Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації»

(тема роботи)

Виконав: студент 2-ого курсу групи 1АКІТ-22м
(шифр групи)
спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології
(шифр та назва спеціальності)
Олександр ДУСАНЮК
(Ім'я, ПРІЗВИЩЕ студента)

Керівник: к.т.н., доцент каф. АІІТ
Володимир КОЦЮБИНСЬКИЙ
(науковий ступінь, вчене звання / посада, Ім'я, ПРІЗВИЩЕ керівника)
«4» грудня 2023 р.

Опонент: к.т.н., професор каф. КСУ
Микола БИКОВ
(науковий ступінь, вчене звання / посада, Ім'я, ПРІЗВИЩЕ опонента)
«7» грудня 2023 р.

Допущено до захисту
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
(науковий ступінь, вчене звання)
«11» грудня 2023 р.

Вінниця ВНТУ – 2023 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти ІІ-ий(магістерський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма – Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«20» 09 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Дусанюку Олександру Сергійовичу
(ПІБ автора повністю)

1. Тема роботи: Розробка мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації.
Керівник роботи: к.т.н., доцент каф. АІІТ Володимир КОЦЮБИНСЬКИЙ
Затверджені наказом ВНТУ від «18» 09 2023 року № 247.
2. Строк подання роботи студентом: до «05» 12 2023 року.
3. Вихідні дані до роботи: серверна частина системи соціальної мережі, що базується на основі геолокації. Вимоги до техніки: ОС Ubuntu 20.04, Процесор Intel Core i5 3 GHz або вище, Об'єм оперативної пам'яті 8 Gb або більше, Жорсткий диск 120 Gb SATA2 або вище.
4. Зміст текстової частини: Вступ. Аналіз предметної області та огляд існуючих аналогів та систем. Вибір та огляд технологій для розробки даної системи. Вибір архітектури системи. Розробка структури бази даних. Розробка та тестування розробленої системи. Економічний розділ. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного (або графічного) матеріалу: UML Deployment діаграма. UML Data Flow діаграма системи. ER-моделі бази даних. UML діаграма класів. UML Data Flow діаграми API. Результати тестування системи.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Володимир КОЦЮБИНСЬКИЙ, к.т.н., доцент каф. АІІТ	21.09.2023	04.12.2023
4	Володимир КОЗЛОВСЬКИЙ, к.е.н., професор каф. ЕПтаВМ	21.09.2023	06.12.2023

7. Дата видачі завдання: «21» 09 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір, узгодження та затвердження теми МКР	21.09.2023 - 23.09.2023	Виконано
2	Актуальність систем соціальних мереж, що базуються на основі геолокації	24.09.2023 - 27.09.2023	Виконано
3	Порівняльна характеристика існуючих систем та аналогів	28.09.2023 - 30.09.2023	Виконано
4	Огляд та вибір технологій для розробки системи	01.10.2023 - 04.10.2023	Виконано
5	Вибір та розробка архітектури системи	05.10.2023 - 10.10.2023	Виконано
6	Розробка структури бази даних	11.10.2023 - 16.10.2023	Виконано
7	Розробка серверної частини системи	17.10.2023 - 06.11.2023	Виконано
8	Тестування розробленої системи соціальної мережі	07.11.2023 - 12.11.2023	Виконано
9	Підготовка економічного розділу	13.11.2023 - 15.11.2023	Виконано
10	Оформлення матеріалів до захисту МКР	16.11.2023 - 20.11.2023	Виконано
11	Попередній захист МКР	21.11.2023 - 21.11.2023	Виконано
12	Остаточний захист МКР	12.12.2023 - 14.12.2023	Виконано

Студент _____ (підпис) Олександр ДУСАНЮК (Ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) Володимир КОЦЮБИНСЬКИЙ (Ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

УДК 004.4

Дусанюк О.С. Розробка мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації. Магістерська кваліфікаційна робота зі спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології, освітньо-професійна програма – Інтелектуальні комп'ютерні системи. Вінниця: ВНТУ, 2023. 126 с.

На укр. мові. Бібліогр.: 57 назв; рис.: 35; табл.: 8.

Метою роботи є створення серверної частини мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації. Це дасть змогу покращити комунікацію між користувачами та підвищити достовірність інформації, яка публікується.

В даній роботі розглянуто основні аспекти розробки клієнт-серверної системи соціальної мережі, що базується на основі геолокації, а саме серверної частини. Досліджено реалізацію даної системи та існуючі аналоги. Проаналізовано технології, за допомогою яких була розроблена серверна частина системи, та підхід, щодо реалізації архітектури системи. Розробку серверної частини виконано з використанням мови програмування Python, фреймворку Flask, бази даних MySQL та архітектурного стилю REST. Проведено тестування розробленої системи. В результаті роботи було запропоновано алгоритмічне та програмне забезпечення розробки серверної частини системи.

Ключові слова: соціальна мережа, що використовує принципи геолокації, серверна частина, Python, Flask, REST API, патерн MVC, тестування програмного забезпечення, сервер баз даних.

ABSTRACT

Dusaniuk O.S. Development of a mobile client-server social network system based on geolocation. Master's thesis in specialty 151 – Automation and computer-integrated technologies, educational and professional program – Intelligent computer systems. Vinnytsia: VNTU, 2023. 126 p.

In Ukrainian language. Bibliography: 57 titles; fig.: 35; tabl.: 8.

The purpose of the work is to create the server part of the mobile client-server system of the social network based on geolocation. This will make it possible to improve communication between users and increase the reliability of the information that is published.

In this work, the main aspects of the development of a client-server system of a social network based on geolocation, namely the server part, are considered. The implementation of this system and existing analogues have been studied. The technologies with which the server part of the system was developed and the approach to the implementation of the system architecture were analyzed. The server part was developed using the Python programming language, the Flask framework, the MySQL database, and the REST architectural style. The developed system was tested. As a result of the work, algorithmic and software development for the server part of the system was proposed.

Keywords: social network using the principles of geolocation, server part, Python, Flask, REST API, MVC pattern, software testing, server database.

ЗМІСТ

ВСТУП

Актуальність роботи. По всьому світу люди використовують соціальні мережі, щоб бути в курсі останніх подій, залишатися на зв’язку та отримувати актуальні новини. Ці платформи надають засоби для особистої і групової комунікації, пошуку інформації, перегляду новин і багато іншого. Саме через велику кількість інформації, що публікується постає питання в її достовірності. В епоху стрімкого обміну та поширення інформації це питання є питанням інформаційної безпеки особистості, суспільства і навіть держави [1].

Інформація може вважатися достовірною, якщо вона взята з першоджерела, а також, якщо є фото- та відео підтвердження. Але проблема фото- та відео матеріалів полягає у тому, що їх можна створити в одному місці, а пізніше написати, що даний матеріал створений зовсім в іншому місці, завдяки чому ввести в оману багатьох людей. Для того, щоб підтвердити достовірність фото та відео необхідно при їх створенні фіксувати геолокацію, де вони були опубліковані.

Саме тому актуальним є створення системи, яка буде не лише забезпечувати обмін інформацією та зв’язок з іншими користувачами, а також у якій при публікації фото, відео тощо, для даних матеріалів буде фіксуватися їх геолокація, і яка буде надавати користувачеві інформацію про події, що відбуваються навколо нього.

Метою магістерської кваліфікаційної роботи є покращення комунікації між користувачами та підвищення достовірності інформації, яка публікується, шляхом створення мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації.

Задачі досліджень магістерської кваліфікаційної роботи:

1. Огляд основних принципів функціонування соціальних мереж та геолокаційних соціальних мереж;

2. Вивчення існуючої реалізації геолокаційних соціальних мереж;

3. Аналіз технологій та підходів для створення серверної частини соціальної мережі;

4. Розробка алгоритмічного та програмного забезпечення серверної частини соціальної мережі;

5. Тестування розробленого програмного забезпечення.

Об'єктом дослідження є процес обміну інформації між користувачами за допомогою засобів соціальних мереж, що базуються на основі геолокації.

Предметом дослідження є структура моделі комунікації між користувачами засобами соціальної мережі, що базується на основі геолокації.

Методи дослідження. У роботі використано методи ідентифікації, збору і обробки даних, які отримуються від користувачів, та використання цих даних для покращення взаємодії між користувачами та підвищення достовірності інформації, яка публікується.

Новизна отриманих результатів дослідження полягає в тім, що на відміну від існуючих геолокаційних соціальних мереж, дана соціальна мережа дозволить: створювати публікації до яких буде прикріплюватися геолокація, що базується на поточному місцезнаходженні користувача; створювати публікації лише з того пристрою з якого було зроблене фото, зняте відео, записаний звук тощо; переглядати створені публікації на карті з відображенням геолокації; отримувати сповіщення від певної локації, коли там публікується пост, а також отримувати сповіщення, якщо пост публікується поруч із користувачем. Усі ці можливості дозволять покращити комунікацію між користувачами та підвищити достовірність опублікованої інформації.

Практична цінність отриманих результатів дослідження полягає в тім, що були удосконалені методи збирання та обробки даних, які отримуються від користувачів при використанні засобів геолокації.

Апробація та публікації матеріалів досліджень. Основні результати виконання магістерської кваліфікаційної роботи були опубліковані в матеріалах Всеукраїнської науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи»

(Вінниця, ВНТУ, 2022-2023 рр.) [2] та у матеріалах LII Науково-технічної конференції підрозділів Вінницького національного технічного університету (Вінниця, ВНТУ, 2023 р.) [3].

1 ЗАГАЛЬНІ ВІДОМОСТІ

1.1 Актуальність соціальних мереж та перспективи їх використання

У сучасному світі комунікація між людьми стала максимально простою та доступною. На сьогоднішній день будь-яка відстань між людьми не являється перешкодою для спілкування. Це стало можливо завдяки тотальній інформатизації нашого життя, зокрема появі Інтернету.

З початком свого існування соціальні мережі спочатку виконували комунікативно-інформаційну роль, проте з часом їх функції розширилися. Сьогодні, у будь-якій соціальній мережі, ви побачите не лише профіль користувача, а й рекламні повідомлення.

Соціальні мережі міцно увійшли в наше повсякденне життя та стали його невід'ємною частиною, а кількість користувачів по всьому світу вимірюється мільйонами. На стадії свого становлення соціальні мережі в основному використовувалися тільки для спілкування та розваг. Проте зараз ми можемо спостерігати, що соціальні мережі стають також і повноцінним інструментом для бізнесу. Це пов'язано, насамперед, із наявністю в соціальних мережах конкретної цільової аудиторії та можливістю активної взаємодії з реальними людьми. Таким чином, історія виникнення та еволюції соціальних мереж поступово стає одним із актуальних напрямків наукових досліджень і привертає увагу вчених.

Сучасні соціальні платформи пропонують своїм користувачам широкий спектр засобів для взаємодії одне з одним: відео, чати, зображення, музику, блоги, форуми та інше. Одночасно, для бізнесу, соціальні мережі – це новий канал спілкування з клієнтами і досить важливий інструмент дослідження їхніх уподобань.

Соціальні мережі перейшли від виключно розважальної ролі до статусу важливого інструменту у сфері інформаційних баталій, впливу на думку

громадськості та засобу впливу на свідомість електорату. Вони стали нескінченним джерелом даних для аналізу споживацької поведінки.

Тому, геолокація залишається одним з видів інформації, за яким можна виявити “больові точки” користувачів та знайти свій підхід подання інформації через групи спортклубів, клубів по зацікавленностях, закладів масового харчування, в яких були здійснені геолокаційні мітки тощо.

Соціальні мережі з використанням технології геолокації мають назву “геосоціальні мережі” [4-7].

Геосоціальна або геолокаційна соціальна мережа – це різновид соціальних мереж, що використовує геокодування та геомітки для розширення можливостей соціалізації. Користувачі діляться інформацією про своє місцезнаходження, що дозволяє координувати та об’єднувати їхні дії на основі присутності людей у конкретних місцях або подій, що там відбуваються. Геосоціальні мережі дозволяють користувачам не лише обмінюватись місцезнаходженням, а й створювати та взаємодіяти в контексті подій та спільних інтересів, формуючи нові форми взаємодії та спільнот. Для геолокації за допомогою комп’ютерів можуть використовуватися IP-адреси або трилатерація, а в мобільних пристроях – GPS, текстова інформація або LBS. [8].

У сучасному світі росте популярність геолокаційних соціальних мереж, наприклад, Foursquare. Ці сервіси, які вказують користувачам їх місцезнаходження, стають все більш важливими для компаній у проведенні маркетингових та PR-кампаній.

Завдяки можливостям геолокації бренди можуть легше спрямовувати свої зусилля на потенційних клієнтів у конкретний час та місце, впливаючи більш ефективно на їхні споживчі уподобання. У Foursquare стандартними елементами взаємодії з аудиторією є програми лояльності, огляди, купони, рекомендації, знижки та інші можливості.

Отже, можна визначити, що соціальні медіа виступають ключовим маркетинговим інструментом у галузі організації подій, і тому професіонали

повинні володіти навичками використання цими інструментами для успішної PR-діяльності.

Різноманітний, цікавий і цільово спрямований контент виступає ключовою силою в соціальних мережах, і життєздатність спільноти визначається тим, наскільки інформація, розміщена в ній, відповідає інтересам цільової аудиторії. Таким чином, важливо публікувати анонси майбутніх подій, розкривати деталі, використовувати відео та фотографії, взаємодіяти з учасниками спільноти, відслідковувати та оперативно реагувати на негативні відгуки, вживати заходів для усунення недоліків, підтримувати позитивні коментарі та репости, обговорювати проблеми, збирати ідеї для компанійського розвитку і спілкуватися з відомими блогерами та ЗМІ [9].

Таким чином, поширення та популярність соціальних мереж кардинально змінили не лише спосіб спілкування, але й саму суть бізнесу. Робота в цих мережах стала ключовим інструментом у маркетинговій комунікації в сфері організації подій, і використання цього інструменту має великий вплив на загальний успіх та розвиток компанії.

1.2 Порівняльна характеристика існуючих систем та аналогів

В наші дні існує багато різних соціальних мереж з функцією геопозиціонування. Всі вони представляють різноманітний набір можливостей та функцій. Розглянемо деякі з них, а саме Foursquare, Instagram, Zenly.

1.2.1 Foursquare

Foursquare – це популярна соціальна мережа з функцією геопозиціонування. Простіше кажучи, з її допомогою користувач може відзначати своє поточне місцезнаходження на карті та дивитися, де знаходяться

друзі та знайомі. Мається на увазі не інформація про місце проживання, а додавання до свого профілю відвідуваних кафе, магазинів та визначних пам’яток.

Для того, щоб розпочати користуватися цією соціальною мережею можна зареєструватися на сайті або встановити мобільну версію на базі операційних систем Android та iOS.

Особливості використання. Наприклад, якщо користувачеві сподобався певний заклад, він може відзначити своє відвідування та написати коментар. Зробивши це, користувач може поділитися цією інформацією з іншими користувачами. Місце буде вказано на карті за допомогою датчика GPS. Друзі відразу помітять новину у профілі, та й інші зможуть читати залишену користувачем інформацію. Процес відмітки свого візиту називається “Check in”. Користувач, за власним бажанням, може прикріпити фотографії до своїх повідомлень, щоб інші користувачі могли отримати більше інформації про це місце за їх допомогою. Крім того, користувач за кожен коментар отримує спеціальний значок та п’ять балів у якості винагороди. Збираючи їх, користувач зможе у майбутньому отримувати знижки у конкретних закладах. Той, хто відвідує те чи інше місце найчастіше, отримує почесне звання мера. Якщо користувач буде використовувати програму правильно, він зможе отримати безліч різних нагород [10, 12-14].

Дана соціальна мережа пропонує такі можливості:

- додавання друзів та слідкування за їхніми публікаціями;
- відображення цікавих та популярних місць, які знаходяться поруч;
- рекомендація популярних місць та користувачів;
- пошук місця за назвою або типом;
- відображення інформації про користувача в його профілі: найбільш відвідувані місця; кількість набраних балів; коментарі, які користувач написав у відвідуваних місцях; місця, де користувач є мером;
- система бонусів, яка дозволяє отримувати знижки в конкретних закладах [11, 12].

1.2.2 Instagram

Instagram, створений Кевіном Сістромом і Майком Крігером у 2010 році, увійшов під власність компанії Facebook у 2012 році, що відзначилося початком активного розвитку цієї соціальної мережі.

Instagram – це безкоштовний додаток, який сполучає обмін фотографіями та відео з можливостями соціальної мережі. Дана платформа дозволяє створювати і редагувати фото та відео, застосовуючи фільтри до них, а також розповсюджувати їх через власний сервіс та інші соціальні платформи.

Початковий вигляд Instagram був значно простіший порівняно з тим, що існує на сьогоднішній день. Функціональні можливості цієї соціальної мережі значно розширилися і продовжують розвиватися [15].

Instagram має як Web-версію, так і мобільну версію на базі операційних систем Android та iOS.

Дана соціальна мережа пропонує такі можливості:

- публікація фото чи відео. Реагування на публікації (лайки, коментарі). Можливість додати геолокацію до публікації;

- збереження публікацій у “Favourite”;

- Instagram Stories дозволяють в окремій секції додатку обмінюватися фотографіями та короткими відеороликами, прикрашаючи їх різноманітними елементами, такими як наклейки, малюнки, текст або геотеги;

- обмін повідомленнями, посиланнями, фотографіями і відео з іншими користувачами. В мобільній версії також є можливість голосового чи відеозв’язку;

- можливість налаштувати профіль, як приватний, або використовувати бізнес-акаунт. Ця можливість відкриває нові горизонти для інтернет-магазинів, надаючи доступ до розширеної статистики про аудиторію та записи, сприяючи зручному спілкуванню з клієнтами, а також можливості просування публікацій прямо з додатку;

- для бізнес-акаунтів є можливість перегляду статистики. Вона дозволяє стежити за кількістю передплатників, лайків і коментарів. Користувач також має можливість переглянути статистику профілю, історій та відео, щоб дізнатися кількість переглядів;

- Exploration – розділ у якому знаходяться фотографії та відео, які сервіс пропонує, в залежності від уподобань і місця розташування користувача;

- Instagram Reels – можливість записувати та редагувати 15-30-секундні вертикальні відеоролики з ефектами та звуком;

- можливість підписуватися на інших користувачів та стежити за їхніми публікаціями;

- наявність списку нотифікацій, в якому відображаються різні типи сповіщень [15-17].

