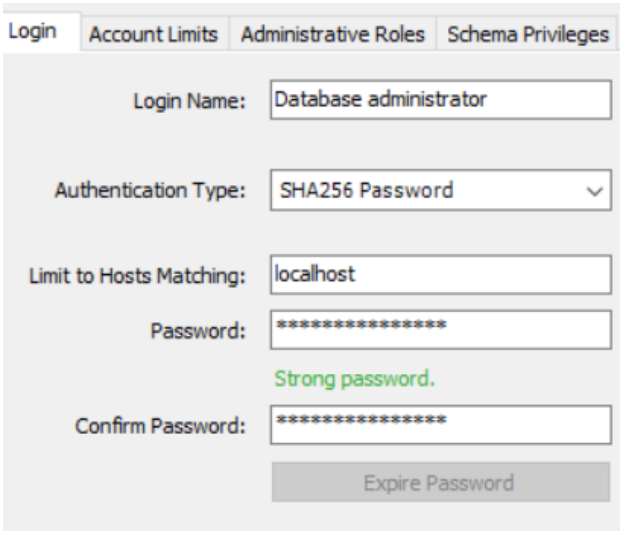


Рисунок 2.22 – Форма додавання адміністартора ЬД

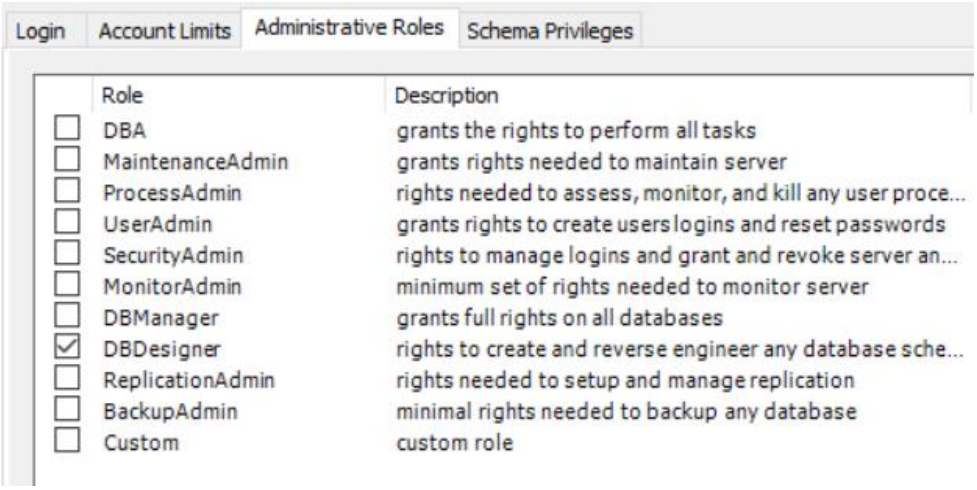


Рисунок 2.23

Далі створимо роль адміністратора відновлення, ця роль відповідальна за виконання операцій резервного копіювання, наприклад, за допомогою mysqldump або інструментів MySQL Enterprise Backup, та відновлення даних. Основні, обрані в селекторі, привілеї ролі це PROCESS, REPLICATION CLIENT, SUPER, SHOW DATABASES, SHOW VIEW, SELECT. Для цієї ролі нау підійте профіль BackupAdmin (рисунок 2.24, рисунок 2.25).

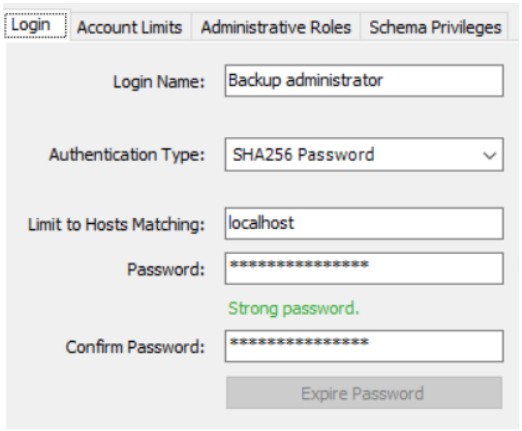


Рисунок 2.24 – Форма додавання адміністартора відновлення

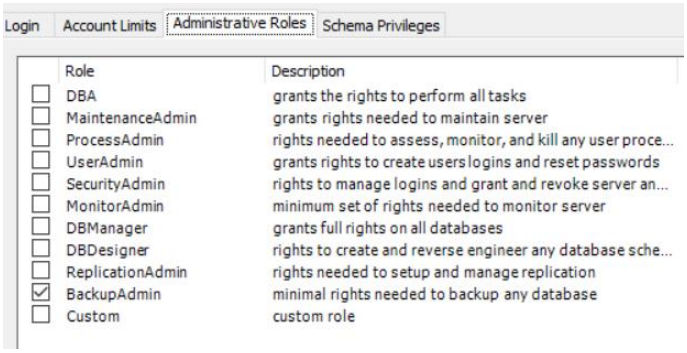


Рисунок 2.25

Останнім створимо роль адміністратора моніторингу, ця роль відповідальна за моніторинг стану сервера, активних процесів, продуктивності запитів, використання ресурсів. Цей акаунт має мати лише доступ для читання системної інформації. Основні, обрані в селекторі, привілеї ролі це PROCESS, REPLICATION CLIENT, SUPER, SHOW DATABASES, SHOW VIEW, SELECT. Для цієї ролі нav підійте профіль BackupAdmin (рисунок 2.26).

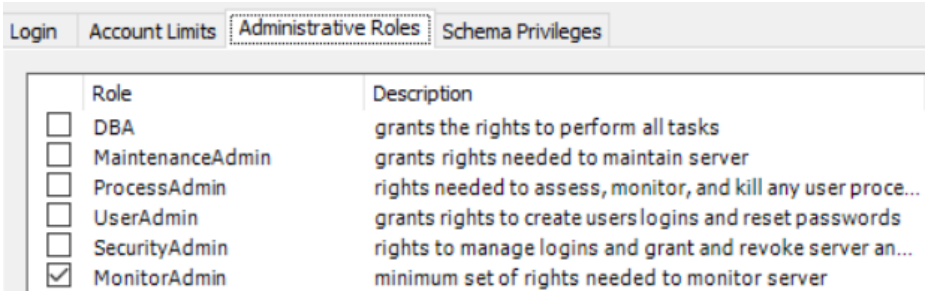


Рисунок 2.26

Наступним кроком налаштуємо резервне копіювання. Воно є однією з найважливіших задач адміністрування. Воно забезпечує можливість відновлення даних у випадку різноманітних проблем: збоїв обладнання, програмних помилок, випадкового видалення чи зміни даних, зловмисних атак тощо. Без надійної стратегії резервного копіювання існує високий ризик втрати цінної інформації.

Для безпечного збереження облікових даних, які будуть використовуватись для скрипту запуску mysqldump, необхідно запустити наступну команду "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql_config_editor.exe" set --login-path=mylocalbackup --host=localhost --user=Backup_admin --password".

Для автоматизації запуску скрипта створимо файл backup_mysql.ps1, у якому опишемо змінні для нашого скрипта та запуск самого скрипта (рисунок 2.27).

backup_mysql.ps1 U

backup_mysql.ps1

```
$backupDir = "D:\Projects\Social App\Backup"

$loginPath = "mylocalbackup"

$databaseName = "social"

$mysqldumpPath = "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqldump.exe"

$timestamp = Get-Date -Format "yyyyMMdd_HHmmss"

$backupFile = Join-Path $backupDir "$databaseName_$timestamp.sql"

Write-Host "Starting backup of $databaseName to $backupFile..."

try {
    & $mysqldumpPath --login-path=$loginPath $databaseName | Out-File -Path $backupFile -Encoding UTF8
    Write-Host "Backup completed successfully."
} catch {
    Write-Error "ERROR during backup: $($_.Exception.Message)"
}

Write-Host "Script finished."
```

Рисунок 2.27 – Скрипт бекапу

Після написання скрипта, наступним кроком буде налаштування автоматичного запуску за допомогою планувальника завдань Windows. Спочатку запустимо програму, у правій панелі дій потрібно натиснути кнопу Create Task. У вкладці General потрібно ввести назву задачі, обрати користувача від імені якого буде відбуватись запуск задачі (рисунок 2.28). Цей користувач повинен мати

доступ до запуску mysqldump , читання файлу .mylogin.cnf та можливість запису у папку з бекапами[40].