1.2.3 Zenly

Zenly – соцмережа, заснована на добровільному обміні місцем розташування, що дозволяє відстежувати переміщення друзів у реальному часі. Сервісу вже п'ять років: за цей час він пережив невдалий запуск, перероджения, першу хвилю популярності та продаж.

Zenly показує розташування на карті: необхідно додати людину в друзі, щоб стежити за тим, де вона знаходиться. Для обміну не потрібно постійно тримати програму відкритою: геолокація все одно оновлюється в реальному часі. Не обов’язково бути недалеко, щоб побачити друга: додані користувачі відображаються на карті всього світу.

Не можна просто підписатися на людину і стежити за її переміщеннями: користувач повинен прийняти та схвалити заявку, і лише після цього він з’являється на карті. З цього моменту можна побачити, де і з ким він проводить час.

Додаток також дозволяє згенерувати посилання, за яким за пересуваннями користувача зможуть у браузері спостерігати ті, у кого сервіс не встановлений.

Дана соціальна мережа пропонує такі можливості:

- слідкувати за друзями в режимі реального часу;
- слідкувати за рівнем заряду акумулятора на смартфонах друзів;
- на карті відзначаються місця, де бував користувач;
- обмін повідомленнями з друзями;
- підвищувати рейтинг за рахунок додавання нових друзів;
- залишати на карті емодзі [18-19].

1.3 Висновки до розділу

В даному розділі було розглянуто актуальність соціальних мереж та перспективи їх використання. Також було проведено порівняльну характеристику існуючих систем та аналогів, таких як Foursquare, Instagram та Zenly, розглянуті їхні можливості. На основі проведених досліджень, до системи, що розробляється були сформовані такі вимоги:

1) Можливість створення постів з текстовим та медіа-контентом. Створення заходів;

2) Можливість створення груп для об’єднання за спільними інтересами та ідеями;

3) Рекомендація публікацій та заходів з позначенням геолокації, базуючись на місцезнаходженні користувача.

2 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

2.1 Вибір мови програмування для реалізації серверної частини системи та її бізнес логіки

Найбільш популярними серед серверних мов програмування є: JavaScript, Python, та Java.

Python є динамічною, об’єктно-орієнтованою, та інтерпретованою мовою програмування з строгою динамічною типізацією. Вона була розроблена голландським програмістом Гвідо ван Россумом у 1990 році.

Python – універсальний інструмент програмування, який сприяє створенню читабельного коду. Використання Python може значно скоротити обсяг програми порівняно з аналогом, написаним на іншій мові.

Дана мова програмування є кросплатформенною, що означає можливість запуску програм на Python у різних операційних системах без будь-яких модифікацій [20].

Ще однією перевагою Python є його вбудована бібліотека, яка автоматично встановлюється разом з мовою і включає готові інструменти для взаємодії з операційною системою, базами даних, веб-сайтами, різними форматами даних та розробки графічного інтерфейсу програм і багато іншого.

Розроблені за допомогою мови програмування Python програми, можуть варіюватися від невеликих скриптів до більш складних систем.

Є кілька ключових відмінностей Python від інших мов програмування, і ось основні з них:

1. Управління пам’яттю – повністю автоматизоване, не потрібно вручну розподіляти або звільняти пам’ять;

2. Типи в Python пов’язані з об’єктами, а не змінними, дозволяючи присвоювати змінним значення будь-якого типу і масивам містити об’єкти різних типів, що відрізняється від традиційних мов;

3. Операції в Python виконуються на вищому рівні абстракції завдяки особливостям самої мови та розширеній стандартній бібліотеці кодів, яка постачається разом з нею [20, 21].

Переваги мови програмування Python:

- об’єктно-орієнтований підхід та високорівневість, що спрощує програмування та полегшує розуміння коду;

- маючи динамічну типізацію, ця мова зменшує обсяг коду, який потрібен для виконання завдання, що прискорює процес написання коду і полегшує синтаксис мови;

- маючи простий синтаксис, мова забезпечує легкий старт у програмуванні для новачків, що робить процес написання коду приємним та зручним;

- універсальна мова програмування, яку можна використовувати в широкому спектрі сфер, від освітніх проектів та розробки веб-додатків до автоматизації різноманітних завдань, розробки штучного інтелекту та Data Science;

- маючи високу модульність, Python дозволяє розробляти частини програмного забезпечення окремо, уникнувши при цьому ризику впливу на роботу інших компонентів;

- багато наявних бібліотек та модулів у Python надають широкі можливості для розробки у багатьох галузях завдяки розгалуженому інструментарію;

- мова програмування, яка може функціонувати на різних платформах завдяки інтерпретації та відповідним інтерпретаторам, що робить її універсальною та кросплатформенною;

- Python є мовою програмування з відкритим вихідним кодом, що дозволяє перевірити відсутність недоліків та шкідливих елементів і при цьому брати участь у її подальшому поліпшенні;

- існування єдиного стандарту для написання коду, а саме PEP, сприяє підтримці та зрозумілості коду, навіть у випадку, коли над ним працює інший розробник;

- існування обширної документації відіграє важливу роль у процесі освоєння Python та пошуку відповідей на різноманітні питання.

Один з потенційних недоліків Python полягає у швидкості виконання коду. У Python відсутня компіляція. Спочатку код перетворюється на внутрішній байт-код, який потім виконується інтерпретатором Python. Часто програми, написані на Python, працюють повільніше у порівнянні з мовами, такими як C.

Однак, у сучасних комп'ютерах настільки висока обчислювальна потужність, що для більшості випадків швидкість розробки має важливіше значення, ніж швидкість виконання, і програми на Python, як правило, створюються значно швидше.

Крім того, Python легко розширюється за допомогою модулів, написаних на C++ або C, які можуть використовуватися для виконання фрагментів програми, що потребують великої обчислювальної потужності.

Python застосовується у різних сферах: розробка ігор, веб-додатків, внутрішніх інструментів для різноманітних проектів, а також у вирішенні прикладних завдань та наукових дослідженнях [22, 23].

JavaScript – це прототипна, об'єктно-орієнтована, та динамічна мова програмування, яка часто використовується для створення скриптів на веб-сторінках. Вона дозволяє взаємодіяти з користувачем на клієнтській стороні, керувати браузером, асинхронно обмінюватися даними з сервером, а також змінювати зовнішній вигляд та структуру веб-сторінки.

Особливості:

1. Скриптова мова програмування: JavaScript найчастіше використовується для написання коду клієнта, який виконується безпосередньо в браузері користувача;

2. Динамічна типізація: Змінні JavaScript можуть мати різні типи даних, і їх тип може змінюватися під час виконання програми;

3. Інтерпретованість: JavaScript – це мова, що інтерпретується, що означає, що код виконується в реальному часі браузером без попередньої компіляції;

4. Складні типи даних: JavaScript підтримує складні типи даних, такі як об’єкти та масиви, що робить його придатним для роботи зі структурами даних різної складності;

5. Асинхронне програмування: JavaScript підтримує асинхронне виконання коду, що особливо корисно під час операцій введення-виведення (I/O);

6. Функціональне програмування: JavaScript підтримує функціональне програмування, дозволяючи використовувати функції як об’єкти першого класу та застосовувати функціональні концепції, такі як замикання та вищі порядки функцій.

Переваги мови програмування JavaScript:

- Універсальність: JavaScript може виконуватись у будь-якому сучасному браузері, що робить його стандартом для клієнтської веб-розробки;

- Інтерактивність: JavaScript дозволяє створювати динамічні та інтерактивні елементи на веб-сторінках, що покращує досвід користувача;

- Велика екосистема: Існує велика кількість бібліотек та фреймворків, таких як React, Angular та Vue.js, які спрощують розробку на JavaScript;

- Серверна розробка: JavaScript може використовуватися для серверної розробки за допомогою засобів, таких як Node.js.

Недоліки мови програмування JavaScript:

- Крос-браузерна сумісність: Різні браузери можуть інтерпретувати JavaScript по-різному, що потребує крос-браузерної підтримки;

- Безпека: JavaScript піддається деяким видам атак, таким як міжсайтовий скриптинг (XSS) та міжсайтова запитова підробка (CSRF);

- Асинхронність і колбеки: Асинхронне програмування з використанням колбеків може призвести до складно читанного і складно керованого коду у великих проектах;

- Інтерпретованість: Інтерпретованість JavaScript може зробити його менш ефективним порівняно з деякими мовами, що компілюються [24-27].

JavaScript має широке застосування в таких областях:

- створення інтерактивних сценаріїв для веб-сторінок;

- розробка односторінкових та прогресивних веб-додатків (React, AngularJS, Vue.js);

- використання на стороні сервера (Node.js);

- розробка стаціонарних та мобільних застосунків (React Native, Cordova);

- сценарії у прикладних програмах, таких як Adobe Creative Suite або Apache JMeter;

- використання у PDF-документах та інших областях [28].

Java – це універсальна, надійна, безпечна, високорівнева та об’єктно-орієнтована мова програмування, яка підходить для різних платформ.

Ключові особливості Java:

1. Жорстка типізація. У Java всі змінні мають бути оголошені із зазначенням їхнього типу. Це запобігає помилкам типізації, дає змогу писати зрозумілий код і знаходити баги на етапі компіляції, а не виконання;

2. Автоматичне керування пам’яттю. У мові програмування Java реалізовано автоматичне збирання сміття. Воно звільняє пам’ять, зайняту об’єктами, що більше не використовуються. Розробникам не потрібно робити це власноруч;

3. Об’єктно-орієнтованість. Java ґрунтується на концепції об’єктно-орієнтованого програмування. Це означає, що все в Java є об’єктом, який має свої властивості та методи. Об’єктно-орієнтований підхід дає можливість Java-розробникам створювати модульні, гнучкі та безпечні застосунки [29-31].

Переваги мови програмування Java:

- Java вважається безпечною мовою програмування, оскільки вона не оперує явними покажчиками, як, наприклад, в мовах C і C++, які дозволяють отримати доступ до місця розташування пам’яті. Java також використовує такі

концепції ООП, як інкапсуляція, абстракція, успадкування, що підвищує безпеку;

- Java слідує функції WORA (Write Once Run Anywhere). Це означає, що код Java може працювати безпосередньо на кількох платформах, тобто не потрібно компілювати його щоразу. Код на Java під час компіляції перетворюється у байт-код, який є платформо-незалежним і може виконуватися на різних платформах;

- Java є портативною мовою. Це тому, що Java не залежить від платформи, а також не потребує спеціального обладнання для роботи. Це робить Java буквально сумісною майже з усіма можливими пристроями;

- щоб досягти максимального використання ЦП, багатопотоковість є ключовим компонентом. Java – це мова програмування, яка підтримує багатопотоковість. За допомогою Java можна запускати більше одного потоку одночасно. Вони використовують спільну пам’ять для підвищення ефективності та продуктивності програми. Потоки виконуються незалежно один від одного;

- Java отримує регулярні оновлення для виправлення помилок. Це робить Java однією з найстабільніших мов програмування. Майже всі баги усуваються відразу через оновлення;

- Java має механізм для обміну даними та програмами між декількома комп’ютерами, тому це розподілена мова програмування. Це підвищує продуктивність і ефективність системи. Java також має підтримку RMI (Remote Method Invocation), яка забезпечує розподілену обробку в java. Java також може спільно використовувати об’єкти в розподіленому середовищі, оскільки підтримує програмування сокетів і технологію COBRA;

- Java ділить пам’ять на дві частини: heap area та stack area. JVM виділяє пам’ять з будь-якої з двох частин залежно від вимоги. Це допомагає ефективно керувати пам’яттю.

Недоліки мови програмування Java:

- порівняно з іншими мовами програмування, такими як C++, Java повільніше виконує деякі завдання. Це може бути проблемою для застосунків, що потребують високої продуктивності або малої затримки.

- мова Java вимагає більше пам’яті для виконання програм. Це пов’язано з механізмом автоматичного керування пам’яттю та через те,що вона працює поверх віртуальної машини Java.

Основні сфери використання мови програмування Java: веб застосунки, мобільна розробка, корпоративне ПЗ, ігри на Java, Big Data, інтернет речей, наукові та дослідницькі програми, AR і VR [32, 33].

Порівняльну характеристику вищенаведених серверних мов програмування подано в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика серверних мов програмування

	Python	JavaScript (Node.js)	Java
Швидкість	Швидко	Швидше	Найшвидше
Продуктивність	Висока	Низька	Висока
Масштабованість	Середня	Найвища	Висока
Простота	Дуже проста	Середня	Проста
Спільнота	Сильна	Сильна	Сильна
Вартість	Безкоштовно	Безкоштовно	Платно
Крос-функціональність	Висока	Висока	Висока

Для створення серверної частини даної системи підходить мова програмування Python, так як вона має високу продуктивність, є об’єктно-орієнтованою, має підтримку всіх необхідних типів даних, структур даних, підтримку реляційних (SQL) та нереляційних (NoSQL) баз даних, а також має велику кількість бібліотек та фреймворків, включаючи Flask та

Django, за допомогою яких можна створювати API, керувати базами даних, обробляти HTTP-запити тощо.

2.2 Вибір фреймворку для реалізації серверної частини системи та її бізнес логіки

На сьогоднішній день найбільш популярними веб фреймворками на Python для написання серверної частини та створення API є Flask та Django.

Flask – це мікрофреймворк для створення простого та швидкого проекту мовою програмування Python з можливістю масштабування до складних програм. Flask, як мікрофреймворк, не включає в себе спеціалізовані інструменти або бібліотеки для взаємодії з базою даних, обробки форм, адміністрування, чи інші компоненти з широкими можливостями, що доступні в більших фреймворках. Проте існують розширення, які додають через сторонні бібліотеки такі функції, як встановлення об’єктно-реляційних зв’язків, перевірка форм, керування процесом завантаження, підтримка різноманітних технологій аутентифікації та різноманітні засоби, які є стандартними для фреймворків.

Flask використовують веб-розробники Python. Мікрофреймворк підходить для новачків. Дозволяє швидко створити веб-додаток, використовуючи лише один файл Python. Flask може служити як для розробки тренувальних проектів або невеликих сайтів з простим бекендом, так і для створення API та складних електронно-комерційних проектів. Ядро фреймворку можна масштабувати під різні завдання. Розробник повинен сам вибрати бібліотеки та інструменти, які хоче використати. Для цього необхідно підключити пакети розширення [34, 35].

Фреймворк Flask використовує Jinja2 – додаток для обробки шаблонів – і Werkzeug – інструмент для роботи з WSGI (стандартом взаємодії між програмою Python, яка виконується на стороні сервера, і самим веб-сервером).

Для створення ізольованого середовища Python використовується модуль Virtualenv [36].

Flask містить такі особливості:

- спроектований як мікрофреймворк, що означає, що він надає основні інструменти для створення веб-додатків, але залишає розробнику свободу вибору та інтеграції додаткових бібліотек і компонентів за необхідності;

- володіє простим і зрозумілим синтаксисом, що робить його доступним навіть для розробників-початківців. Це дозволяє швидко створювати прототипи та веб-додатки;

- легко розширюється за допомогою різноманітних розширень та сторонніх бібліотек. Розробники можуть вибирати та інтегрувати тільки ті компоненти, які необхідні для їхнього проекту;

- оснований на двох потужних бібліотеках: Werkzeug для обробки HTTP-запитів та Jinja2 для шаблонізації HTML. Ці бібліотеки надають високу продуктивність та гнучкість;

- надає легкий та зручний спосіб визначення URL-маршрутів та їх обробки за допомогою декораторів та функцій;

- підтримує інтеграцію з різними системами управління базами даних, включаючи NoSQL;

- надає інструменти для керування сесіями та роботою з cookies, що корисно для аутентифікації та збереження стану користувача;

- має режим налагодження, який полегшує процес налагодження веб-додатків;

- дозволяє легко створювати RESTful API за допомогою вбудованих інструментів та розширень;

- підтримує тестування додатків із використанням сторонніх бібліотек та інструментів;

- має багату екосистему розширень, які полегшують інтеграцію функціональності, такої як аутентифікація, авторизація, робота з формами тощо;

- має активну спільноту розробників, що забезпечує постійне оновлення та розвиток фреймворку [37].

Django – це високорівневий фреймворк для розробки веб-додатків мовою програмування Python, що використовує шаблон проектування MVC. Він надає розробникам набір інструментів та структуру проекту, які спрощують створення потужних та масштабованих веб-додатків.

Ключові особливості Django:

- поставляється з багатьма готовими компонентами, які забезпечують функціональність, таку як аутентифікація, робота з базами даних, обробка форм, адміністративний інтерфейс та багато іншого. Розробники можуть використовувати ці компоненти та модулі при необхідності, що прискорює розробку;

- включає ORM (Object-Relational Mapping), що дозволяє працювати з базами даних, використовуючи об’єктно-орієнтований підхід. Це спрощує створення та маніпуляцію даними без необхідності написання SQL-запитів;

- надає готовий адміністративний інтерфейс, який дозволяє адміністраторам керувати даними та додатком без написання коду на клієнтській стороні. Це особливо корисно для управління контентом та адміністрування;

- пропонує потужну систему маршрутизації, яка дозволяє визначити, які дії виконуються для різних URL-запитів. Це спрощує створення чистих та зрозумілих URL-адрес;

- забезпечує вбудований захист від багатьох загроз, таких як SQL-ін’єкції, CSRF (міжсайтова підробка запитів), і XSS (міжсайтовий скриптинг);

- використовує мову шаблонів, яка спрощує створення динамічних HTML сторінок і повторне використання коду;

- існує велика спільнота розробників та безліч сторонніх розширень і бібліотек, що дозволяє розширити функціональність Django для різних завдань;

- підтримує різні СУБД, включаючи PostgreSQL, MySQL, SQLite тощо, та надає потужний ORM для роботи з ними;

- легко інтегрується з бібліотеками для створення RESTful API [38-40].

Фреймворк Flask підходить для реалізації даної системи так як має можливість для побудови REST API, модулі для аутентифікації, авторизації, комунікації з базою даних тощо. Також даний фреймворк надає основні інструменти для розробки серверних додатків, але при цьому залишає здатність для розширення, завдяки чому можна інтегрувати лише необхідні компоненти та бібліотеки, що сприяє створенню оптимізованих та легковажних додатків.

2.3 Вибір технологій збереження та обробки даних

2.3.1 Аналіз структур даних і реалізація реляційних структур даних

Для того, щоб обрати систему управління базами даних, необхідно проаналізувати, які дані будуть зберігатися та, які зв’язки будуть між цими даними.

Для збереження текстових даних будь-якої довжини необхідний символьний тип з підтримкою довільної довжини, числових даних (в тому числі координат) – числові типи з підтримкою цілих чисел, чисел з плаваючою та фіксованою точкою, також треба булевий тип та тип з підтримкою дати і часу.

Таблиці можуть мати зв’язки у форматі один-до-одного, один-до-багатьох або багато-до-багатьох.

Відношення один-до-одного виявляється, коли значенню поля однієї таблиці відповідає лише одне значення поля другої таблиці, і, навпаки, значенню поля другої таблиці відповідає лише одне значення поля першої.

Відношення один-до-багатьох виникає, коли одному значенню поля першої таблиці може відповідати кілька значень поля другої таблиці, та кожному значенню поля другої таблиці відповідає лише одне значення поля першої.

Відношення багато-до-багатьох виникає, коли кожному значенню поля першої таблиці відповідає багато значень поля другої таблиці, і кожному значенню поля другої таблиці відповідає багато значень поля першої.

Для зв’язку таблиць в базі даних даної системи будуть використовуватися наступні зв’язки: один-до-одного, один-до-багатьох та багато-до-багатьох.

2.3.2 Вибір технологій реалізації сервера баз даних

Найбільш поширеними та популярними реляційними системами управління базами даних є MySQL та PostgreSQL.

MySQL – це система управління базами даних, спрямована на підтримку реляційних баз даних, і вона є програмним забезпеченням з відкритим кодом, що підтримується корпорацією Oracle. Спочатку створена шведською компанією MYSQL AB, вона потім була придбана Sun Microsystems і, нарешті, стала частиною корпорації Oracle. Базуючись на відкритому коді, MySQL дозволяє користувачам змінювати вихідний код згідно з їхніми потребами. Це масштабована, швидка і надійна система управління базами даних, яка працює на різних платформах, включаючи Windows, Linux, Unix тощо, і може бути встановлена як на робочому столі, так і на сервері.

В мережевому середовищі сервер MySQL функціонує як окрема програма або може бути використаний у вигляді бібліотеки, яку можна інтегрувати з окремим додатком. Існує ряд корисних застосунків для ефективного адміністрування бази даних MySQL. На противагу цьому, клієнти mysql встановлюються на комп’ютерах у мережі, інструкції відправляються від клієнта mysql на сервер, де сервер діє згідно до них. Навіть при встановленні mysql на одній машині, він може пересилати бази даних в різні локації, і користувачі мають можливість отримати доступ до них через різноманітні клієнтські інтерфейси MySQL.

MySQL також підтримує реплікацію даних та розподіл таблиць, що дозволяє користувачам досягти вищої продуктивності та забезпечити більшу довговічність. Для зберігання та отримання доступу до даних можна використовувати різні двигуни зберігання, такі як NDB, InnoDB і інші. MySQL написаний на мовах програмування C та C++ і підтримується на численних платформах, включаючи Windows, Linux, Mac тощо.

MySQL може встановлювати з'єднання із сервером mysql, використовуючи різноманітні протоколи, такі як TCP/IP та інші. Крім того, він підтримує різноманітні інструменти для адміністрування клієнтів та утиліти, такі як MySQL Workbench, а також кілька програм командного рядка [41-43].

Переваги MySQL:

1. Масштабованість. MySQL гарантує високу масштабованість для керування і координації роботи з глибоко вбудованими додатками, навіть у великих обсягах з великою кількістю даних, використовуючи обмежений слід. Гнучкість є ключовою особливістю MySQL, дозволяючи електронним комерційним компаніям повністю адаптувати сервер баз даних до унікальних ситуацій, оскільки MySQL пропонує рішення з відкритим кодом.

2. Безпека. MySQL є однією з найбільш захищених систем управління базами даних у світі та використовується у провідних веб-застосунках, таких як Youtube, Twitter, Facebook, WordPress тощо. Остання версія MySQL гарантує безпеку та підтримку транзакційної обробки, що надає значні переваги бізнесу, особливо у сфері електронної комерції, де великий обсяг фінансових операцій є надзвичайно важливим.

3. Продуктивність. MySQL розроблено для обслуговування найвимогливіших програм та забезпечення адекватної швидкості одночасно. Ця система пропонує покращену продуктивність через унікальні кеші пам'яті та повнотекстові покажчики. Незалежно від того, чи це веб-сайт електронної комерції з масою щоденних запитів, чи будь-яка операційна система, MySQL пропонує унікальну систему збереження даних, що дозволяє адміністраторам

налаштовувати сервер без будь-яких обмежень, що забезпечує високу ефективність.

4. Повний контроль робочого процесу. MySQL пропонує комбіноване рішення, яке включає функції самокерування для автоматизації процесів розширення простору та управління базами даних. З даним рішенням мінімізований час налаштування, і його можна швидко впровадити на різних операційних системах, таких як Windows, Linux, Unix, Macintosh і т.д.

5. Відкритий код. MySQL, який має відкритий код, розв’язує різноманітні проблеми: технічне обслуговування, налаштування, швидкі оновлення та поліпшення користувацького досвіду. Його захищена обробка даних ефективно працює з великими обсягами інформації.

6. Транзакційна підтримка. MySQL визнаний як основний механізм транзакційних баз даних завдяки його здатності забезпечити повну цілісність даних. Цей інструмент пропонує безліч блокувань на рівні рядків, а також повну підтримку атомних та множинних версій транзакцій для ізольованої, послідовної та стійкої обробки інформації [43, 44].

MySQL підтримує великий набір вбудованих типів даних:

1) Числові типи:

- Цілі;
- Типи з плаваючою точкою;
- Типи з фіксованою точкою;

2) Текстові типи:

- Символьні типи довільної довжини;
- Двійкові типи (включаючи BLOB);
- ENUM та SET типи;

3) Типи “дата/час”:

- DATETIME;
- TIMESTAMP;

4) Спеціальні типи даних:

- Булевий тип;

- JSON;

5) Просторові типи даних:

- POINT, POLYGON тощо [45].

PostgreSQL є системою управління об’єктно-реляційними базами даних із відкритим вихідним кодом. Вона повністю сумісна з SQL і була створена з багатим набором функцій. Вона також має можливість для розширення, що робить її корисною для тих, кому потрібні корпоративні інструменти. Вона була спеціально розроблена для ефективності та може бути інтегрована майже в будь-яке програмне забезпечення.

PostgreSQL є об’єктно-орієнтованою, що дозволяє розширювати типи даних для створення власних типів, створювати власні функції і підтримує майже будь-яку базу даних.

PostgreSQL пропонує широкі можливості для розробників у створенні додатків, наділяє адміністраторів засобами забезпечення цілісності та надійності даних, і створення надійного середовища, а також допомагає керувати інформацією, незалежно від її розміру [46-48].

Ключові особливості PostgreSQL:

1) Робота із великими обсягами. У більшості менших СУБД існують обмеження щодо розміру та обсягу записів у базі даних, спрямованих на середні та менші проекти. У PostgreSQL обмежень немає. Обмеження відносяться до окремих записів. Розмір однієї таблиці обмежений 32 Тб, а розмір одного запису – 1,6 Тб. У кожному полі запису може бути не більше 1 Гб даних, а максимальна кількість полів залежить від типу і може варіюватися від 250 до 1600. Даних обмежень цілком достатньо для зберігання різноманітних даних у базі даних.

2) Підтримка складних запитів – це одна з сильних сторін PostgreSQL. Вона ефективно обробляє складні та комплексні запити, здійснюючи трудомісткі операції, такі як читання, запис та валідація одночасно. Хоча у випадку з операцією читання дана СУБД може бути менш ефективною порівняно з конкурентами, проте вона виявляється більш продуктивною в інших аспектах.

3) Написання функцій кількома мовами. У PostgreSQL є можливість створювати власні функції – це блоки коду, які виконують певні завдання. Ця функціональність, хоч і притаманна більшості СУБД, відрізняється в PostgreSQL більшою підтримкою мов програмування. Окрім стандартного SQL, тут доступно написання коду на Python, Java, C++ і C, PHP, Lua та Ruby. Важливо відзначити, що PostgreSQL підтримує двигун V8 для JavaScript, тому його можна використовувати разом з цією СУБД. За потреби можна розширити систему під інші мови програмування.

4) Одночасна модифікація бази даних. Важливою особливістю PostgreSQL є можливість одночасного доступу до бази даних з декількох пристроїв. У СУБД реалізовано клієнт-серверну архітектуру, коли база даних зберігається на сервері, а доступ до неї здійснюється з клієнтських комп'ютерів. Однією з можливих складнощів є ситуація, коли кілька людей одночасно модифікують базу даних і треба уникнути конфліктів.

5) Відповідність ACID. ACID – це набір принципів забезпечення цілісності даних. Абревіатура розшифровується як Atomicity, Consistency, Isolation, Durability – атомарність, узгодженість, ізольованість, довговічність. Якщо база даних відповідає цим принципам, вона веде себе максимально передбачувано та надійно. У ній низький ризик конфлікту чи непередбаченої поведінки системи. PostgreSQL дотримується вимог ACID завдяки технології MVCC (Multiversion Concurrency Control). Це робить систему надійною та безпечною у використанні, а дані – захищеними від можливих збоїв, помилок та втрат.

6) Можливість розширення. Розробник може написати для СУБД власні типи та їх перетворення, операції та функції, обмеження та індекси, власну процедурну мову для запитів. PostgreSQL можна модифікувати практично під будь-яке нестандартне завдання.

7) Висока потужність та широка функціональність. PostgreSQL – можливо, єдина безкоштовна СУБД з відкритим вихідним кодом, яка

розрахована на роботу з об’ємними та складними проектами. Вона потужна, продуктивна, здатна ефективно працювати з великими масивами даних.

8) Кросплатформність. Найчастіше PostgreSQL використовують на серверах із операційними системами сімейства Linux, але СУБД підтримує й інші ОС. Її можна встановити в системи на базі Windows, BSD, MacOS і Solaris. Крім того, PostgreSQL має автономний веб-сервер PostgREST, з яким можна працювати за допомогою REST API.

У порівнянні з деякими іншими СУБД, PostgreSQL може споживати більше ресурсів (включаючи оперативну пам’ять та процесорний час). Це особливо помітно під час роботи з великими обсягами даних та виконання складних запитів. Також за показниками продуктивності PostgreSQL повільніший, ніж MySQL [48, 49].

PostgreSQL підтримує наступні типи даних:

1) Цілі числа (smallint, integer, bigint);

2) Числа з плаваючою точкою (real, double precision);

3) Текстові типи (character(n), varchar(n), text);

4) Дата та Час (date, time, timestamp, interval);

5) Булевий тип (boolean);

6) Бінарні дані (bytea);

7) Масиви (integer[], text[]);

8) JSON та JSONB (json, jsonb);

9) Просторові типи даних (point, line, polygon);

10) IP-адреси та мережі (inet, cidr);

11) UUID;

12) Інші спеціалізовані типи (bit, money, macaddr, xml) [50].

В якості СУБД для реалізації даної системи підходить MySQL, так як вона має високу продуктивність та здатність масштабуватися, підтримку всіх необхідних типів даних, підтримується мовою програмування Python та фреймворком Flask, а також підтримує безліч операційних систем.

2.4 Вибір протоколу взаємодії між клієнтською та серверною частинами системи

REST (Representational State Transfer) – це архітектура, тобто принципи побудови розподілених систем гіпермедіа, включаючи універсальні методи обробки та передачі станів ресурсів по HTTP. Термін REST виник у 2000 році, вперше використаний Роєм Філдінгом, одним з розробників HTTP-протоколу. Системи, що слідують принципам REST, відомі як RESTful-системи.

У загальному розумінні, REST є доволі простим інтерфейсом для керування інформацією без використання допоміжних внутрішніх шарів. Кожен елемент інформації є чітко визначеним глобальним ідентифікатором, таким як URL, який, в свою чергу, має чітко встановлений формат [51].

Що дає REST підхід:

- Масштабованість взаємодії компонентів системи;
- Спільність інтерфейсів;
- Незалежне використання компонентів;
- Проміжні компоненти, що знижують затримку, що посилюють безпеку.

До переваг REST можна віднести:

- Відсутність додаткових проміжних шарів під час передачі даних означає, що інформація надсилається у своєму власному форматі, без перетворень. Іншими словами, немає обгортання даних у формат XML, як це відбувається в SOAP і XML-RPC, або використання AMF, як це робиться у Flash і т.д. Просто передаються самі дані;

- Кожна окрема інформаційна одиниця (ресурс) є чітко визначеною за URL. Це означає, що URL виступає як основний ключ для кожної частини даних. Формат даних, який міститься за цим посиланням, не має значення – це може бути HTML, jpeg, документ Microsoft Word чи будь-що інше;

- Управління інформацією ресурсу повністю ґрунтується на протоколі передачі даних, і найбільш розповсюдженим є протокол HTTP. Цей протокол використовує методи, такі як GET (отримати), POST (додати, змінити,

видалити), PUT (замінити, додати) і DELETE (видалити), щоб визначати операції з даними. Таким чином, операції CRUD (Create-Read-Update-Delete) можуть виконуватися як з використанням всіх чотирьох методів, так і лише з використанням GET та POST.

Щоб розподілена система вважалася сконструйованою за архітектурою REST (RESTful), необхідно, щоб вона задовольняла наступним критеріям:

1) Client-Server. Система має бути поділена на клієнтів та на сервери. Розподіл інтерфейсів вказує на те, що клієнти і збереження даних не зв’язані, що дозволяє зберігати дані всередині кожного сервера, поліпшуючи мобільність клієнтського коду. Сервери не залежать від інтерфейсу користувача чи стану, тому вони можуть бути більш простими та масштабованішими. Клієнти та сервери можуть бути замінені та розроблятися окремо, не змінюючи при цьому інтерфейс;

2) Stateless. Сервер не має зберігати жодної інформації про клієнтів. Запит повинен містити всю необхідну інформацію для виконання запиту та, за потреби, ідентифікації клієнта;

3) Cache. Кожна відповідь має вказувати, чи є вона закешованою чи ні, щоб уникнути використання клієнтами застарілих або неправильних даних у відповідь на наступні запити;

4) Uniform Interface. Єдиний інтерфейс встановлює зв’язок між серверами та клієнтами, спрощуючи та розділяючи архітектуру, що дозволяє кожній складовій розвиватися окремо;

5) Layered System. У REST допускається розділити систему на ієрархію шарів, але з умовою, що кожен компонент може бачити компоненти лише наступного шару. Наприклад, якщо користувач викликає службу PayPal, а він у свою чергу викликає службу Visa, користувач про виклик служби Visa нічого не повинен знати.

Сервери можуть тимчасово кастомізувати або доповнити можливості клієнта, передаючи йому логіку для виконання.

Архітектура REST сама по собі не обмежена конкретними протоколами чи технологіями, проте у сучасному Інтернеті створення RESTful API майже завжди означає використання HTTP та різних розповсюджених форматів для представлення даних, таких як JSON або, рідше, XML.

З точки зору RESTful-сервісу, операція (чи виклик сервісу) вважається ідемпотентною, якщо клієнти можуть виконувати один і той самий виклик повторно і отримувати на сервері однаковий результат. Іншими словами, багаторазове створення тотожних запитів має той же ефект, що й один запит. Зазначимо, що, хоч ідемпотентні операції мають однаковий результат на сервері, відповідь може відрізнятися (наприклад, стан ресурсу може змінитися між запитами).

Методи DELETE та PUT є ідемпотентними за своєю природою. Але є одне відмінне положення для методу DELETE. Суть у тому, що успішний DELETE-запит повертає статус 200 (OK) або 204 (No Content), але кожен наступний запит буде постійно повертати відповідь 404 (Not Found). Стан сервера залишається незмінним після кожного виклику DELETE, проте відповіді відрізняються.

Такі методи, як GET, HEAD, OPTIONS та TRACE є безпечними за визначенням. Це означає, що вони служать лише для отримання інформації і не мають на меті змінювати стан сервера. Вони не мають створювати ніяких значних побічних ефектів, за винятком таких як: кешування, логування, показ банерної реклами, або збільшення лічильника.

За визначенням, безпечні операції ідемпотентні, тому що вони призводять до того самого результату на сервері. Безпечні методи реалізовані як операції лише для читання. Проте, безпека не передбачає, що сервер має повертати однаковий результат кожен раз [52, 53].

Групи кодів станів:

1) 1xx (Information) – в цей клас виділені коди, що інформують про процес передачі;

2) 2xx (Success) – цей клас кодів показує, що запит від клієнта було отримано, зрозуміло сервером, він був прийнятий і успішно виконаний;

3) 3xx (Redirect) – ці коди показують клієнтові, що для успішного виконання операції потрібно виконати новий запит, зазвичай, за іншою URI;

4) 4xx (Client Error) – коди даного класу призначені для випадків, в яких клієнт робить невірні запити;

5) 5xx (Server Error) – коди відповідей цього класу відносяться до випадків, в яких сервер ідентифікує, що сталася помилка з його вини або він по якійсь причині не в змозі виконати запит [54].

2.5 Висновки до розділу

В даному розділі було розглянуто та обґрунтовано вибір мови програмування, фреймворку та серверу баз даних для розв’язання поставленої задачі. Проаналізовано можливість реалізації зв’язку між клієнтом та сервером за допомогою використання протоколу взаємодії REST API.

Розглянуто реалізацію системи із використанням наступних технологій: мови програмування Python, фреймворку Flask, серверу баз даних MySQL.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ

3.1 Вибір архітектури системи

Створення архітектури системи є одним з головних етапів розробки клієнт-серверних систем, оскільки правильний вибір архітектури системи забезпечить оптимальну швидкодію, крос-платформеність, дозволить швидко і легко масштабувати систему та додавати новий функціонал в залежності від потреб.

В даній системі для взаємодії між серверною та клієнтською частинами використовується підхід REST API. Його також називають RESTful. Архітектура REST API базується на HTTP (Hypertext Transfer Protocol), протоколі передачі гіпертексту, що є стандартним протоколом в Інтернеті, призначеним для передачі гіпертексту. В наші дні за допомогою HTTP можна надсилати будь-які інші типи даних.

Кожен об'єкт на сервері в HTTP має свою унікальну URL-адресу у строгому послідовному форматі. Наприклад, п'ятий пост зберігатиметься на сервері за адресою http://vivat.sprava.net/article/5.

REST API використовує чотири основні методи HTTP для маніпулювання об'єктами на серверах: GET для отримання даних або списку об'єктів, DELETE для видалення, POST для додавання або заміни даних, PUT для регулярного оновлення даних.