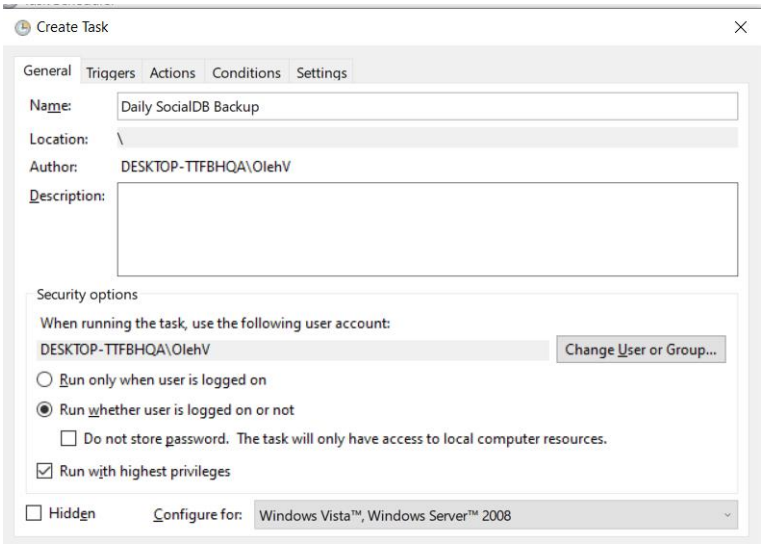


Рисунок 2.28

У вкладці трігерів потрібно натиснути кнопку New, у вікні що з’явилось обрати час виконання "щоденно" та вказати час і дату першого виконання завдання (рисунок 2.29).

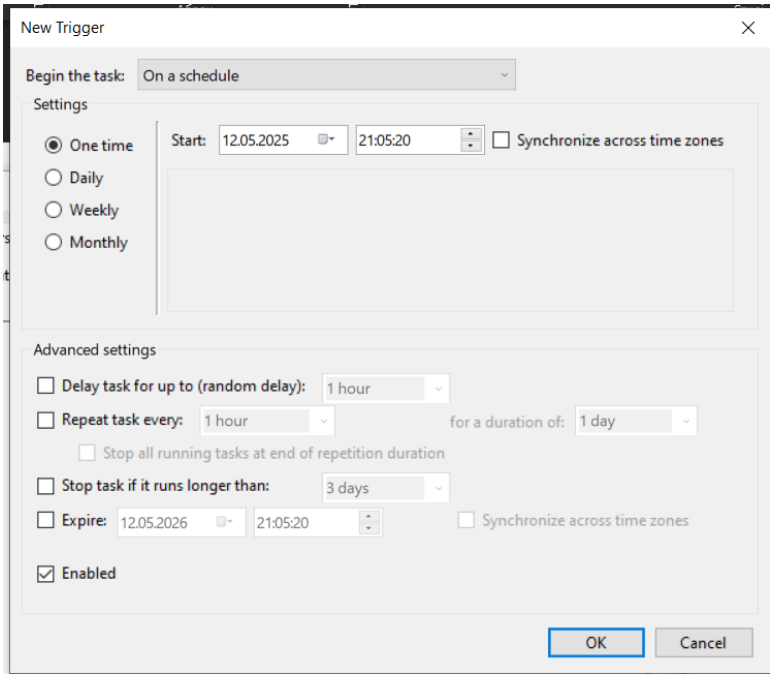


Рисунок 2.29

У вкладці Actions потрібно натиснути кнопку New, у новому вікні обрати тип дії "Start program", обрати шлях де знаходиться скрипт, а саме D:\Projects\Social App\Backup\backup_mysql.ps1 та натиснути Ок (рисунок 2.30).

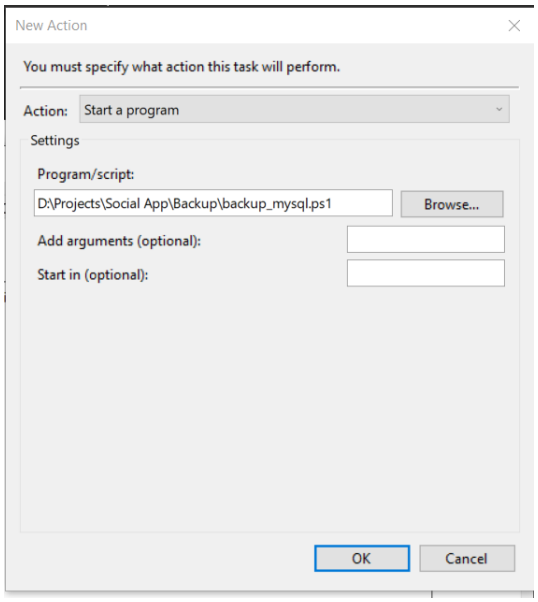


Рисунок 2.30

Ми налаштували щоденне, одноразове резервне копіювання бази даних. Для скрипта було використано профіль Admin Backup та зберегли облікові дані для запуску у безпечному вигляді, для облікових даних входу в msqldump ми використали захищений спосіб збереження у хешованому вигляді.

2.3Розробка системи оробки зібраної інформації

Після успішного створення функціональної клієнтської та серверної частин веб-застосунку, що включають систему авторизації, валідацію форм, управління базою даних та реалізацію API для маніпуляції даними, ми досягли важливого етапу, коли в системі вже накопичується значна кількість інформації. Це відкриває можливості для аналізу даних, що є цінним для розуміння поведінки користувачів, ефективності контенту та загального стану застосунку.

Для проведення аналізу даних буде використана потужна та універсальна мова програмування Python. Python є ідеальним вибором для цього завдання завдяки своїй простоті синтаксису, величезній екосистемі бібліотек для роботи з даними наприклад Pandas для маніпуляцій з табличними даними, Matplotlib/Seaborn для візуалізації, SciPy/NumPy для наукових обчислень, та широким можливостям для підключення до різних баз даних, включаючи MySQL. Python дозволить нам писати скрипти, які зможуть витягувати дані з нашої бази даних social, обробляти їх, виявляти закономірності, генерувати звіти або навіть формувати рекомендації.

Оскільки аналіз даних часто є процесом, який потребує періодичного виконання для підтримки актуальності інформації, ми імплементуємо механізм автоматичного запуску аналітичних скриптів. Для цього буде використано планувальник подій Windows (Windows Task Scheduler). Цей вбудований інструмент операційної системи Windows дозволяє створювати та керувати завданнями, які запускаються автоматично у визначений час або у відповідь на певні системні події. Ми налаштуємо планувальник завдань таким чином, щоб він кожних 12 годин запускав Python-скрипт аналізу. Це забезпечить регулярне оновлення аналітичних даних, дозволяючи нам завжди мати актуальне уявлення про динаміку розвитку застосунку та активність користувачів.

Алгоритм скрипта буде наступним. Для забезпечення підключення до бази даних у Python використовується бібліотека mysql.connector. Ця бібліотека є офіційним коннектором MySQL для Python, надаючи всі необхідні інструменти для взаємодії з MySQL-сервером, виконання запитів та отримання результатів.

На початку скрипта визначається словник DB_CONFIG, який містить всі необхідні параметри для встановлення з'єднання з базою даних. Ці параметри включають в себе адресу сервера бази даних, ім’я користувача БД, пароль і назву БД. Ця конфігурація надає скрипту всю необхідну інформацію для ініціалізації з'єднання. Після визначення цих параметрів, скрипт викликає функцію mysql.connector.connect(), передаючи їй параметри з DB_CONFIG (рисунок 2.31).

У разі успішного підключення, буде створено об'єкт з'єднання, який дозволить скрипту виконувати SQL-запити.

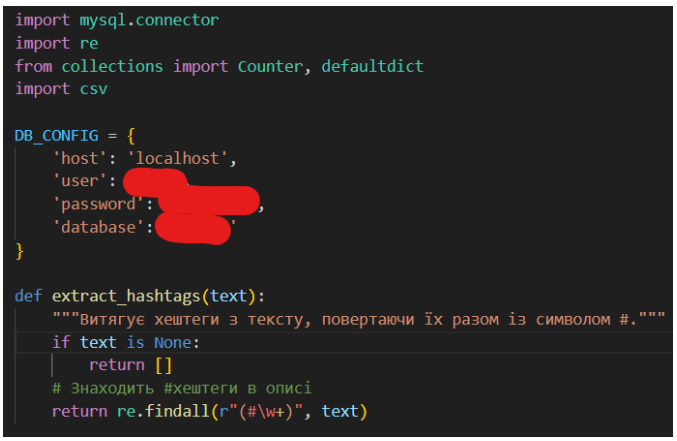


Рисунок 2.31

Після успішного встановлення з'єднання Python-скрипта з базою даних MySQL, наступним кроком в алгоритмі аналізу даних є збір необхідної інформації. Основним фокусом на цьому етапі є отримання всіх наявних постів та супутніх даних, зокрема кількості вподобань для кожного з них. Далі, скрипт переходить до детального аналізу текстового вмісту кожного посту для вилучення хештегів.

Як показано на рисунку 2.32, процес починається з ініціалізації змінних conn та cursor, які згодом будуть використовуватися для управління підключенням до бази даних та виконання запитів відповідно. Блок try забезпечує спробу встановлення з'єднання та створення об'єкта курсора. Використання dictionary=True при створенні курсора є важливим, оскільки це дозволяє отримувати результати запитів у вигляді словників, де ключами є назви стовпців, що значно полегшує доступ до даних та їх подальшу обробку.

Для отримання всіх постів разом з кількістю лайків для кожного з них, формується SQL-запит. Після визначення запиту, він виконується за допомогою cursor.execute(query_posts_and_likes), а отримані результати витягуються за допомогою posts_data = cursor.fetchall(). Якщо ж posts_data виявляється порожнім,

це означає, що у базі даних не знайдено жодного посту, про що буде виведено відповідне повідомлення "Не знайдено жодного поста в базі даних." і скрипт завершить свою роботу.

```
conn = None
cursor = None
try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    # 1. Отримати всі пости разом з кількістю лайків
    # Ми використовуємо LEFT JOIN на випадок, якщо пост не має лайків
    query_posts_and_likes = """
    SELECT
        p.id AS post_id,
        p.desc AS post_description,
        (SELECT COUNT(*) FROM likes WHERE postId = p.id) AS like_count
    FROM
        posts p;
    """
    cursor.execute(query_posts_and_likes)
    posts_data = cursor.fetchall()

    if not posts_data:
        print("Не знайдено жодного поста в базі даних.")
        return
```

Рисунок 2.32

Наступним кроком є перегляд кожного отриманого посту та витягування всіх хештегів з його опису. Для цього використовується функція extract_hashtags(text), яка імплементується за допомогою бібліотеки регулярних виразів (re). Функція extract_hashtags приймає один аргумент, який являє собою текстовий опис посту. Функція містить попередню перевірку, якщо отриманий текст є порожнім, функція повертає порожній список, що забезпечує коректну обробку випадків, коли опис може бути відсутнім. Основна логіка витягування хештегів полягає у використанні регулярного виразу r"(#\w+)" з функцією re.findall(). Цей регулярний вираз спроектований для пошуку всіх входжень символу хештегу, за яким одразу слідує один або більше "словесних" символів (\w+), що охоплює літери, цифри та підкреслення. Результатом виконання re.findall() є список усіх знайдених хештегів, кожен з яких включає сам символ хештегу.

Код зображений на рисунку 2.33, ітерує по кожному елементу у списку posts_data, який містить інформацію про пости. Для кожного post у цьому списку спочатку видобувається його опис та кількість лайків. Далі, скрипт викликає раніше описану функцію extract_hashtags(description), щоб витягнути всі хештеги з текстового опису поточного поста. Результатом цієї операції є список hashtags_in_post_text.

```
# 2. Обробити кожен пост
for post in posts_data:
    description = post['post_description']
    likes_on_post = post['like_count']

    # Витягуємо всі хештеги з опису
    hashtags_in_post_text = extract_hashtags(description)

    if hashtags_in_post_text:
        # Рахуємо кожне входження хештегу для загальної статистики використання
        all_hashtag_occurrences.update(hashtags_in_post_text)

        # Для підрахунку лайків, асоційованих з хештегом,
        # розглядаємо унікальні хештеги в межах одного поста
        unique_hashtags_in_this_post = set(hashtags_in_post_text)
        for tag in unique_hashtags_in_this_post:
            hashtag_total_likes[tag] += likes_on_post
```

Рисунок 2.33

Якщо hashtags_in_post_text не є порожнім, тобто, якщо в пості були знайдені хештеги, виконуються дві важливі операції для агрегації даних. По-перше, скрипт рахує кожне входження хештегу для формування загальної статистики використання. Це досягається шляхом оновлення об'єкта all_hashtag_occurrences який є типом collections.Counter і за допомогою методу update(), передаючи йому список hashtags_in_post_text. Такий підхід ефективно збільшує лічильник для кожного хештегу, що з'явився в пості.

По-друге, для підрахунку загальної кількості лайків, асоційованих з кожним хештегом, скрипт обробляє лише унікальні хештеги в межах поточного поста. Для цього створюється множина unique_hashtags_in_this_, що автоматично видаляє дублікати хештегів, якщо один і той же хештег був використаний кілька разів в одному описі. Потім, для кожного tag у цій множині унікальних хештегів, скрипт додає кількість лайків поточного поста до загальної суми лайків, асоційованих з

цим хештегом, що зберігається в структурі hashtag_total_likes яка є collections.defaultdict. Це дозволяє отримати не тільки частоту використання хештегів, але й їхню "популярність" за кількістю вподобань постів, у яких вони були використані. Цей етап є основним для перетворення необроблених даних на корисні метрики для аналізу контенту та трендів.

Після того, як скрипт обробив усі пости та зібрав необхідну статистику щодо хештегів, тобто їхню загальну кількість використань та суму лайків, асоційованих з кожним хештегом, він переходить до завершальних етапів свого алгоритму, а саме запису зібраних даних у файл у форматі CSV та коректного завершення роботи з базою даних.