Дані запити також відомі як ідентифікатори CRUD: створення (create), читання (read), оновлення (update) та видалення (delete). Це основний набір операцій для роботи з даними.

Кожен HTTP-запит містить заголовок, за яким йде опис об'єкта на сервері – це представлення його поточного стану.

Deployment діаграму та Data Flow діаграму наведено на рисунках 3.1 та 3.2 відповідно.

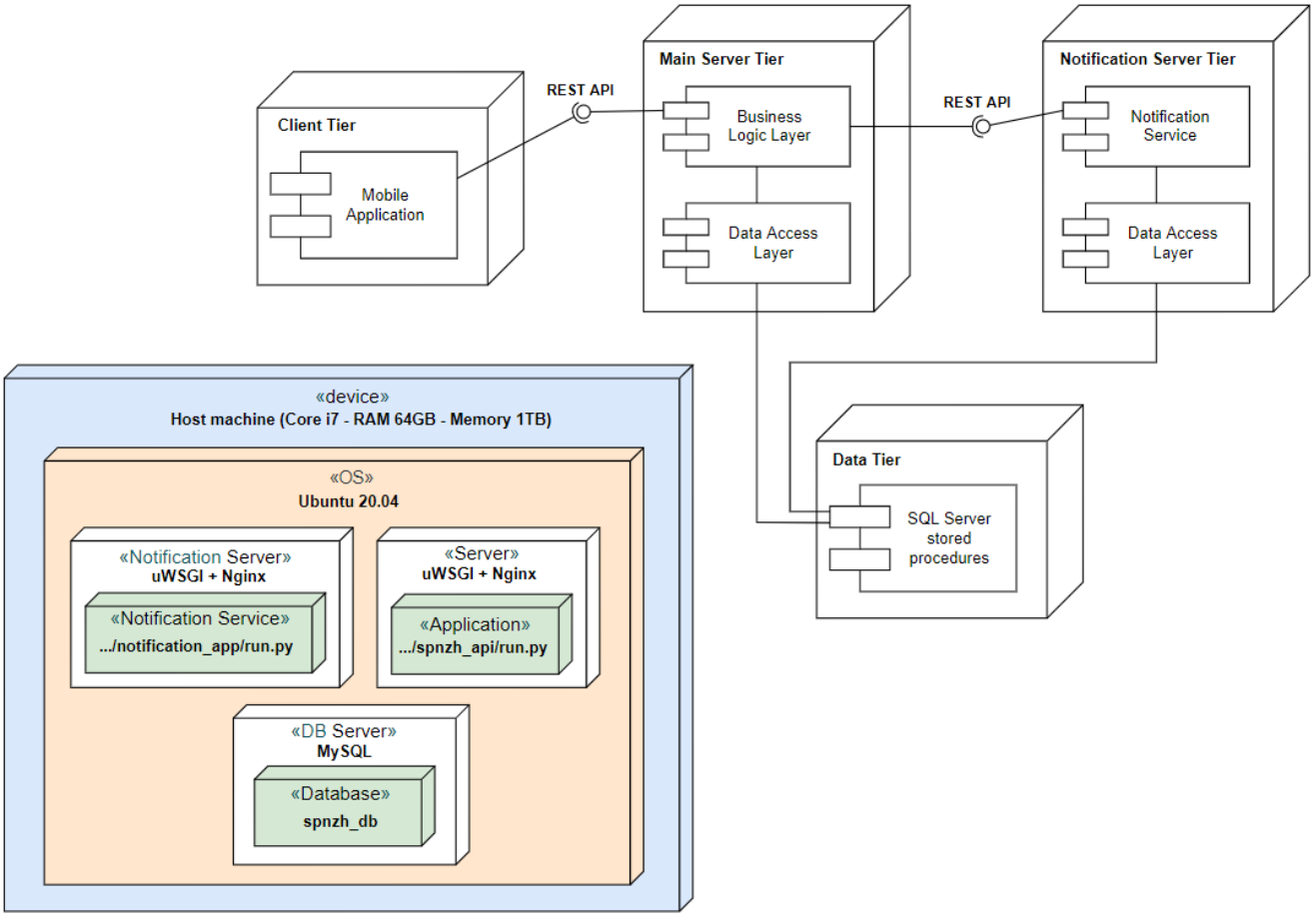

Рисунок 3.1 – Deployment діаграма

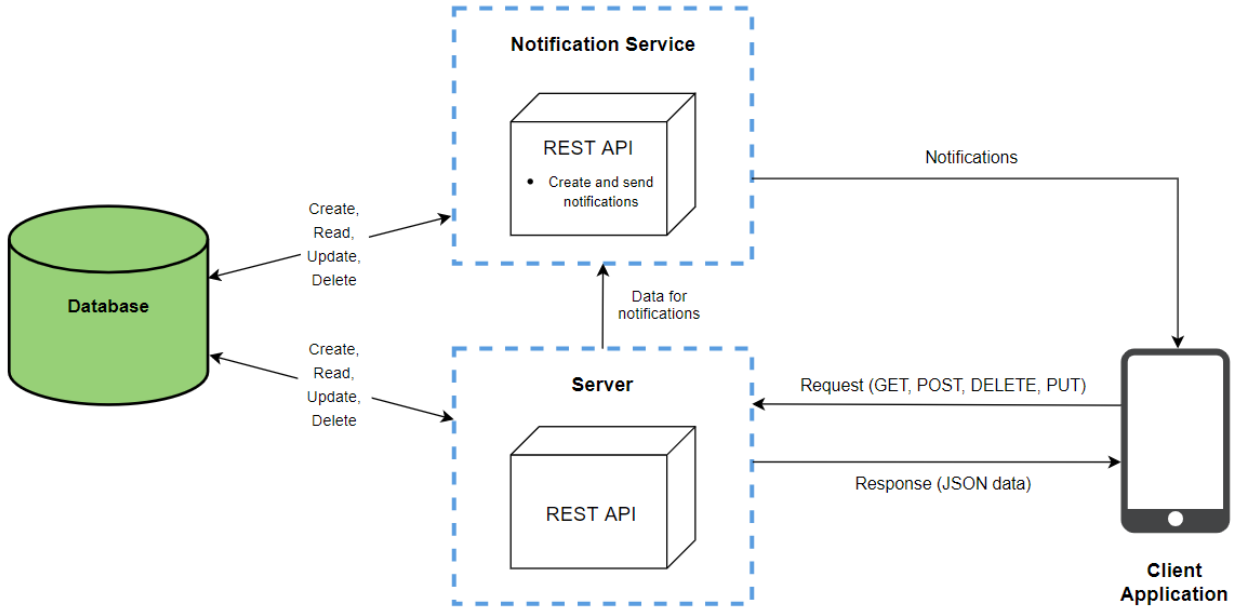

Рисунок 3.2 – Data Flow діаграма

Одними із основним процесів в системі є процес авторизації та аутентифікації. Вони необхідні для того, щоб користувач міг отримати доступ до функціоналу системи.

Так як для розробки даної системи було обрано фреймворк Flask, буде доцільно використовувати розширення Flask-Login.

Flask-Login забезпечує керування сеансами користувача для Flask: вхід (logging in), вихід (logging out) із системи та запам’ятовування сеансу користувача протягом тривалого періоду часу.

Flask-Login надає такі можливості:

1) Збереження ідентифікатора активного користувача під час сеансу, для того, щоб забезпечити легкий вхід (login) або вихід (logout) із системи;

2) Використання функції “Запам’ятати мене”;

3) Захист сеансів користувачів від крадіжки зловмисниками файлів cookie;

4) Можливість інтеграції з Flask-Principal або іншими розширеннями авторизації.

Принцип роботи аутентифікації та авторизації за допомогою Flask-Login зображено на рисунку 3.3.

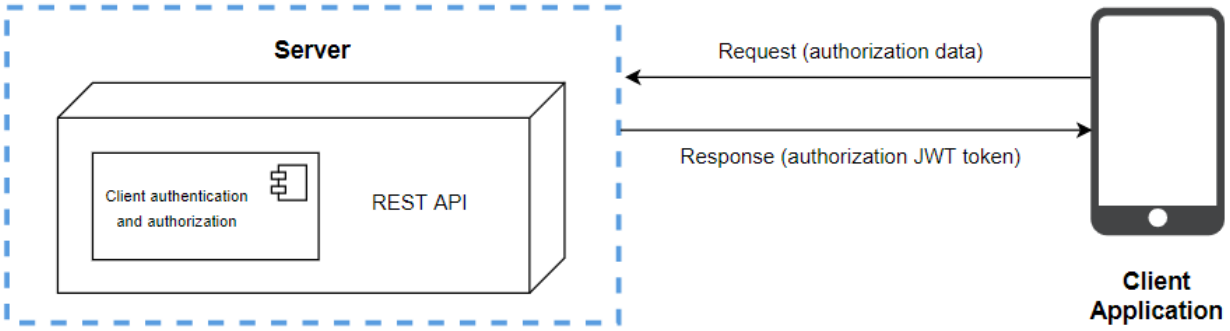

Рисунок 3.3 – Принцип роботи аутентифікації та авторизації

Щоб детальніше пояснити схему роботи системи, для прикладу можна взяти одну із її функцій, а саме створення поста.

Процес роботи системи при використанні даної функції:

1) Спочатку клієнт відправляє запит на аутентифікацію та, якщо запит успішний, отримує спеціальний JWT токен, за допомогою якого зможе авторизуватися при наступних запитах;

2) Потім користувач через клієнтський додаток, разом з токеном та необхідними параметрами, відправляє запит на сервер на створення поста;

3) Сервер створює пост, відправляє деякі дані про користувача та пост до сервісу нотифікацій, і повертає клієнту відповідь з даними поста у вигляді JSON;

4) Сервіс нотифікацій приймає дані, створює і розсилає нотифікації користувачам, які знаходяться поруч, або є друзями чи фоловерами даного користувача.

Для реалізації архітектури серверної частини ідеально підходить модель MVC. MVC (Model-View-Controller) – це архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення. Дана модель передбачає поділ системи на три взаємопов’язаних частини:

- Model – включає всі дані та пов’язану з ними логіку;

- View – відповідає за представлення даних користувачеві або обробку взаємодії з ним;

- Controller – являє собою інтерфейс між компонентами Model та View [55].

Шар Model (Модель) представляє об’єктну модель конкретної предметної області, містить дані та методи для їх обробки, відповідає на запити від контролера, надаючи та/або змінюючи дані. Важливо відзначити, що модель не визначає, як відображати дані або в якому форматі їх представляти, а також не взаємодіє з користувачем напряму.

Шар View (Представлення) – відповідає за відображення інформації (візуалізацію). Він отримує дані з рівня Model і відображає дані користувачеві, але він не взаємодіє безпосередньо з шаром Model, а отримує інформацію через контролер. Однакові дані можна відображати по-різному та відтворювати їх у різних форматах. Наприклад, набір об’єктів можна представити користувачеві у вигляді списку чи таблиці, а через API дані можна експортувати у JSON, XSLX, або XML.

Шар Controller (Контролер) – це інтерфейс між компонентами Model та View. Через шар View він отримує запити від користувача, обробляє їх, і надсилає на шар Model для обробки даних. Після обробки дані надсилаються на шар View, щоб їх можна було відобразити. Контролер містить безпосередньо всю бізнес логіку системи та оперує даними системи [56].

Мета шаблону полягає в створенні гнучкого дизайну програмного забезпечення, що спрямований на спрощення подальших модифікацій та розширення програм, а також він сприяє повторному використанню окремих їхніх складових. Великі системи, де використовується цей шаблон, мають більш структуровану архітектуру, завдяки чому вони стають менш складними та більш зрозумілими.

Опис використаної MVC архітектури представлено на рисунку 3.4.

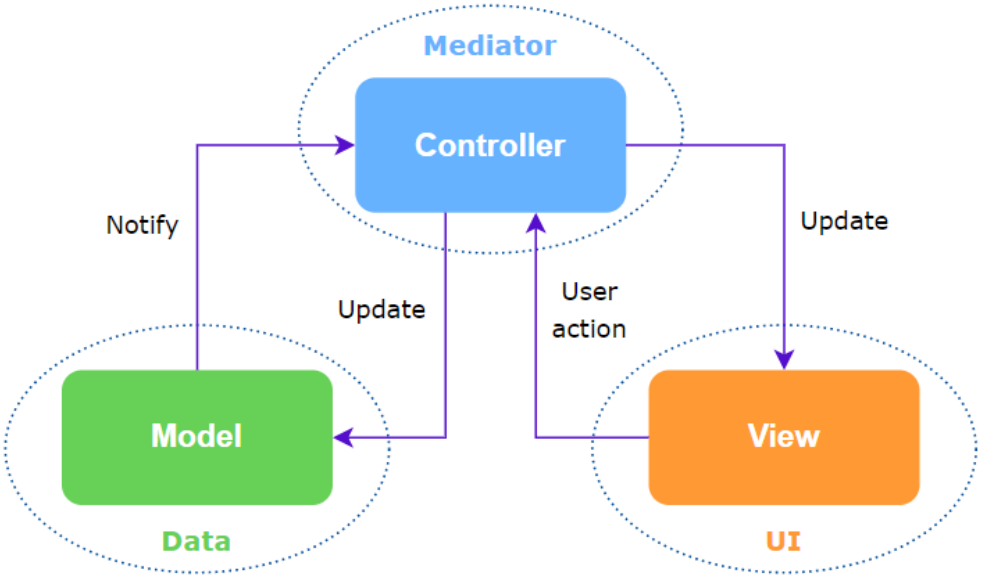

Рисунок 3.4 – Опис MVC архітектури

3.2 Розробка структури бази даних

Для роботи даної системи є необхідність збереження даних в надійному сховищі. Оскільки необхідно зберігати великі обсяги інформації, система повинна бути високо продуктивною та потужною. Потрібно зберігати інформацію про користувачів, пости, групи, локації, нотифікації тощо. Саме тому необхідно розробити ER (Entity-Relationship) модель бази даних, яка буде описувати всі ключові сутності та зв’язки між ними у вигляді схеми.

ER-модель, відома як модель “сутність-зв’язок”, є методом опису концептуальних схем через узагальнені конструкції блоків. Це мета-модель для опису даних, що використовується для створення моделей даних. Її основними елементами є сутності, зв’язки та атрибути.

Сутність – це віртуальний чи реальний об’єкт, який має ключове значення для аналізованої області та інформація про нього потребує зберігання.

Зв’язок – це узгоджене сполучення між двома чи кількома сутностями, що має важливе значення для предметної області, що аналізується. Зв’язки можуть мати наступні типи: багато до багатьох, багато до одного, один до багатьох, або один до одного.

Атрибут – це будь-яка властивість сутності, яка є суттєвою для розглянутої предметної області та використовується для ідентифікації, класифікації, кваліфікації, кількісної оцінки чи відображення стану сутності.

ER-модель часто впроваджується у вигляді баз даних. У випадку реляційної бази, де дані зберігаються у таблицях, кожен рядок цих таблиць відображає один конкретний екземпляр сутності. Окремі поля даних цих таблиць посилаються на індекси в інших таблицях, що вказує на фізичне втілення зв’язків між сутностями.

ER-модель складається з багатьох схем, але розглянемо основні з них, а саме схеми users, articles, media, groups, notifications, locations. В схемі users знаходяться основні таблиці в яких містяться дані про користувачів, їх налаштування, друзів та фоловерів, а саме таблиці: users, user_profile, user_devices, user_settings, user_token, users_followers, friends. На рисунку 3.5 зображена ER-модель схеми users.

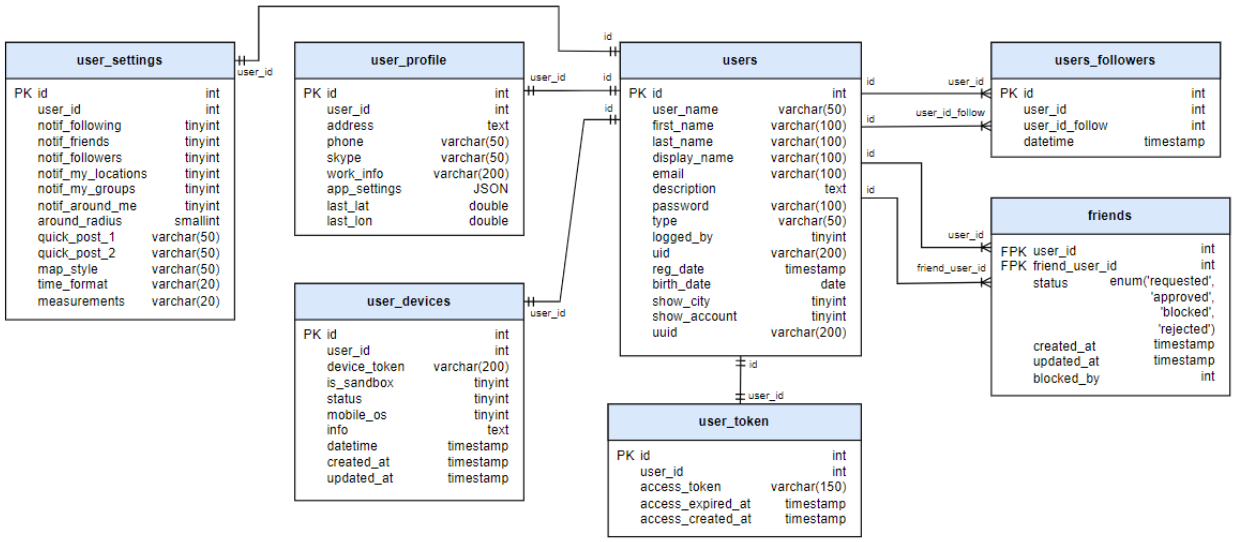

Рисунок 3.5 – ER-модель схеми users

В таблиці users знаходяться дані про користувачів, user_profile – додаткова інформація про користувачів, а також координати їх останнього місцезнаходження, user_devices – містить дані про пристрої користувачів, в тому числі токен пристрою, user_settings – дані про налаштування користувачів, user_token – інформація про JWT токен для авторизації. В users_followers міститься інформація про фоловерів, friends – інформація про друзів.

В схемі articles знаходяться таблиці в яких міститься інформація про пости, а саме таблиці: articles, articles_groups. На рисунку 3.6 зображена ER-модель схеми articles.