Як показано на рисунку 2.34, перш ніж здійснювати запис, скрипт перевіряє, чи була взагалі зібрана статистика хештегів. Якщо all_hashtag_occurrences порожній, це означає, що жодного хештегу в постах не знайдено, про що виводиться відповідне повідомлення, і скрипт завершує свою роботу, уникаючи створення порожнього файлу.

```
# 3. Записати статистику в CSV файл
if not all_hashtag_occurrences:
    print("Не знайдено жодного хештега в постах для запису в CSV.")
    return

with open("stats.csv", mode='w', newline='', encoding='utf-8') as file:
    csv_writer = csv.writer(file)

    # Записуємо заголовок CSV
    csv_writer.writerow(['Хештег', 'Кількість використань', 'Загальна кількість лайків'])

    sorted_hashtags_by_occurrence = all_hashtag_occurrences.most_common()

    for tag, count in sorted_hashtags_by_occurrence:
        total_likes_for_tag = hashtag_total_likes.get(tag, 0)
        csv_writer.writerow([tag, count, total_likes_for_tag])

print(f"\nСтатистику по хештегах успішно записано у файл: {"stats.csv"}")
print(f"Загальна кількість унікальних хештегів: {len(all_hashtag_occurrences)}")
```

Рисунок 2.34

У разі наявності даних, скрипт відкриває файл stats.csv у режимі запису з кодуванням utf-8 та вказує newline, для коректної обробки переносів рядків, що важливо для CSV файлів. За допомогою об'єкта csv_writer, який надає бібліотека

csv, записується заголовок файлу: 'Хештег', 'кількість використань', 'Загальна кількість лайків'.

Далі, зібрана статистика хештегів сортується за частотою їх використання. Це досягається за допомогою all_hashtag_occurrences.most_common(), що повертає список кортежів (хештег, кількість використань), відсортованих за спаданням частоти. Скрипт ітерує по цьому відсортованому списку. Для кожного хештегу та його кількості використань, він отримує загальну кількість лайків, асоційованих з цим хештегом. В результаті ми отримуємо дані у вигляді – хештег, його частота використання та сума лайків, які записуються в окремий рядок CSV файлу за допомогою csv_writer.writerow([tag, count, total_likes_for_tag]).

Після успішного запису всіх даних, скрипт виводить повідомлення про те, що статистика по хештегах успішно записана у файл stats.csv. Також відображається загальна кількість унікальних хештегів, що були знайдені. На завершення, скрипт коректно закриває підключення до бази даних (cursor.close() та conn.close()), звільняючи ресурси. У всьому скрипті передбачені механізми обробки можливих помилок на етапах підключення до бази даних, виконання запитів та запису у файл, що забезпечує його стабільність та надійність.

В результаті виконання цього скрипта ми отримаємо CSV файли, які є зручним форматом для подальшого аналізу популярної тематики в соціальному застосунку. У масштабах нашого проекту, цього інструменту збору та аналізу даних буде цілком достатньо для отримання цінних інсайтів. Важливо зазначити, що скрипт виконується в межах сервера, без використання сторонніх сервісів, що є додатковим засобом захисту інформації, оскільки дані не залишають контрольоване середовище. Останнім кроком для автоматизації буде створення .exe білда цього Python-скрипта, який потім буде налаштований для виконання кожних 12 годин за допомогою планувальника завдань Windows, по аналогії запуску скрипта бекапу, забезпечуючи регулярне оновлення аналітичних даних.

3 ТЕСТУВАННЯ ЗАСТОСУНКУ ТА СИСТЕМИ ОБРОБКИ

3.1 Тестування клієнтської частини

Перше з чого варто почати, при тестуванні клієнтської частині – це форма реєстрації. Форма реєстрації має валідувати поля на вміст, тобто вони не повинні бути пустими, проведемо відповідне тестування (рисунок 3.1).

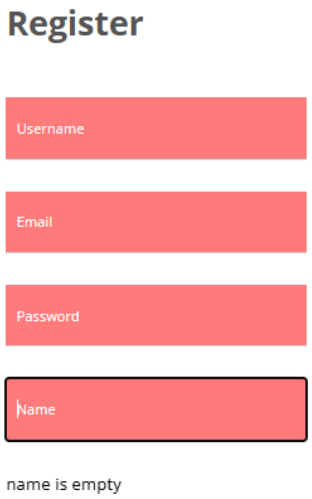


Рисунок 3.1 – Виділення пустих полів

Також поле паролю має бути заповненим відповідним паролем, який має містити велику літеру, цифру та спецсимвол (рисунок 3.2).

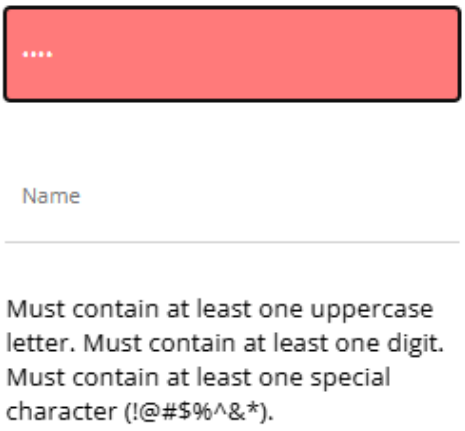


Рисунок 3.2 – Повідомлення про невірний формат паролю

Поле електронної пошти теж має перевірятись на правильність формату введеньої пошти (рисунок 3.3).

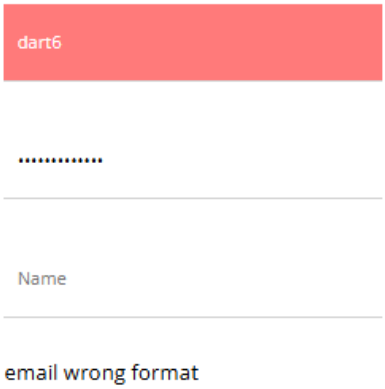


Рисунок 3.3 – Повідомлення про невірний формат пошти

Форма логіну теж має валідуватися, але в цьому випадку лише на наявність пустих полів, для унеможливлення відправки некоректного запиту.

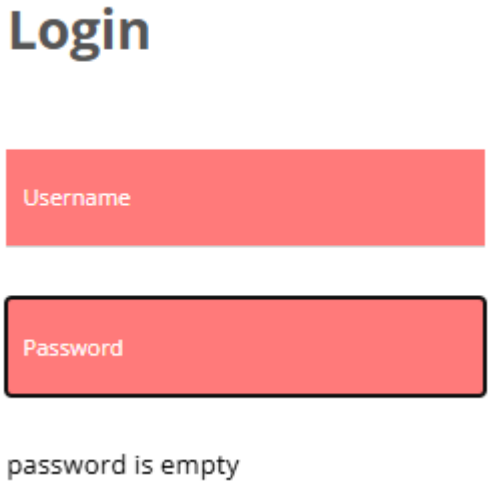


Рисунок 3.4 – Валідація форми логіну

Тестування клієнта проведено успішно, всі форми введення даних працюють коректно, та забезпечують валідацію даних.

3.2 Тестування серверннної частини

Тепер перейдемо до тестування серверної частини застосунку . Здійснимо перевірку одного із енпоінтів на спробу SQL ін’єкції. Для цього використаємо API логіну. Використаємо базовий варіант обходу пароля, у поле username введемо значення "' OR '1'='1" і відправимо запит.