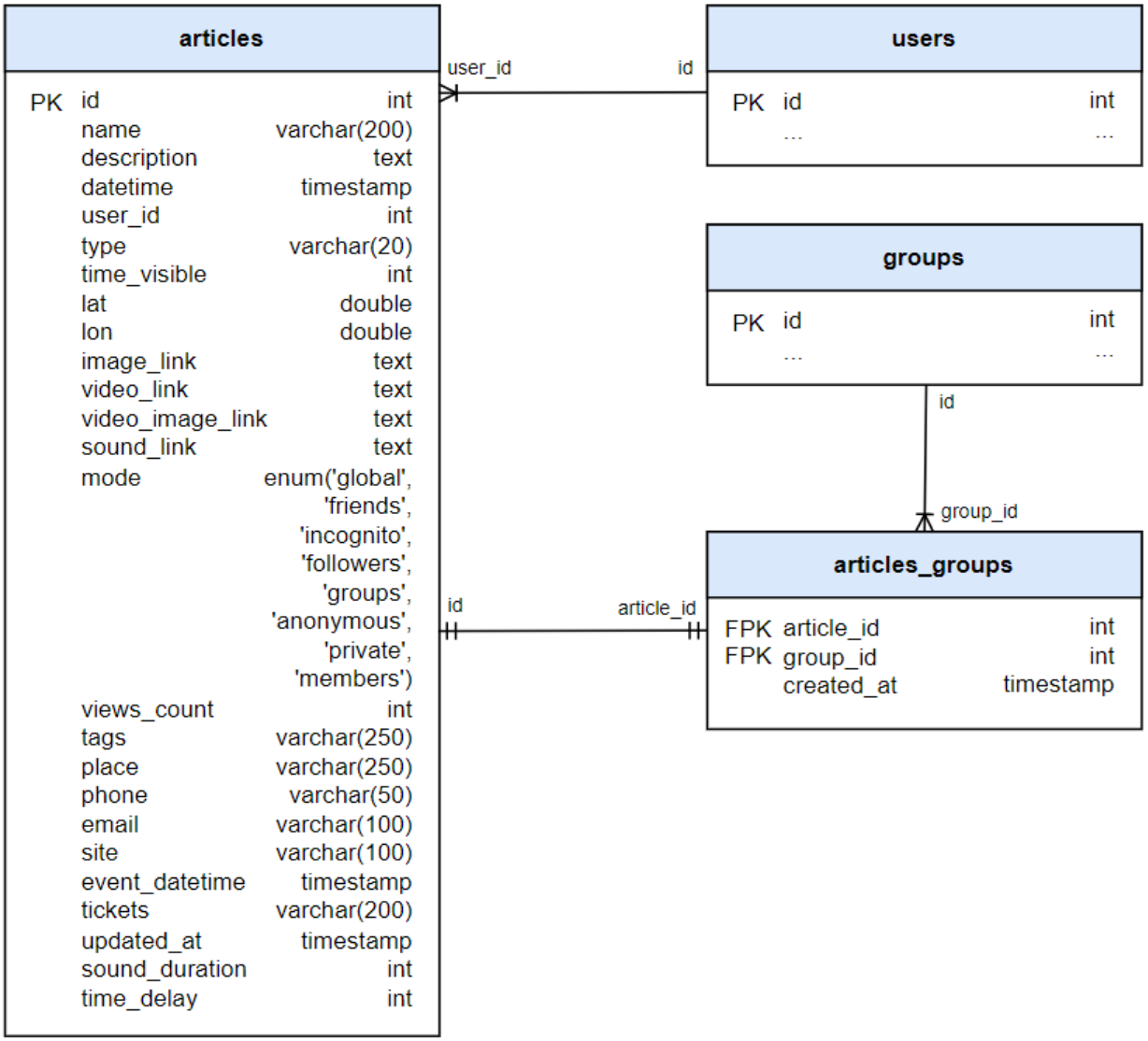

Рисунок 3.6 – ER-модель схеми articles

В таблиці articles знаходяться дані про пости, а в articles_groups – інформація про те, які пости опублікували групи.

Схема groups містить таблиці в яких знаходяться дані про групи, їх фоловерів та учасників, а саме таблиці: groups, users_groups, groups_followers. На рисунку 3.7 зображена ER-модель схеми groups.

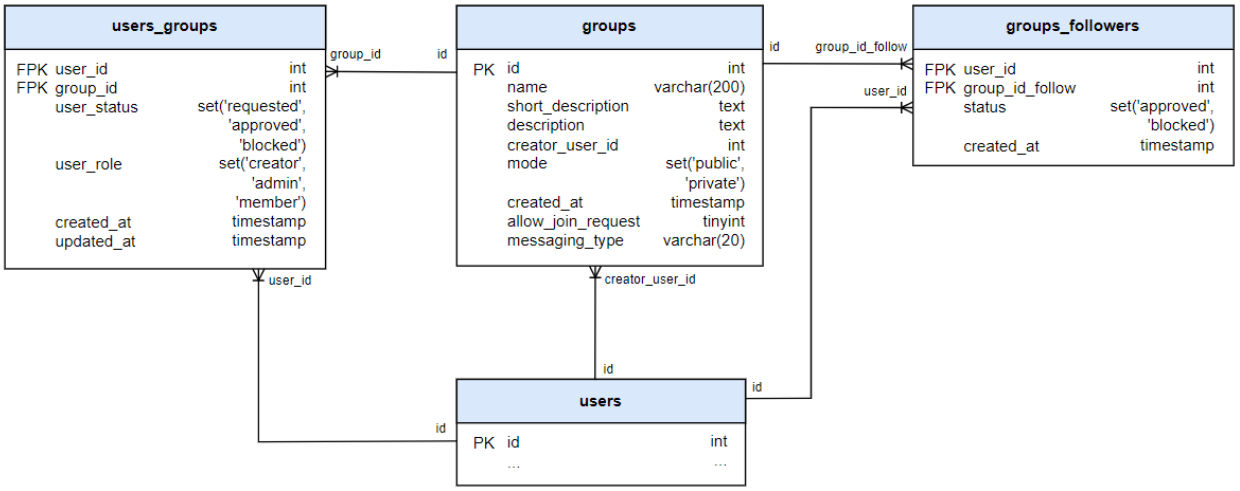

Рисунок 3.7 – ER-модель схеми groups

В таблиці groups знаходяться дані про групи, users_groups містить дані про учасників груп, groups_followers – інформацію про фоловерів груп.

В схемі media знаходяться таблиці, які містять інформацію про зображення, відео, аудіо, а саме таблиці: images, videos, sounds. На рисунку 3.8 зображена ER-модель схеми media.

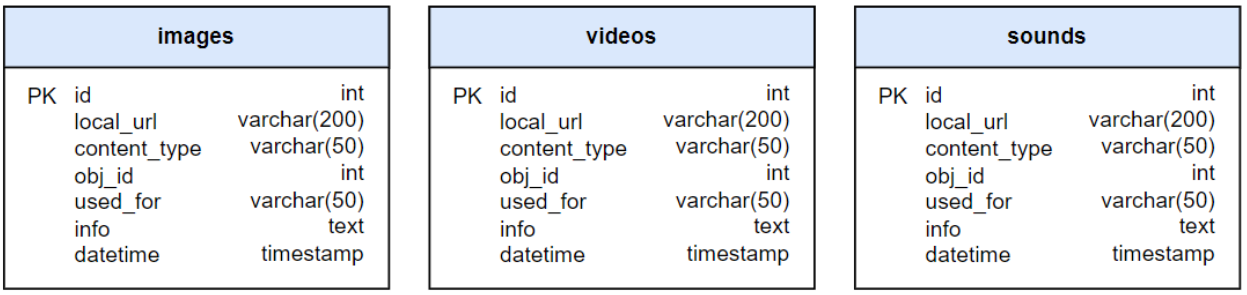

Рисунок 3.8 – ER-модель схеми media

В таблицях images, videos, sounds знаходиться інформація про зображення, відео, та аудіо відповідно, а також шлях до них та для чого вони використовуються.

Схема notifications містить таблиці в яких знаходяться дані про нотифікації та на які групи нотифікацій підписаний користувач, а саме таблиці: notifications, notifications_viewed, notifications_subscriptions, notifications_time_visit. На рисунку 3.9 зображена ER-модель схеми notifications.

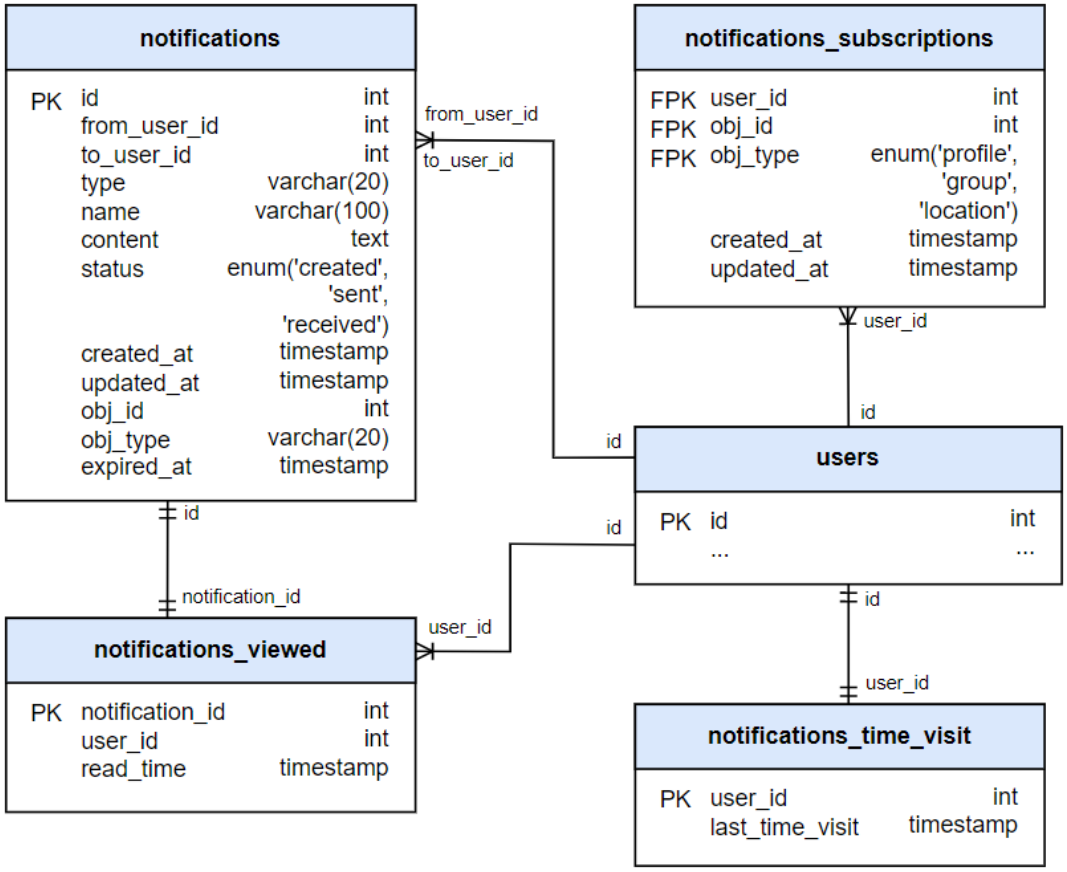

Рисунок 3.9 – ER-модель схеми notifications

В таблиці notifications міститься інформація про нотифікації, від кого та кому вони адресовані. Таблиця notifications_viewed призначена для збереження прочитаних нотифікацій. В notifications_subscriptions знаходяться дані про те, від кого користувач буде отримувати нотифікації. А в таблиці

notifications_time_visit зберігається інформація про те, коли користувач останній раз відкривав список з нотифікаціями.

Остання схема locations складається з однойменної таблиці в якій зберігається інформація про локації, які створив користувач. На рисунку 3.10 зображена ER-модель схеми locations.

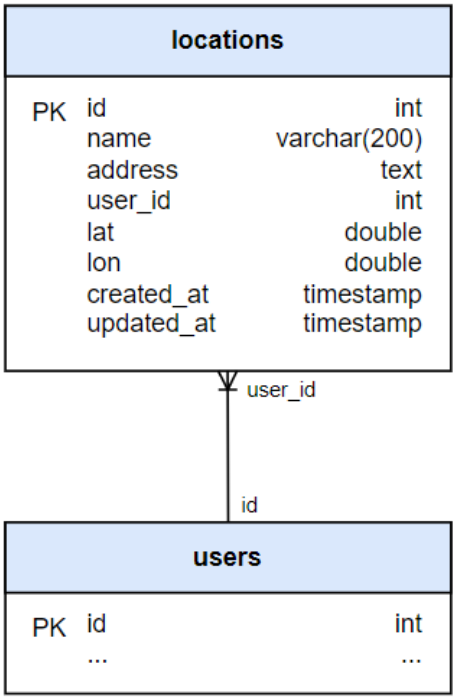

Рисунок 3.10 – ER-модель схеми locations

В таблиці locations містяться дані про локації, які створив користувач. А саме назва, адреса, координати тощо.

Для збереження даних була обрана база даних MySQL, так як вона відповідає усім вимогам та має підтримку всіх необхідних типів даних.

3.3 Розробка серверної частини

Для розробки серверної частини даної системи розробку необхідно розділити на дві частини, а саме на: розробку основної серверної частини, тобто сервера, який буде відповідати за всі основні бізнес процеси та розробку сервісу нотифікацій, який буде відповідати за створення і відправку нотифікацій клієнту.

Розробка основного сервера та сервісу нотифікацій буде проведена за допомогою фреймворку Flask. Даний фреймворк дозволяє розробити всі необхідні компоненти, дотримуючись архітектурного патерну MVC. Для цього серверну частину потрібно розбити на шари, а саме на: Model, View та Controller.

Model шар основного серверу буде містити в собі Python класи, які будуть відповідати таблицям в базі даних. Ці класи будуть задіяні для збереження даних і їх подальшого використання в системі. Шар моделей буде містити в собі такі класи як:

1) Articles, Articles_Groups – класи, які відповідають за збереження даних по постах;

2) User, Profile, UserDevice, UserSettings, UserToken, Friend, UserFollower – класи, які відповідають за збереження даних користувача, його налаштування, друзів і фоловерів;

3) Groups, UsersGroups, GroupFollower – класи, які відповідають за збереження даних по групам, їх учасників і фоловерів;

4) ImageStorage, VideoStorage, SoundStorage – класи, які відповідають за збереження даних по зображеннях, відео та аудіо;

5) Notification, NotificationSubscription, NotificationsTimeVisit, NotificationViewed – класи, які використовуються для збереження даних по нотифікаціям;

6) Location – клас, який відповідає за збереження даних по локаціям.

Лістинг моделей основного серверу наведено в додатку В.

Controller шар основного серверу буде містити в собі Python класи, в яких буде розташовуватися вся необхідна логіка для функціонування системи, а саме: ArticleController – контролер, який використовується для роботи з постами, ArticleGroupController – логіка для роботи з постами, які належать групі, UserController – контролер, який відповідає за роботу з користувачами, включаючи аутентифікацію, GroupsController – логіка для роботи з групами, UserFollowerController – логіка для роботи з фоловерами користувачів, GroupFollowerController – контролер, який використовується для роботи з фоловерами груп, UsersGroupsController – контролер, який відповідає за роботу з учасниками груп, FriendController – контролер, який використовується для роботи з друзями користувачів, ProfileController – логіка для роботи з профілями користувачів, UserDeviceController – логіка для роботи з пристроями користувачів, UserSettingsController – контролер, який відповідає за роботу з налаштуваннями користувачів, UserTokenController – логіка для роботи з токеном для авторизації користувачів, BaseStorageController – основна логіка для роботи з медіа контентом, ImageStorageController – логіка для роботи із зображеннями, VideoStorageController – логіка для роботи з відео, SoundStorageController – логіка для роботи з аудіо, NotificationSubscriptionController – контролер, який відповідає за роботу з нотифікаціями, LocationController – контролер, який використовується для роботи з локаціями. Лістинг контролерів основного серверу наведено в додатку Г.

Діаграма класів основного серверу зображена на рисунках 3.11 та 3.12 відповідно.