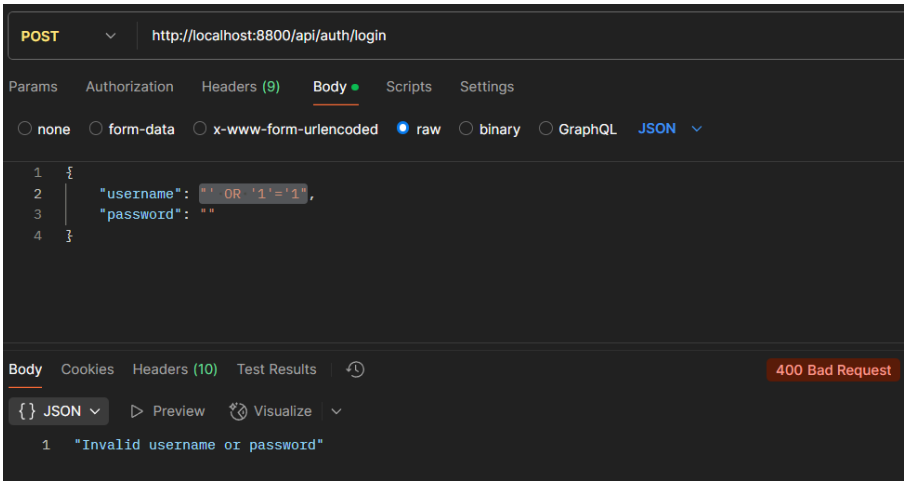


Рисунок 3.5 – Результат запиту

Як бачимо, сервер не дав виконати такий запит, так як запит формується через захищенний механізм функції. Тепер спробуємо ін’єкцію з коментарем, для цього в поле username введемо значення "test' --", що у деяких СУБД використовуються для коментування решти SQL-рядка. Запит може стати таким: SELECT * FROM users WHERE username = 'test' --' AND password = 'any_password';. Все після -- ігнорується, і сервер може спробувати увійти як користувач test без перевірки пароля.

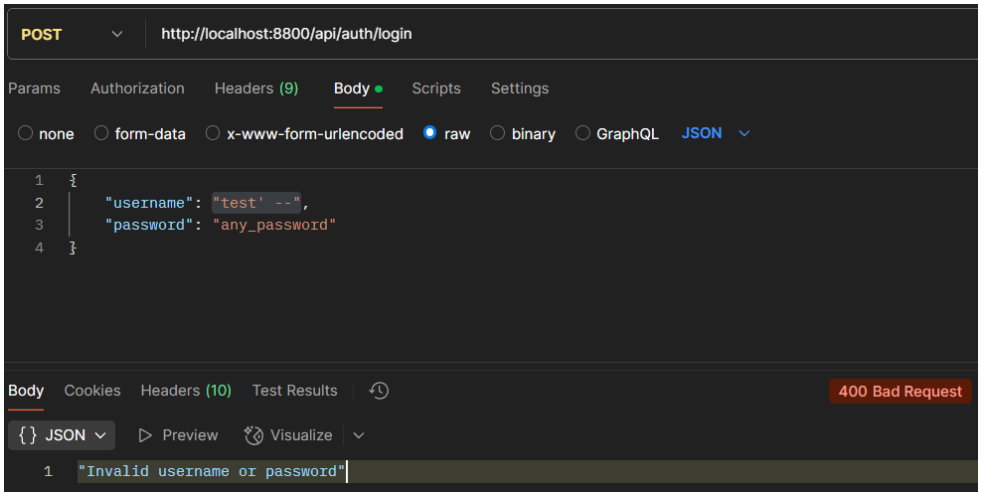


Рисунок 3.6 – Результат запиту

Ще спробуємо ін’єкцію виявлення помилок, для цього в поле username введемо значення "’", якщо додаток вразливий і помилки бази даних виводяться користувачеві, ви можете побачити повідомлення про синтаксичну помилку SQL, що підтвердить наявність вразливості.

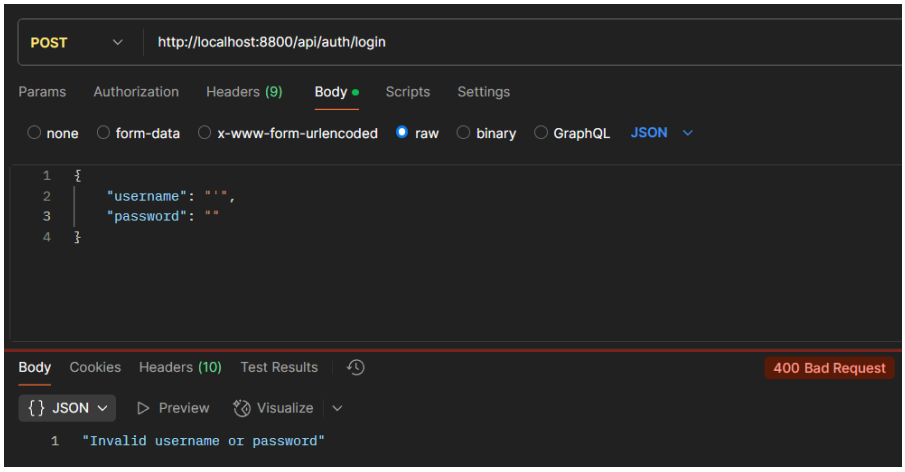


Рисунок 3.7 – Результат запиту

В підсумку тестування на SQL ін’єкції, можна зробити висновок що застосунок має високу стійкість даного до такого типу атак.

ВИСНОВКИ

У даній кваліфікаційній роботі було досліджено актуальну проблему захисту персональних даних в умовах обробки великих обсягів даних (Big Data). Проаналізовано предметну область Big Data, включаючи її основи, типи даних (структуровані, напівструктуровані, неструктуровані), технології та інструменти, а також сфери застосування. Визначено ключові виклики та ризики, пов'язані з безпекою Big Data, та обґрунтовано необхідність комплексного підходу до їх захисту.

В рамках практичної частини роботи було розроблено веб-застосунок, який виступає як джерело генерації Big Data. Для реалізації застосунку було обрано стек MERN (MySQL, Express, React, Node.js). Детально описано процес розробки клієнтської частини з реалізацією валідації форм для забезпечення безпеки введення даних. Також спроектовано та реалізовано структуру бази даних MySQL для зберігання різних типів даних, що генеруються застосунком. Розроблено серверний функціонал з використанням Node.js та Express, включаючи маршрути та контролери для обробки запитів, а також механізми автентифікації на основі JWT токенів та захисту від SQL ін'єкцій.

Важливою частиною роботи стала розробка системи обробки зібраної інформації. За допомогою мови програмування Python було створено скрипт для аналізу даних з бази даних, зокрема для виявлення популярних хештегів та підрахунку їх використання та асоційованих лайків. Реалізовано механізм автоматичного запуску скрипта за допомогою планувальника подій Windows.

Проведено тестування розробленого застосунку та системи обробки. Тестування клієнтської частини підтвердило ефективність реалізованої валідації форм. Тестування серверної частини, зокрема спроби SQL ін'єкцій, продемонструвало високу стійкість застосунку до даного типу атак завдяки використанню захищених механізмів формування запитів.

Для забезпечення ефективного захисту персональних даних при обробці Big Data необхідно ретельно налаштовувати права доступу, використовувати надійні методи автентифікації та шифрування даних. Важливо регулярно оновлювати програмне забезпечення, здійснювати моніторинг активності для виявлення підозрілих дій, а також мати план резервного копіювання та відновлення даних після аварій. Дотримання принципів мінімізації привілеїв та використання мережевих засобів захисту є ключовими елементами стратегії безпеки. Ефективний захист вимагає постійного аналізу ризиків та впровадження відповідних заходів безпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Big Data Defined: Examples and Benefits. Cloud Google. URL: https://cloud.google.com/learn/what-is-big-data (дата звернення: 29.02.2025).

2. Що таке Big Data? - Характеристики, приклади, методи обробки великих даних. URL: https://university.sigma.software/what-is-big-data (дата звернення: 05.03.2025).

3. Big Data, Explained: The 5V s of Data. Medium. URL: https://medium.com/@get_excelsior/big-data-explained-the-5v-s-of-data-ae80cbe8ded1 (дата звернення: 29.02.2025).

4. Big Data Open Source Tools and its Frameworks. Xenostack. URL: https://www.xenonstack.com/blog/big-data-tools (дата звернення: 04.03.2025).

5. Apache Hadoop Documentation. Apache Hadoop. URL: https://apache.github.io/hadoop (дата звернення: 04.03.2025).

6. Apache Hadoop YARN Overview. Apache Hadoop. URL: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html (дата звернення: 04.03.2025).

7. Apache Spark Documentation. Apache Spark. URL: https://spark.apache.org/docs/3.5.5 (дата звернення: 04.03.2025).

8. Amazon Web Services About. AWS. URL: https://aws.amazon.com (дата звернення: 04.03.2025).

9. The Essential Guide to Big Data. KDNuggets. URL: https://www.kdnuggets.com/2016/01/essential-guide-big-data.html (дата звернення: 05.03.2025).

10. Understanding Big Data: Concepts, Technologies, and Applications. Springer. URL: https://www.springer.com/gp/book/9783662498967 (дата звернення: 05.03.2025).

11. Ризики, пов'язані із захистом персональних даних в контексті Big Data. «Юридична газета» – професійне юридичне видання. URL: https://yur-

gazeta.com/publications/practice/inshe/riziki-povyazani-iz-zahistom-personalnih-danih-v-konteksti-big-data.html1 (дата звернення: 08.03.2025).

12. Data protection laws. DLA Papers. URL: https://www.dlapiperdataprotection.com/?t=law (дата звернення: 08.03.2025).

13. Identity and Access. Management Security. Engineer Introduction to Security Engineering. Medium. URL: https://iritt.medium.com/identity-and-access-management-security-engineer-introduction-to-security-engineering-744a9df66f87 (дата звернення: 09.03.2025).

14. Data Security in the Age of Big Data. Forbes. URL: https://www.forbes.com/sites/forbestechcouncil/2021/08/17/data-security-in-the-age-of-big-data/?sh=73943486395b (дата звернення: 09.03.2025).

15. Network Security in 2025: Threats, Security Models and Technologies. Faddom. URL: https://faddom.com/network-security-in-2025-threats-security-models-and-technologies. (дата звернення: 09.03.2025).

16. Що таке аналіз соціальних мереж (Social network analysis, SNA). TheTransmitted. URL: https://thetransmitted.com/adlucem/shho-take-analiz-soczialnyh-merezh-social-network-analysis-sna (дата звернення: 10.03.2025).

17. How to build a social network website from scratch. CleverRoad. URL: https://www.cleveroad.com/blog/how-to-create-a-social-media-website (дата звернення: 10.03.2025).

18. What Is the MERN Stack? Guide & Examples. Oracle. URL: https://www.oracle.com/ua/database/mernstack (дата звернення: 13.03.2025).

19. Налаштування Node.js проекту з нуля. Digitalocean. URL: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-project-with-npm (дата звернення: 05.03.2025).

20. How to Build a RESTful API Using Node, Express, and MongoDB. Freecode camp. URL: https://www.freecodecamp.org/news/build-a-restful-api-using-node-express-and-mongodb (дата звернення: 13.03.2025).

21. Досвід контейнеризації Node.js застосунків з Docker. DevZone. URL: https://devzone.org.ua/post/dosvid-konteyneryzatsiyi-nodejs-zastosunkiv-z-docker (дата звернення: 05.03.2025).

22. Головні дані: види, характеристики та переваги. ITPedia. URL: https://uk.itpedia.nl/2017/08/29/big-data-soorten-kenmerken-en-voordelen (дата звернення: 09.03.2025).

23. Big Data методи, технології та інструменти. r_d media. URL: https://robotdreams.cc/uk/blog/577-shcho-take-big-data (дата звернення: 09.03.2025).

24. Технології і тенденції роботи з Big Data. IT Enterprise. URL: https://www.it.ua/knowledge-base/technology-innovation/big-data-bolshie-dannye. (дата звернення: 09.03.2025).

25. Big Data в маркетингу: що це таке і як може допомогти. Kyivstar Business Hub. URL: https://hub.kyivstar.ua/articles/big-data-v-marketingu-shho-cze-take-i-yak-mozhe-dopomogti (дата звернення: 20.03.2025).

26. Технологія big data: сутність, можливості для бізнесу. Economics MSU. URL: https://economics-msu.com.ua/web/uploads/pdf/Scientific%20Bulletin%20of%20MSU.%20Series%20Economics_2019_Issue_2(12)_51-56.pdf (дата звернення: 23.03.2025)

27. Big Data в охороні здоров'я: аналіз масивних даних для покращення медичних послуг. Med Expert. URL: https://med-expert.com.ua/news-uk/big-data-v-ohoroni-zdorovya-analiz-masivnih-danih-dlya-pokrashennya-medichnih-poslug (дата звернення: 23.03.2025)

28. Big Data в промисловості: інновації, до яких доведеться звикати. OGS. URL: https://www.ogcs.com.ua/uk/big-data-v-promislovosti-innovatsiyi-do-yakih-dovedetsya-zvikati/ (дата звернення: 25.03.2025)

29. Великі дані в транспорті та логістиці. SOLIX. URL: https://www.solix.com/uk/products/answers/big-data-in-transportation-and-logistics/ (дата звернення: 25.03.2025)

30. Особливості взаємозалежності технологій Big Data та Інтернету речей. Вісник ПДТУ. URL: https://journals.uran.ua/vestnikpgtu_tech/article/view/288096 (дата звернення: 25.03.2025)

31. Що таке шифрування даних?. PAYPRO. URL: https://payproglobal.com/ (дата звернення: 29.03.2025)

32. Псевдонімізація. VPN Unlimited. URL: https://www.vpn-unlimited.com/ua/help/cybersecurity/pseudonymization?srsltid=AfmBOoqNnh8Gmq_FFEYl14x9rSKr4OK9ppQxTcfhV17tznVqgs6txY-H (дата звернення: 29.03.2025)

33. Правильна валідація форми React. FoxmindED. URL: https://foxminded.ua/validatsiia-formy-react. (дата звернення: 10.04.2025)

34. MySQL Documentation. MySQL. URL: https://dev.mysql.com/doc (дата звернення: 12.04.2025)

35. How to Design Database for Social Media Platform. Geeks For Geeks. URL: https://www.geeksforgeeks.org/how-to-design-database-for-social-media-platform/ (дата звернення: 12.04.2025)

36. Routing in NodeJS. Tutorial and use exaples. Geeks For Geeks. URL: https://www.geeksforgeeks.org/routing-in-node-js (дата звернення: 13.04.2025)

37. NodeJS - MySQL2 documentation with examples. NPM. URL: https://www.npmjs.com/package/mysql2#documentation (дата звернення: 13.04.2025).

38. JSON Web Token (JWT). Geeks For Geeks. URL: https://www.geeksforgeeks.org/json-web-token-jwt/?adsafe_ip/ (дата звернення: 14.04.2025).