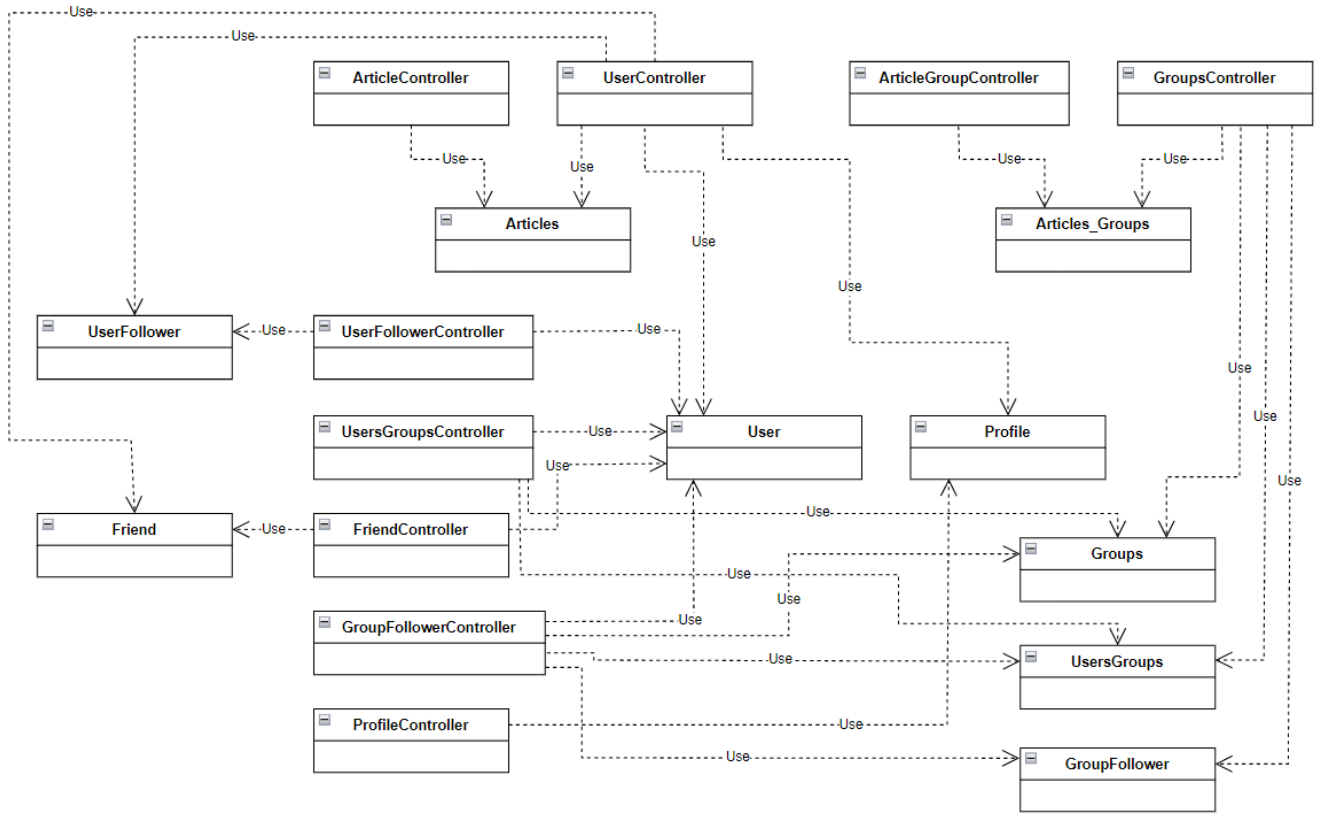

Рисунок 3.11 – Діаграма класів основного серверу (частина 1)

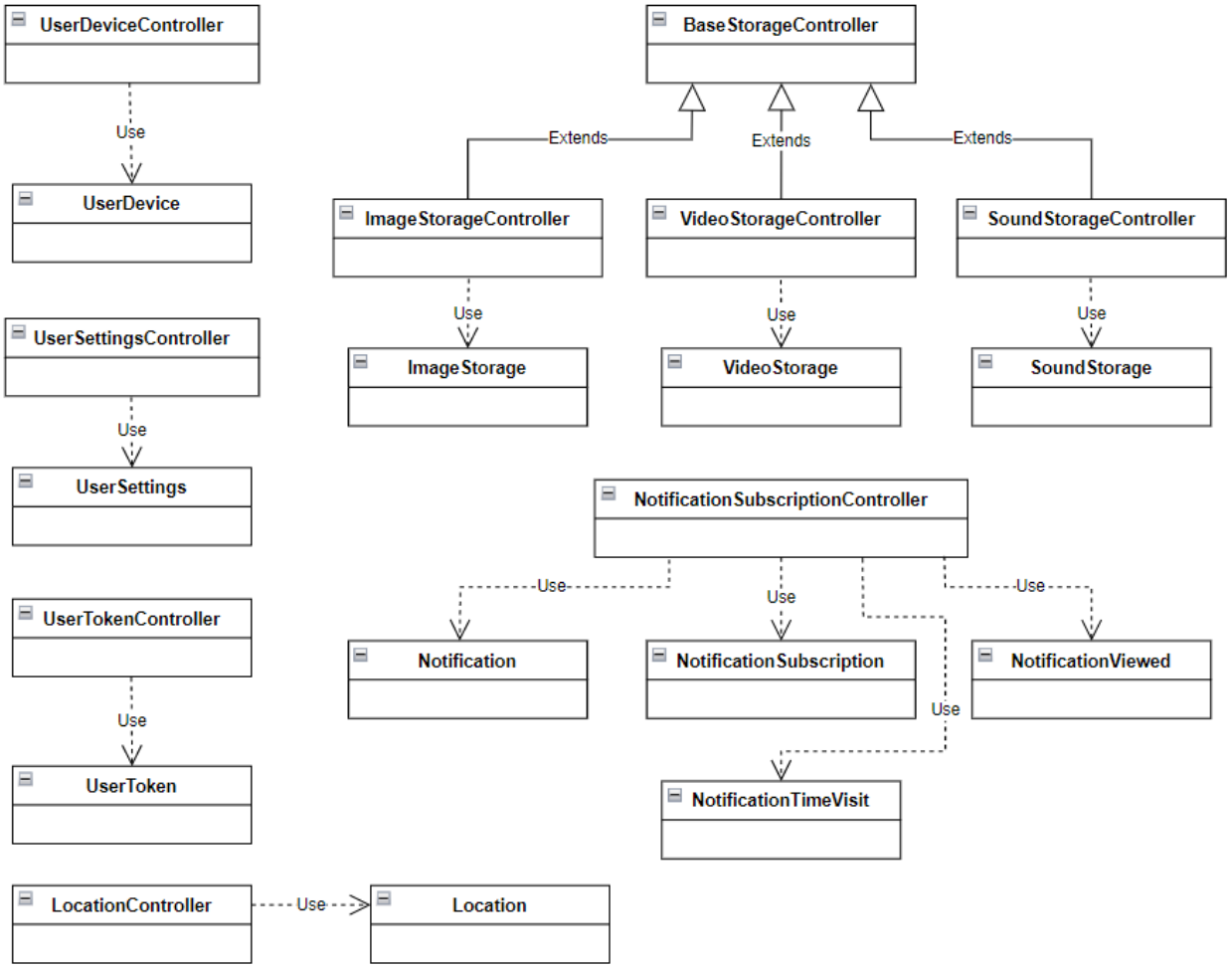

Рисунок 3.12 – Діаграма класів основного серверу (частина 2)

View шар основного серверу буде містити файли з Python методами, які будуть представляти собою API, за допомогою якого клієнт може комунікувати з сервером та надсилати йому запити. Він буде містити в собі такі файли, як: article.py який буде містити API пов’язані з постами, files_storage.py – API для отримання та завантаження медіа контенту, followers.py – API пов’язані з фоловерами користувача, friend.py – API пов’язані з друзями користувача, groups.py – API пов’язані з групами та їх фоловерами, location.py – API пов’язані з локаціями, notifications.py – API пов’язані з нотифікаціями, user.py – API пов’язані з користувачами, включаючи аутентифікацію, users_groups.py – API пов’язані з учасниками груп.

Файли та контролери, які в них використовуються, наведені в таблиці 3.1. Лістинг API шару основного серверу наведено в додатку Д.

Таблиця 3.1 – Файли та контролери, які в них використовуються

№	File	Controllers
1	article.py	ArticleController, ArticleGroupController, FriendController, GroupsController
2	files_storage.py	ImageStorageController, VideoStorageController, SoundStorageController
3	followers.py	UserFollowerController
4	friend.py	FriendController
5	groups.py	GroupsController, GroupFollowerController, UsersGroupsController, UserFollowerController, NotificationSubscriptionController
6	location.py	LocationController
7	notifications.py	NotificationSubscriptionController
8	user.py	UserController, UserTokenController, UserDeviceController, UserSettingsController
9	users_groups.py	UsersGroupsController

Model шар сервісу нотифікацій буде містити два класи: Notification та UserDevice. Лістинг моделей сервісу нотифікацій наведено в додатку Е. Controller шар буде містити клас NotificationController, в якому буде знаходитися вся логіка створення та відправки нотифікацій. Лістинг контролеру сервісу нотифікацій наведено в додатку Ж. View шар буде містити файл з однією API для створення і відправки нотифікацій. Лістинг API шару сервісу нотифікацій наведено в додатку К.

Діаграму класів сервісу нотифікацій наведено на рисунку 3.13.

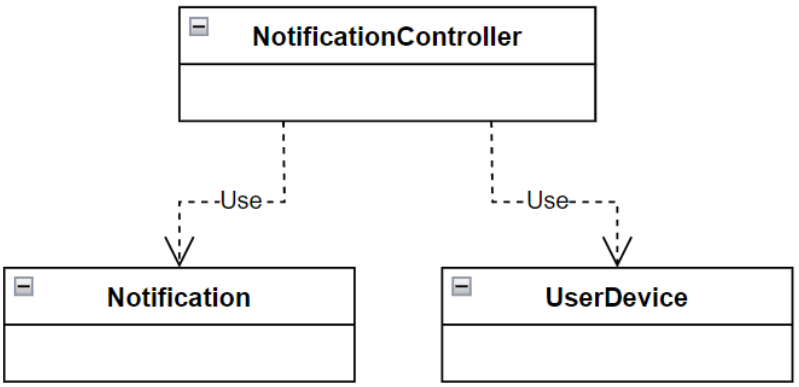

Рисунок 3.13 – Діаграма класів сервісу нотифікацій

Розглянемо деякі основні API, такі як: api_user_register, api_user_login, api_article_add, api_group_add, api_add_location та create_send_notifications.

Принцип роботи API для реєстрації користувача (api_user_register):

1) Користувач надсилає запит на створення акаунту, в тілі запиту відправляються основні дані для реєстрації – юзернейм, електронна пошта, ім’я, фамілія, та пароль;

2) Відбувається створення нового користувача та його логін в систему;

3) Формується авторизаційний токен, який необхідний для того, щоб отримати доступ до інших функцій системи;

4) Зберігаються дані про пристрій користувача;

5) Формується відповідь та відправляється клієнту.

Data Flow діаграма роботи API для реєстрації користувача наведена на рисунку 3.14.

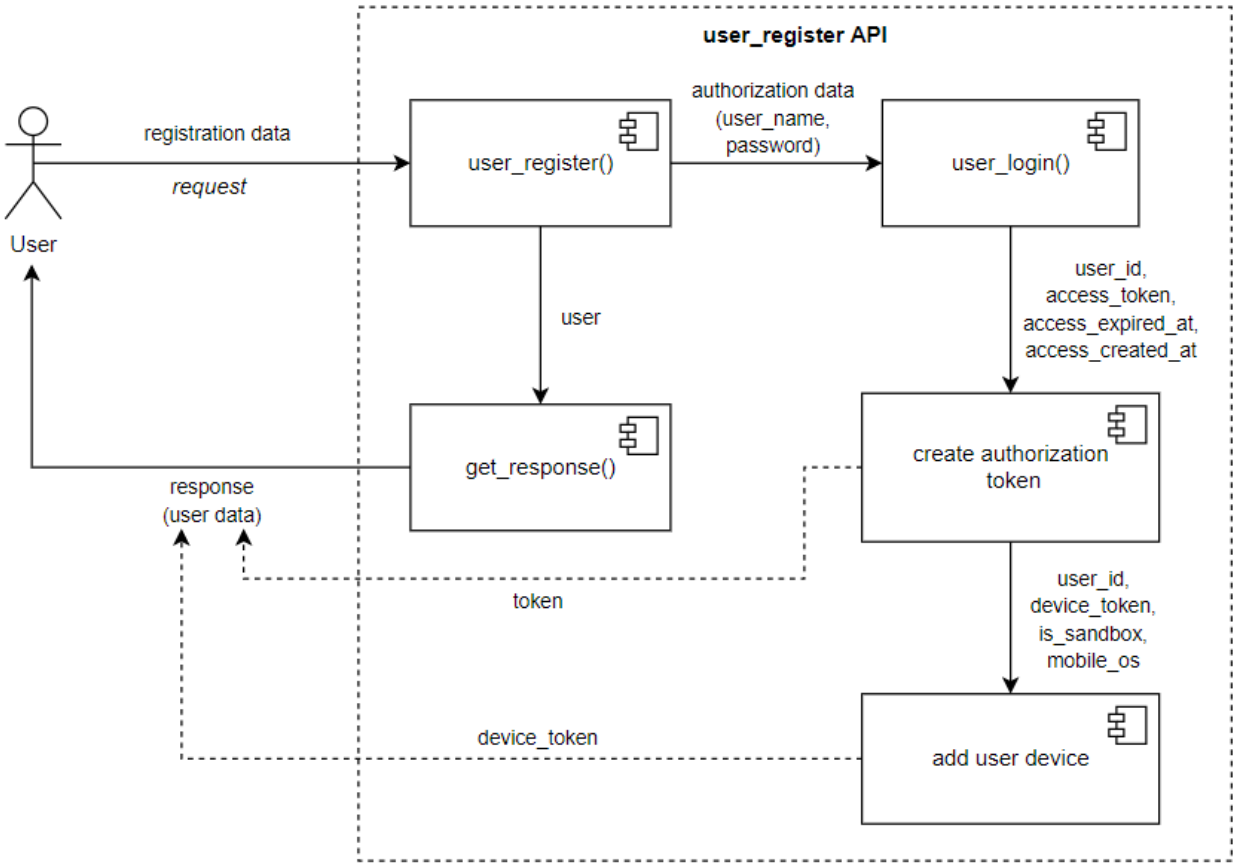

Рисунок 3.14 – Data Flow діаграма роботи API для реєстрації користувача

Принцип роботи API для логіну користувача (api_user_login):

1) Користувач надсилає запит на логін, в тілі запиту відправляються авторизаційні дані – юзернейм та пароль;

2) Сервер авторизує користувача в системі;

3) Відбувається заміна старого токену авторизації;

4) Відбувається оновлення даних про пристрій користувача;

5) Формується відповідь та надсилається клієнту.

Data Flow діаграма роботи API для логіну користувача наведена на рисунку 3.15.

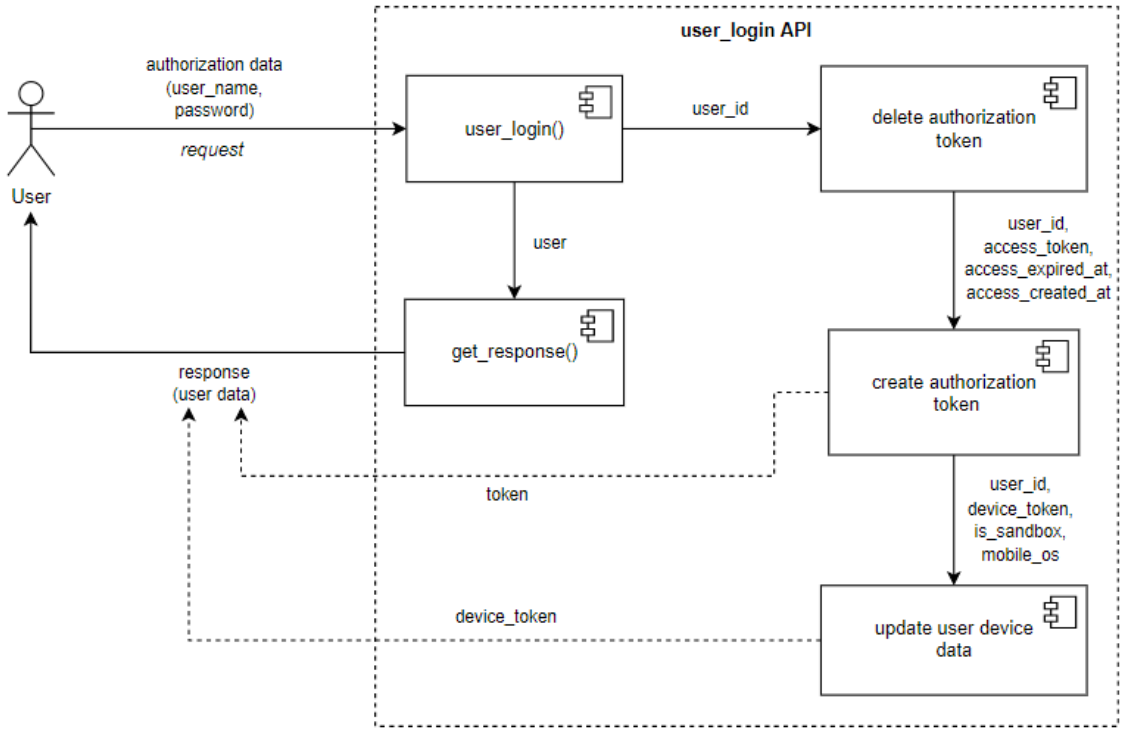

Рисунок 3.15 – Data Flow діаграма роботи API для логіну користувача

Принцип роботи API для створення поста (api_article_add):

1) Користувач надсилає запит на створення нового поста, в тілі запиту відправляються дані про пост (назва, опис, координати, тип, скільки часу цей пост будуть бачити інші користувачі, тощо);

2) Якщо користувач при створенні поста вказав його координати, то йде процес отримання назви локації за допомогою GooglePlacesAPI;

3) Відбувається створення нового поста;

4) Якщо в тілі запиту користувач передав айді групи, то цей пост закріплюється за конкретною групою;

5) Якщо тип поста вказаний як “event” і в тілі запиту вказані айді інших користувачів, то цим користувачам надсилаються запрошення на дану подію;

6) При створенні поста відправляється запит до сервісу нотифікацій на створення та відправку нотифікацій;

7) Формується відповідь і надсилається клієнту.

Data flow діаграма роботи API для створення поста наведена на рисунку 3.16.

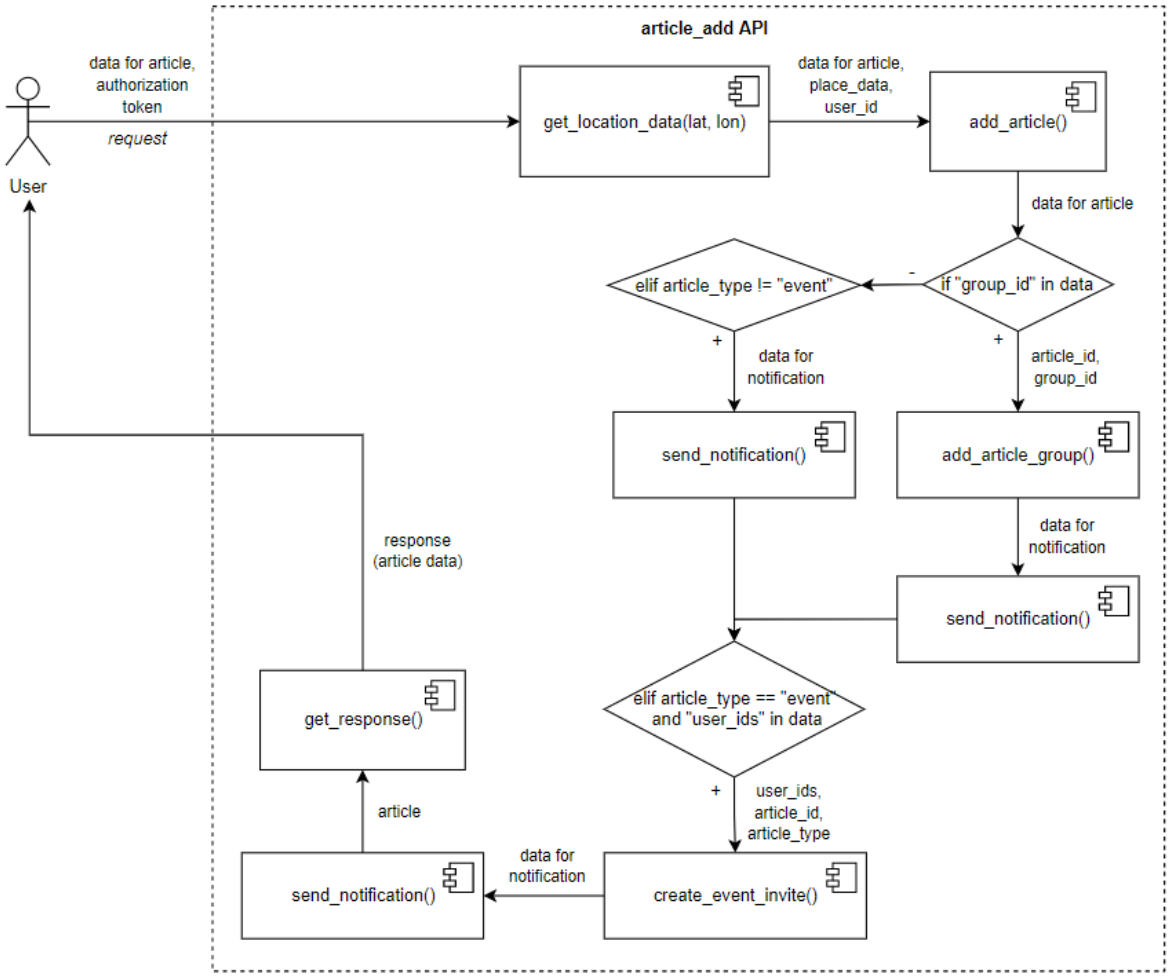

Рисунок 3.16 – Data flow діаграма роботи API для створення поста

Принцип роботи створення групи (api_group_add):

1) Користувач надсилає запит на створення нової групи, в тілі запиту надсилаються дані про групу (назва, опис, тип тощо);

2) Створюється група та учасник, який її створив.

3) Формується відповідь і відправляється клієнту.

Data Flow діаграма роботи API для створення групи наведена на рисунку 3.17.

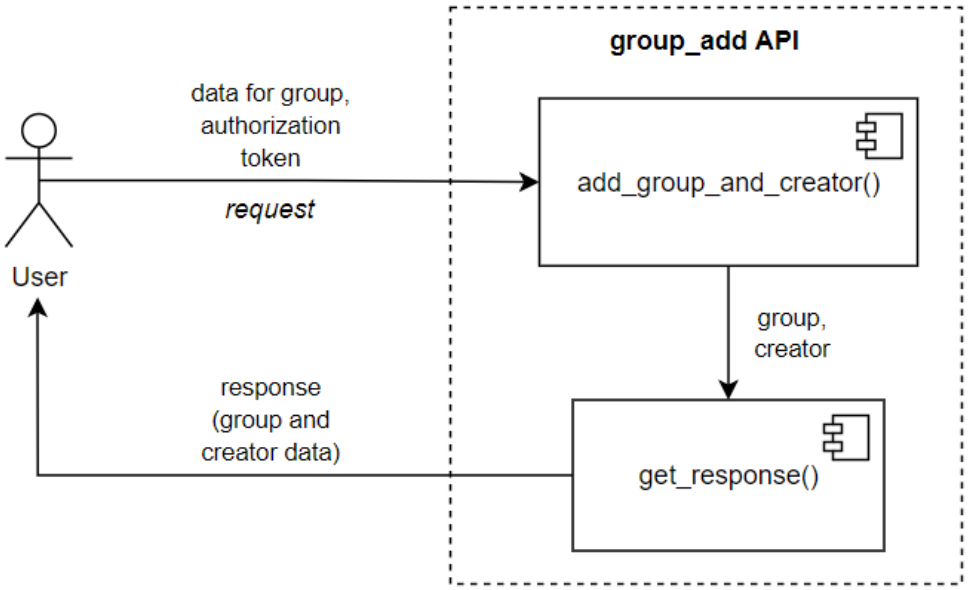

Рисунок 3.17 – Data Flow діаграма роботи API для створення групи

Принцип роботи API для створення локації:

1) Користувач надсилає запит на сервер на створення локації, в тілі запиту надсилаються дані про локацію (назва, адреса та геолокація);

2) Якщо адреса не передається в тілі запиту, то вона формується на основі переданої геолокації;

3) Відбувається створення локації;

4) Формується відповідь і відправляється клієнту.

Data Flow діаграма роботи API для створення локації наведена на рисунку 3.18.

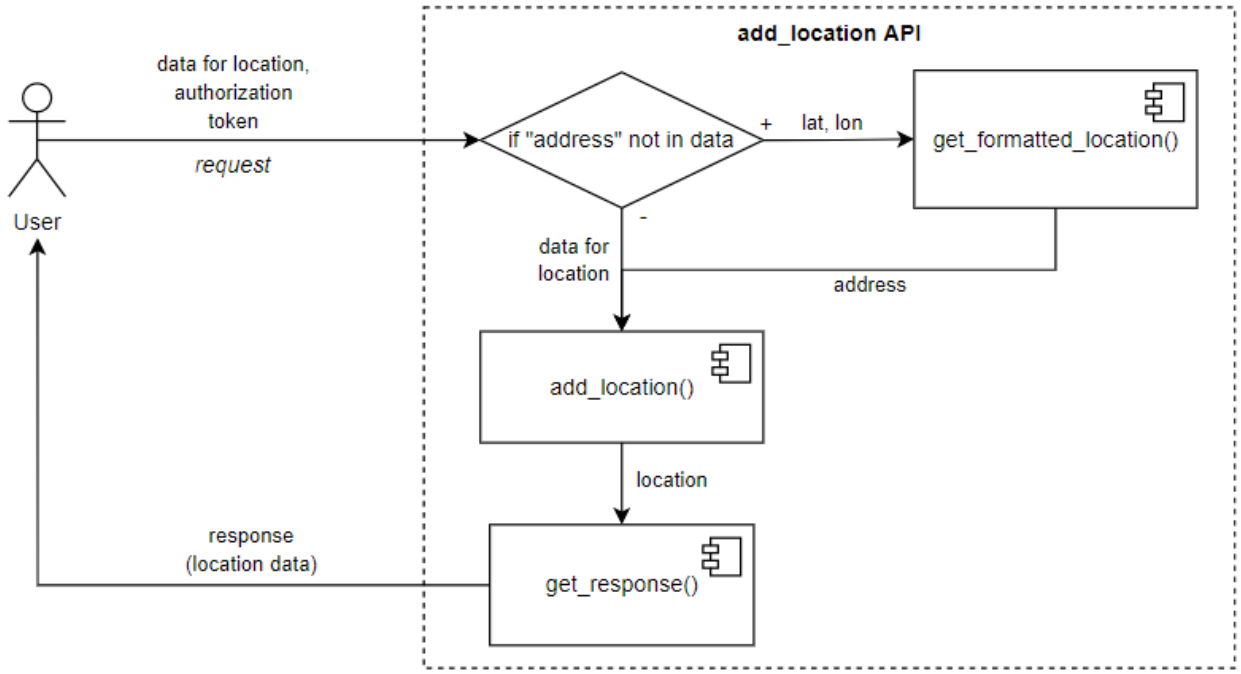

Рисунок 3.18 – Data Flow діаграма роботи API для створення локації

Принцип роботи API для створення та відправки нотифікацій:

1) Основний сервер надсилає запит до серверу нотифікацій. В тілі запиту містяться дані, які необхідні для створення та відправки нотифікацій;

2) Відбувається створення нотифікацій;

3) Отримуються дані по нотифікаціям, які потрібно відправити;

4) Нотифікації відправляються користувачам;

5) Змінюється статус нотифікацій з “created” на “sent”;

6) Отримуються айді нотифікацій та їх статус і видаляються застарілі токени девайсів.

Data Flow діаграма роботи API для створення та відправки нотифікацій наведена на 3.19.

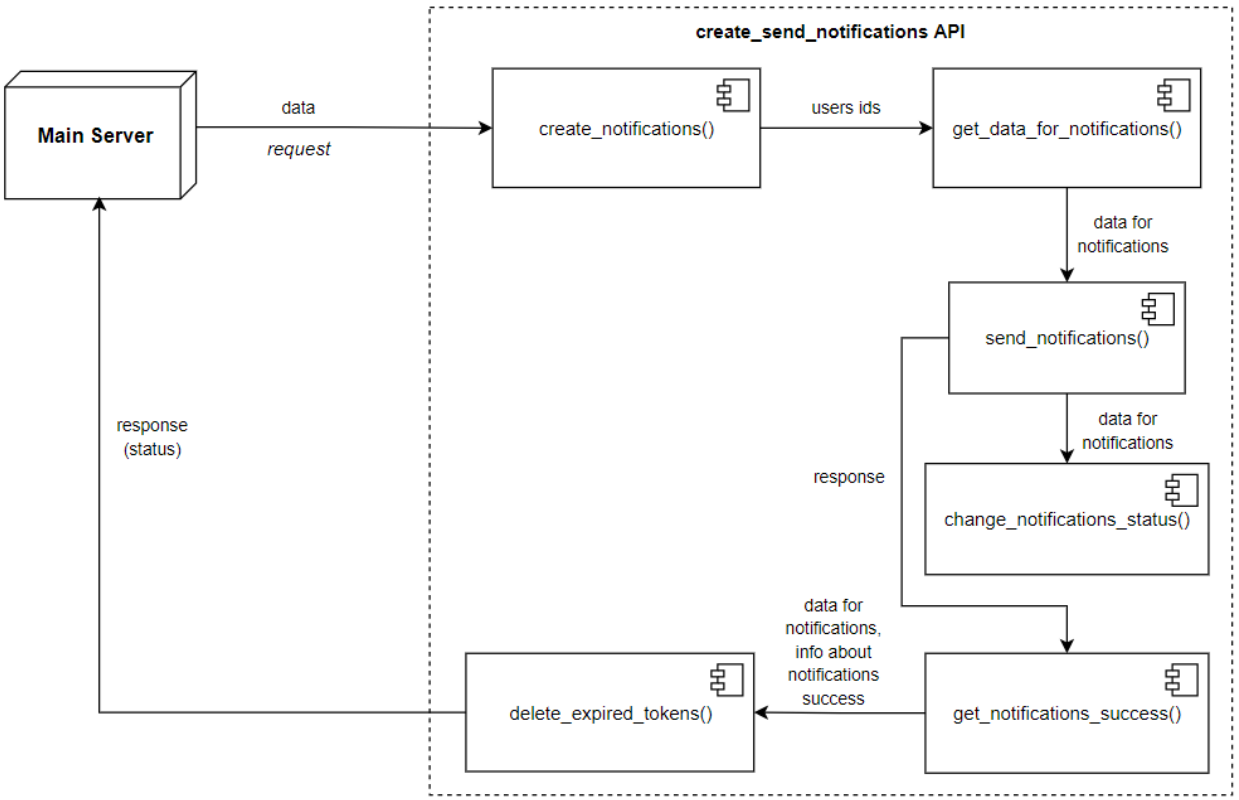

Рисунок 3.19 – Data Flow діаграма роботи API для створення та відправки нотифікацій

Однією з головних особливостей даної системи є рекомендація постів користувачам в залежності від їх місцезнаходження. При створенні поста відправляється запит до сервісу нотифікацій на створення та відправку нотифікацій, в тілі запиту окрім інших даних також передається геолокація поста. Дана геолокація використовуються для отримання списку користувачів яким треба надіслати нотифікації, якщо пост був створений в радіусі користувача, який він вказав у себе в налаштуваннях.

Для того, щоб отримати користувачів спочатку на основі геолокації поста та користувача формуються точки і обчислюється відстань між даними точками.

Для обчислення відстані між точками в кілометрах використовується наступна формула (3.1):

$$
\begin{aligned}
&6371 \cdot acos(\\
&\quad cos(\, radians(< user\_lat >)) \times \\
&\quad \times cos(\, radians(< post\_lat >)) \times \\
&\quad \times cos(\, radians(< post\_lon >) - radians(< user\_lon >)) + \\
&\quad + sin(\, radians(< user\_lat >)) \times \\
&\quad \times sin(\, radians(< post\_lat >)) \\
&)
\end{aligned}
\tag{3.1}
$$

Далі перевіряється чи увімкнена функція отримання нотифікацій навколо користувача. І потім перевіряється чи менша розрахована відстань за радіус користувача, який вказує в якому радіусі користувач буде отримувати нотифікації. Даний SQL запит наведено на рисунку 3.20.

```
SELECT us.user_id,
       us.notif_around_me,
       us.around_radius,
       up.last_lat, up.last_lon,
       (
         6371 * acos (
           cos( radians(up.last_lat) )
           * cos( radians( {post_lat} ) )
           * cos( radians( {post_lon} ) - radians(up.last_lon) )
           + sin( radians(up.last_lat) )
           * sin( radians( {post_lat} ) )
         )
       ) AS distance
FROM user_settings AS us
LEFT JOIN fts_user_profile AS up ON us.user_id = up.user_id
WHERE us.notif_around_me = true
HAVING distance < (us.around_radius / 1000)
```

Рисунок 3.20 – Запит на отримання користувачів для відправки нотифікацій, базуючись на їх місцезнаходженні

3.4 Тестування розробленої системи

Для тестування програмного забезпечення було розроблено юніт тести, з використанням бібліотеки pytest, за допомогою яких було протестовано роботу розглянутих API. Було здійснено перевірку отримання даних та обробки відповіді від серверу, перевірку функціоналу створення нового користувача та його авторизації. Було перевірено функціонал створення поста, групи, та локації. Юніт тести наведено у додатку Л.

Результати модульного тестування наведено на рисунку 3.21.

```
============================================================================================
================== test session starts ==================================================
============================================================
platform win32 -- Python 3.9.6, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: E:\WorkProjects\spnzh
plugins: csv-3.0.0
collected 7 items

app_spnzh\test\test_user.py .

                                                        [ 16%]
app_spnzh\test\test_auth.py ..

                                                        [ 50%]
app_spnzh\test\test_article.py .

                                                        [ 66%]
app_spnzh\test\test_groups.py .

                                                        [ 83%]
app_spnzh\test\test_location.py .

                                                        [100%]

--------------------------------------------------------------------------------------------
------------ CSV report: test_api_result2.csv ----------------------------------------------
------------------------------------------------------------
============================================================================================
================== 6 passed in 1.32s ====================================================
============================================================
```

Рисунок 3.21 – Результати модульного тестування

Звіт з модульного тестування наведено на рисунку 3.22.

	1	2	3	4	5	6	7
1	function	status	message	api_url	data	data_type	request_method
2	test_user_register	passed		/user/register	{'user_name': 'Pytest_user', 'email': 'pytest_user@test.test', 'first_name': 'Pytest First name', 'last_name': 'Pytest Last name', 'password': 'qwerty'}	json	<PreparedRequest [POST]>
3	test_user_logout	passed		/user/logout			<PreparedRequest [GET]>
4	test_user_login	passed		/user/login	{'user_name': 'Pytest_user', 'password': 'qwerty', 'device_token': 'device_token_Pytest_user', 'is_sandbox': 1, 'mobile_os': 1}	json	<PreparedRequest [POST]>
5	test_article_add	passed		/article/add	{'name': 'Pytest article', 'description': 'Pytest article description', 'type': 'note', 'time_visible': 3600, 'lat': 49.225980003148344, 'lon': 28.412428206264764, 'mode': 'global', 'group_id': 0, 'time_delay': 0}	json	<PreparedRequest [POST]>
6	test_group_add	passed		/group/add	{'name': 'Pytest group', 'description': 'Pytest group description', 'mode': 'public'}	json	<PreparedRequest [POST]>
7	test_add_location	passed		/location/add	{'name': 'Pytest location ', 'lat': 49.23369021162874, 'lon': 28.45734510390145}	json	<PreparedRequest [POST]>

Рисунок 3.22 – Звіт з модульного тестування

Отже, в ході тестування розглянутих API не було виявлено ніяких помилок, що свідчить про їх коректну роботу на стороні сервера.

Також необхідно протестувати коректність відправки нотифікації при публікації поста поруч із користувачем. Нехай користувач в налаштуваннях вказав, що бажає отримувати сповіщення в радіусі 150 метрів. Сам користувач має наступні координати: lat 49.232789767931905, lon 28.468203011481233. Координати поста: lat 49.23350863009808, lon 28.46991483662076. Згідно Google Maps відстань між користувачем та місцем створення публікації приблизно дорівнює 147 метрів. Це означає, що коли пост буде створений, користувач отримає про це сповіщення.

Відстань між користувачем та місцем публікації поста наведено на рисунку 3.23.

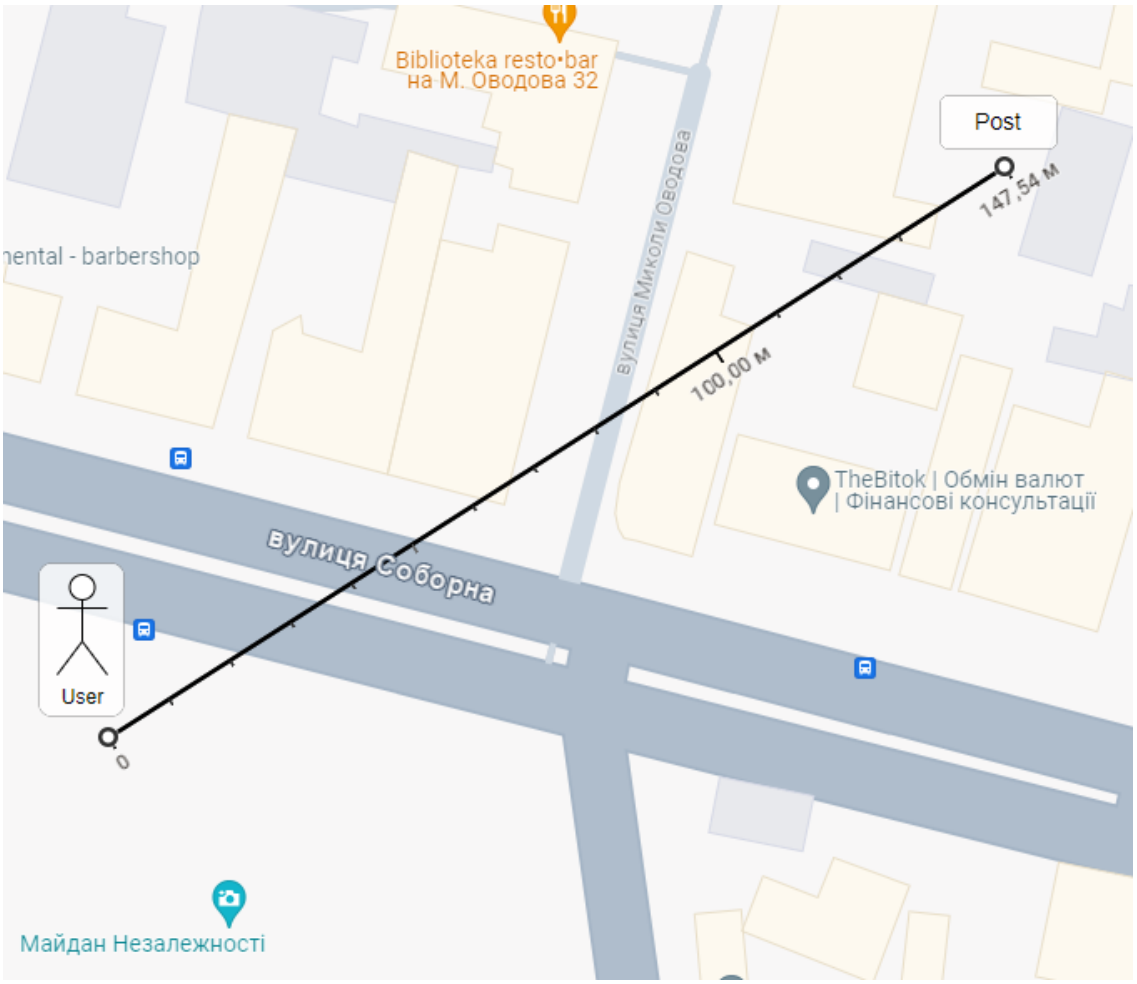

Рисунок 3.23 – Відстань між користувачем та місцем публікації поста

Відправляємо запит на створення поста з наступними обов'язковими полями в тілі запиту: "name", "description", "type", "time_visible" (в даному полі вказується час життя поста в секундах, після чого даний пост перестає показуватися користувачам), "lat" та "lon" (координати поста), "mode", "group_id" (в даному полі передається id групи, що означає, що пост опублікувала група. Якщо ж в полі передається значення 0 (нуль), то пост публікується користувачем), "time_delay". Запит на створення поста разом із тілом запиту наведено на рисунку 3.24.