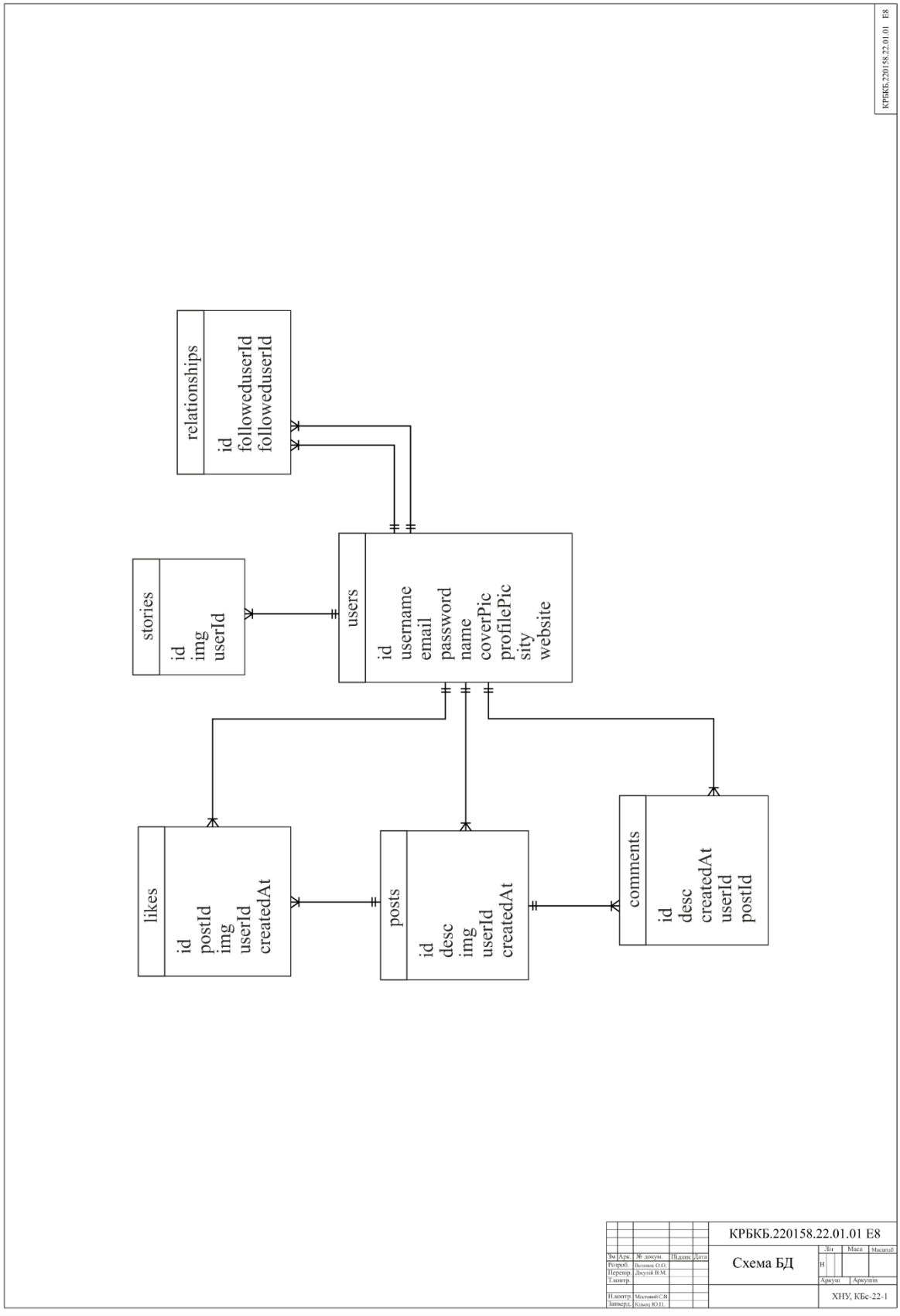
39. MySQL Administrator roles managment guide. MySQL. URL: https://downloads.mysql.com/docs/administrator-uk.pdf (дата звернення: 15.04.2025).

40. Mysqldump — A Database Backup Program. MySQL. URL: https://dev.mysql.com/doc/refman/8.4/en/mysqldump.html (дата звернення: 15.04.2025).

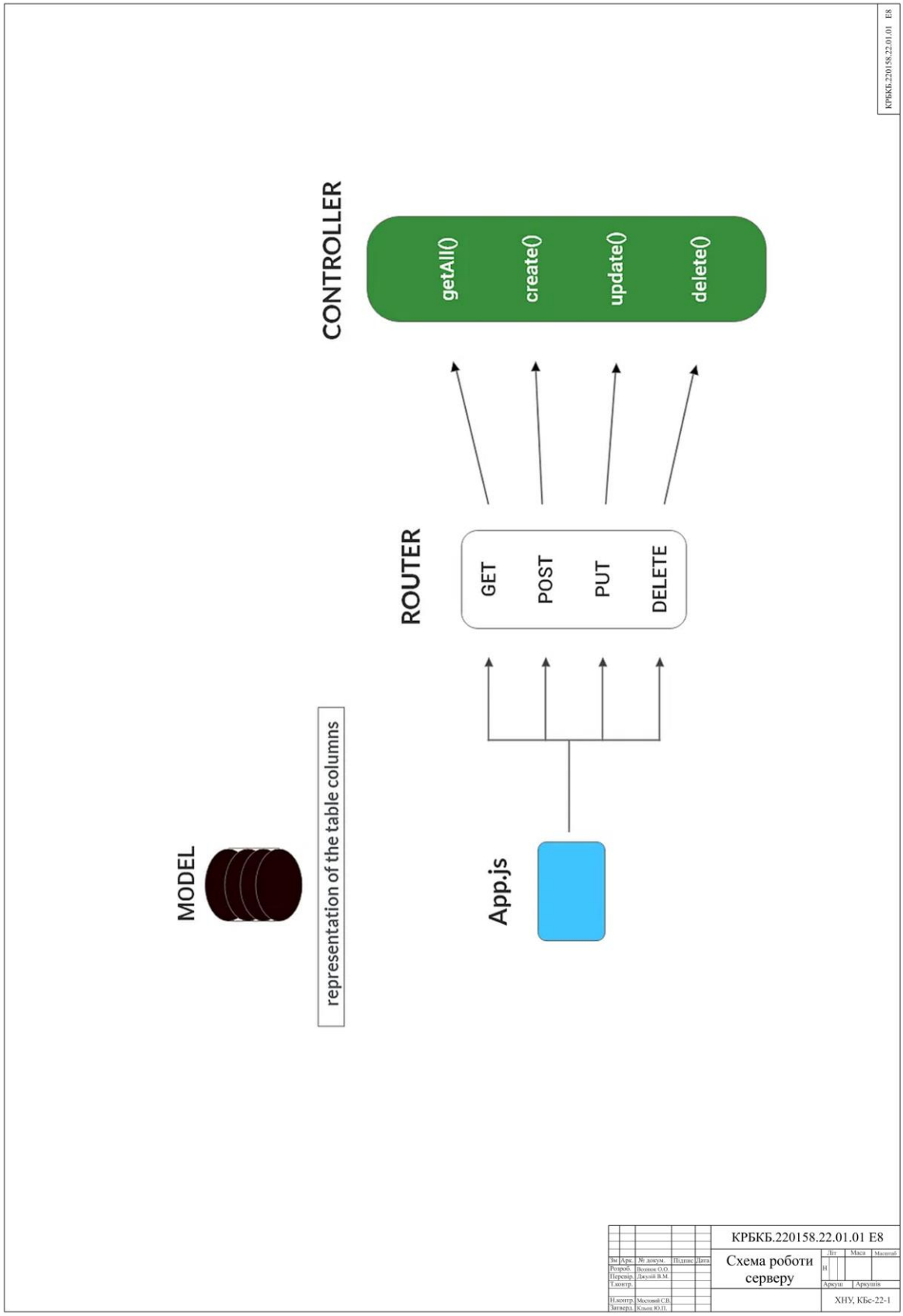
41. What is Structured Data?. Geeks for geeks. URL: https://www.geeksforgeeks.org/what-is-structured-data (дата звернення: 05.03.2025).

ДОДАТОК А

Копії графічної частини



Shog
User
React
Frontend
Frontend
Node.js/Express.js
API Backend
Backend
User
MySQL
Database
Big Data Source
App
Frontend
Database
Scheduled Python Script
Reads data from
MysQL Database
Data Analysis
CSV Report File
КРБКБ.220158.22.01.01 Е8
Схема роботи
застосунку
ХНУ, КБс-22-1


CONTROLLER
getAll()
create()
update()
delete()
ROUTER
GET
POST
PUT
DELETE
App.js
MODEL
representation of the table columns
КРБКБ.220158.22.01.01 Е8
Схема роботи серверу
ХНУ, КБс-22-1

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Вознюка Олега Олеговича
ПІБ здобувача вищої освіти
Студент ФІТ, 3 курсу, групи КБс-22-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

28.05.2025
дата

підпис

Anti-Plagiarism v-15.274 Educational

The maximum coincidence with one document 1.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 12%**

	Document		Sum coincidence on the DB	
ID: 242334 Title: Система захисту персональних даних при обробці Big Data Added in a DB: 2025-05-28 Authors: Вознюк Олег Олегович Heads: Джулій В.М. Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	76785	1150	579 (1%)	8 (1%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Вознюк Олег Олегович

Співавтор:

Назва: Система захисту персональних даних при обробці Big Data

Науковий керівник:

Підрозділ: Кафедра кібербезпеки

Коефіцієнт подібності 1:0.7%

Коефіцієнт подібності 2:0.2%

Мікропробіли: 0

Заміна букв: 1

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-05-29 03:39:50.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

30.05.2025р.

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система захисту персональних даних при обробці Big Data

Автор: Вознюк Олег Олегович

Спеціальність: 125 – Кібербезпека та захист інформації

Освітня програма:Кібербезпека та захист інформації

Науковий керівник: Володимир ДЖУЛІЙ, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою StrikePlagiarism складає 99.3%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism складає 99%.

Згідно з правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 24.09.2024, авторська робота, обсяг оригінального тексту у відсотках до загального обсягу матеріалу в якій складає 90-100%, визначається роботою з високою унікальністю тексту і допускається до захисту.

Керівник роботи Володимир ДЖУЛІЙ

Гарант ОП Віктор ЧЕШУН

Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Вознюк Олег Олегович

Тема Система захисту персональних даних при обробці Big Data

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 2; кількість сторінок записки 67.

1. Короткий зміст роботи та прийнятих рішень. У кваліфікаційній роботі розглянуто задачу створення системи захисту персональних даних при обробці великих даних (Big Data). Основна мета полягала у розробці програмного продукту, що генерує Big Data, та впровадженні системи безпеки для обробки цих даних. Прийняті рішення включають: розробку веб-застосунку – соціальної мережі "Shog" як джерела генерації структурованих, неструктурованих та напівструктурованих великих даних, технологічний стек, що складається з MySQL, Express.js, React, Node.js, проектування та реалізацію бази даних MySQL для зберігання інформації користувачів, публікацій, коментарів та інших взаємодій, розробку серверної частини (API) на Node.js з використанням Express.js, включаючи маршрутизацію та контролери для обробки запитів, впровадження механізмів безпеки, таких як автентифікація на основі JWT токенів, хешування паролів (bcrypt), та захист від SQL-ін'єкцій через параметризовані запити.

2. Висновок про відповідність кваліфікаційної роботи завданню. Кваліфікаційна робота повністю відповідає поставленому завданню. Завдання полягало у розробці системи захисту персональних даних під час обробки в застосунку "Shog". Студентом було успішно розроблено сам застосунок, що генерує дані, впроваджено систему їх обробки та ключові механізми захисту на рівні застосунку та бази даних. Теоретичні аспекти Big Data та їх захисту були проаналізовані та застосовані на практиці.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Робота структурована логічно та послідовно. У першому розділі було проведено детальний аналіз предметної області. Розглянуто основи Big Data, їх характеристики, типи даних, сучасні технології та інструменти для роботи з ними (Hadoop, Spark, NoSQL, Python, Tableau), сфери застосування та особливості захисту Big Data. У другому розділі описано практичну реалізацію проекту. Здійснено вибір технологічного стеку та детально описано розробку клієнтської частини на React, включаючи валідацію форм, та серверної частини на Node.js з Express.js. Проектування бази даних MySQL та реалізацію контролерів для взаємодії з нею, включаючи заходи безпеки, а саме хешування паролів, JWT, захист від SQL-ін'єкцій. Розроблено систему обробки інформації за допомогою Python для аналізу даних та налаштовано резервне копіювання БД. У третьому розділі наведено результати тестування розробленої системи. Проведено тестування клієнтської частини, зокрема валідації форм реєстрації та логіну. Виконано тестування серверної частини на стійкість до SQL-ін'єкцій, яке підтвердило ефективність впроваджених заходів захисту. Студент використав актуальні технології та підходи, такі як React для динамічного інтерфейсу, Node.js для асинхронної обробки запитів, JWT для безпечної автентифікації, bcrypt для хешування паролів та Python з відповідними бібліотеками для аналізу даних.

4. Позитивні сторони роботи Позитивні сторони роботи полягають у її беззаперечній актуальності, оскільки вона торкається важливої проблеми захисту персональних даних в умовах стрімкого зростання обсягів Big Data. Робота вирізняється комплексним підходом до вирішення поставленої задачі, охоплюючи весь цикл від створення джерела даних у

вигляді веб-застосунку до розробки системи їх подальшої обробки, аналізу та захисту. Вагомою перевагою є практична спрямованість дослідження, результатом якої став функціонуючий прототип системи. Студент продемонстрував вміння використовувати сучасний стек технологій, включаючи MySQL, Express, React, Node.js для розробки застосунку та Python для аналізу даних. Було реалізовано важливі механізми безпеки, такі як автентифікація за допомогою JWT токенів, хешування паролів, захист від SQL-ін'єкцій, розмежування прав доступу до бази даних та налаштування резервного копіювання. Крім того, у роботі детально описано процес розробки та тестування створеного застосунку.

5. Негативні сторони роботи Для створення повноцінної промислової системи захисту Big Data спектр необхідних загроз та відповідних контрзаходів є значно ширшим. Наприклад, це могло б включати більш глибоке впровадження методів анонімізації для різних типів даних, розробку системи розширеного логування подій безпеки з їх аналізом у реальному часі, а також детальніший розгляд аспектів захисту на рівні мережевої інфраструктури. Також, розроблена система аналізу даних на Python, хоча й ефективно виконує поставлене завдання з аналізу хештегів, має потенціал для розширення.

6. Оцінка графічного оформлення та пояснювальної записки роботи. Пояснювальна записка обсягом 67 сторінок структурована логічно, матеріал викладено послідовно та зрозуміло. Робота містить 47 рисунків (схеми, скріншоти коду, інтерфейсу, результатів тестування), які добре ілюструють ключові аспекти розробки та функціонування системи. Графічний матеріал є доречним та сприяє розумінню змісту роботи. Оформлення роботи в цілому відповідає встановленим вимогам.

7. Відгук про роботу в цілому. Кваліфікаційна робота Вознюка О.О. на тему "Створення системи захисту персональних даних при обробці Big Data" є завершеним науково-практичним дослідженням. Студент продемонстрував глибоке розуміння проблематики захисту Big Data, навички проектування та розробки веб-застосунків з використанням сучасних технологій, а також здатність реалізовувати системи обробки та аналізу даних. Робота має чітку структуру, логічний виклад матеріалу та практичну цінність. Всі поставлені завдання виконані.

8. Інші зауваження У переліку використаних джерел відсутні посилання на Вікіпедію. Наявні посилання на ресурси для розробників та документацію, наприклад, GeeksForGeeks, Medium, офіційні сайти технологій, що є прийнятним для робіт з комп'ютерної інженерії та кібербезпеки при описі технологій та інструментів. Однак, для поглиблення теоретичної бази варто було б більше спиратися на наукові публікації та монографії, хоча для бакалаврського рівня обсяг та якість джерел є задовільними.

9. Оцінка кваліфікаційної роботи Ураховуючи всі позитивні сторони представленої кваліфікаційної роботи, її актуальність, комплексність вирішення поставленої задачі, практичну реалізацію та належне оформлення, можна зробити висновок, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Бойко Юлій Миколайович,
доктор технічних наук, професор кафедри ТМІТ

« _____ » _____ 2025. _____ (підпис)