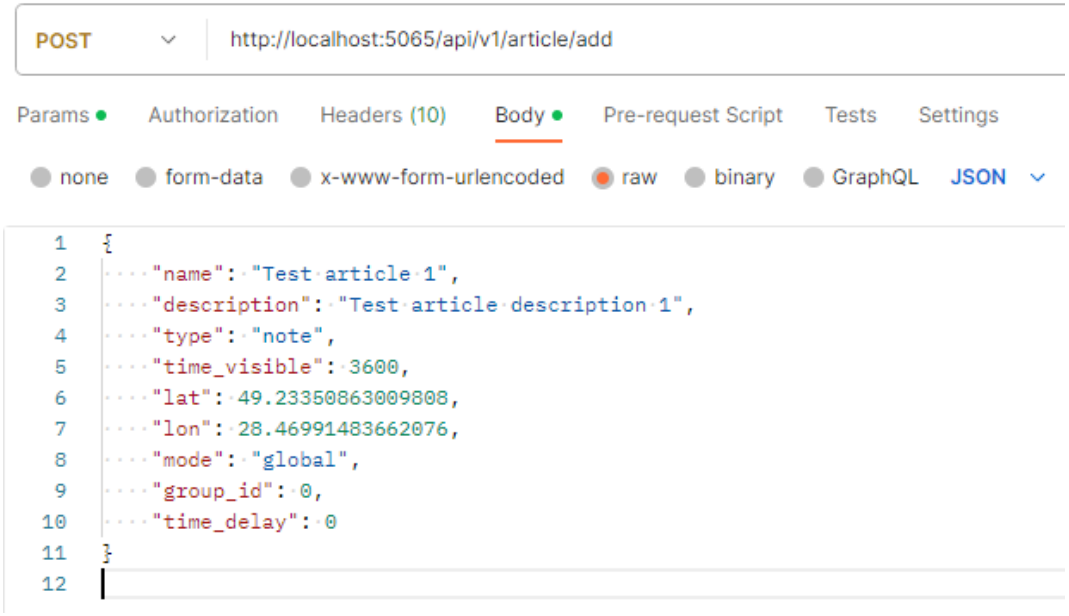

Рисунок 3.24 – Запит на створення поста

Далі надсилаємо запит на отримання списку нотифікацій користувача, якому повинно було надійти сповіщення.

Запит на отримання списку нотифікацій наведено на рисунку 3.25.

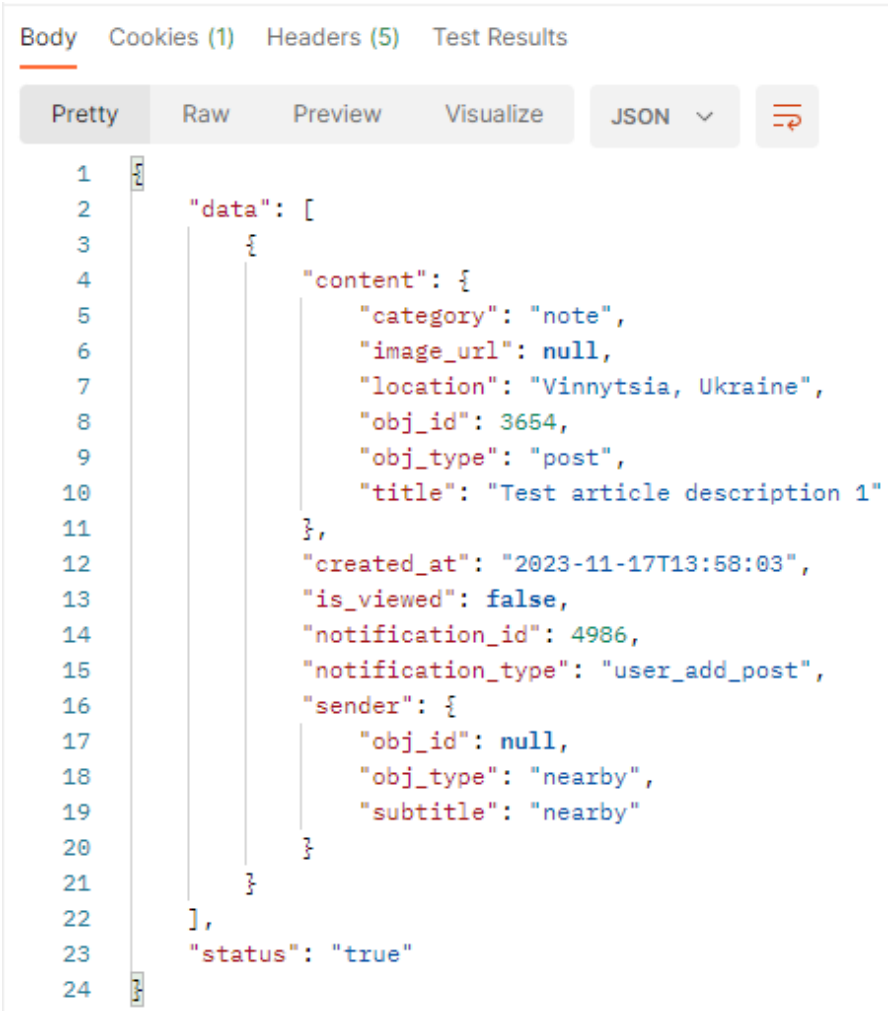

Рисунок 3.25 – Запит на отримання списку нотифікацій

Результатом запиту є список нотифікацій, які отримав користувач. Як видно з вищенаведеного рисунку, користувач отримав сповіщення про те, що поруч з ним була створена публікація.

Протестуємо сценарій, при якому пост створюється за радіусом користувача. Координати користувача залишаються аналогічними, як і в попередньому тесті. Координати нового поста: lat 49.23354195495887, lon 28.469994579167448. Згідно Google Maps відстань між користувачем та місцем створення публікації приблизно дорівнює 154 метрів, що більше за радіус користувача в якому він бажає отримувати сповіщення. При такій ситуації користувач не отримає нової нотифікації. Відстань між користувачем та місцем публікації поста наведено на рисунку 3.26.

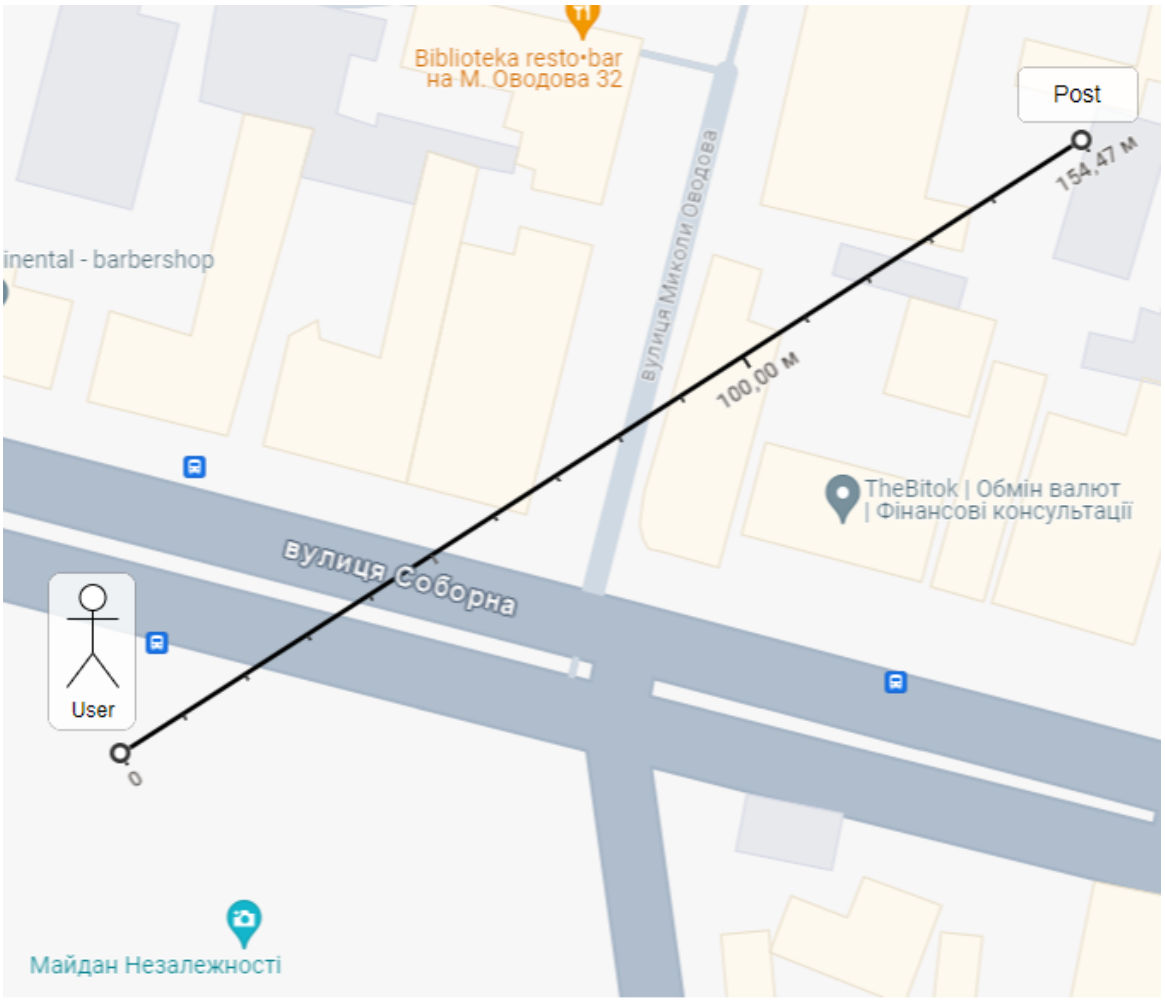

Рисунок 3.26 – Відстань між користувачем та місцем публікації поста

Запит на створення поста наведено на рисунку 3.27.

Рисунок 3.27 – Запит на створення поста

Надсилаємо запит на отримання списку нотифікацій користувача.

Запит на отримання списку нотифікацій наведено на рисунку 3.28.

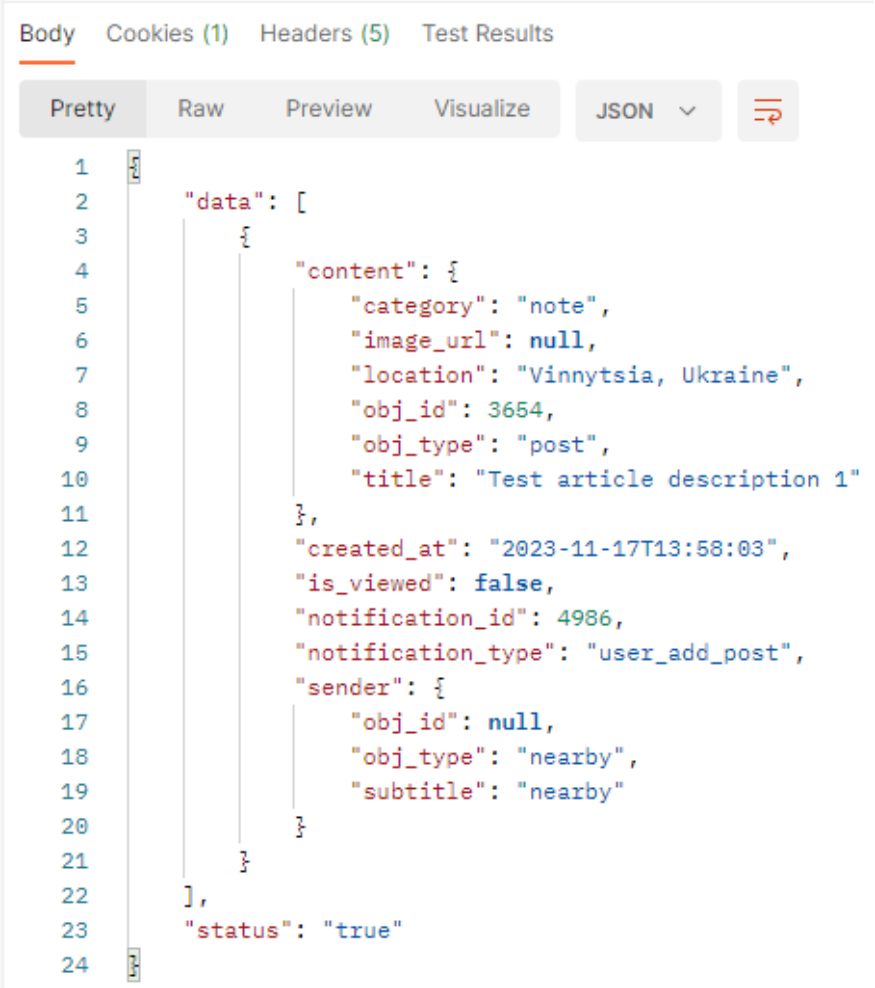

Рисунок 3.28 – Запит на отримання списку нотифікацій

З результату видно, що список нотифікацій користувача містить лише старе сповіщення, а нотифікації про нову публікацію немає. Це свідчить про те, що користувач не отримав нової нотифікації про створений пост.

3.5 Висновки до розділу

Для вирішення поставленої задачі були наведені основні етапи та технології розробки серверної частини. Було розглянуто підхід реалізації архітектури серверної частини клієнт-серверного додатку з використанням підходу REST API та патерну MVC. В ході розробки програмного забезпечення було розроблено структуру бази даних даної системи. Було розроблено систему згідно вимог та продемонстровано роботу основних API. Після завершення процесу розробки програмного забезпечення було здійснено написання юніт тестів та протестовано систему на предмет коректного функціонування.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Технологічний аудит розробленої мобільної клієнт-серверної системи соціальної мережі

Як було зазначено у попередніх розділах даної роботи, люди в усьому світі використовують соціальні мережі, щоб бути в курсі останніх подій, отримувати актуальні новини та залишатися на зв’язку. Соціальні мережі надають інструменти для міжособової та масової комунікації, пошуку потрібної інформації, перегляду новин тощо.

Але саме через велику кількість інформації, що наразі публікується, постає питання в її достовірності. Тому не випадково, в Україні розгорнулася широка робота з оцифровування різноманітних паперових та інших процесів і створення різних веб-сайтів, користування якими дозволяє користувачам цих сайтів оптимізувати свою роботу.

Тому виконана магістерська кваліфікаційна робота переслідувала мету шляхом створення мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації, суттєво покращити комунікації між користувачами інформації та підвищити її достовірність.

Для досягнення поставленої мети було: проаналізовано основні принципи функціонування соціальних та геолокаційних соціальних мереж; вивчено існуючі реалізації геолокаційних соціальних мереж; проаналізовано технології та підходи до створення серверної частини соціальної мережі; розроблено алгоритмічне та програмне забезпечення серверної частини соціальної мережі; проведено тестування розробленого програмного забезпечення.

В результаті було запропоновано соціальну мережу, яка базується на удосконалених методах збирання і обробки інформації, яка надходить від користувачів при використанні засобів геолокації.

Для встановлення рівня комерційного потенціалу розробленої мобільної клієнт-серверної системи соціальної мережі було проведено її технологічний

аудит, для чого було запрошено 3-х експертів: д.т.н., професора Паламарчука Є.А., к.т.н., доцентів Маслія Р.В. та Сторчака В.Г.

Оцінювання комерційного потенціалу розробленої мобільної клієнт-серверної системи соціальної мережі здійснено за критеріями, наведеними в таблиці 4.1.

Таблиця 4.1 – Критерії оцінювання рівня комерційного потенціалу будь-якої розробки та їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатацій-них витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Запрошені експерти оцінили нашу розробку так, як це наведено в таблиці 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали експерта		
	Паламарчук Є.А.	Маслій Р.В.	Сторчак В.Г.
	Бали, виставлені експертами:		
1	3	3	3
2	3	3	4
3	3	3	3
4	3	3	3
5	3	3	4
6	3	4	3
7	4	3	3
8	3	4	3
9	3	3	4
10	4	4	3
11	3	4	3
12	3	3	3
Сума балів	$СБ_1 = 38$	40	39

Середньоарифметична сума балів $\overline{СБ}$, що їх виставили експерти, становила:

$$\overline{СБ} = \frac{\sum_{1}^{3} Б_i}{3} = \frac{38+40+39}{3} = \frac{117}{3} = 39.$$

Загальний рівень комерційного потенціалу будь-якої розробки можна встановити, керуючись таблицею 4.3 [57].

Таблиця 4.3 – Рівні технічного та комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень технічного та комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Керуючись рекомендаціями таблиці 4.3, можна зробити висновок, що розроблена мобільна клієнт-серверна система соціальної мережі, яка оцінена шанованими експертами у 39 балів, має технічний рівень та комерційний потенціал, який вважається значно «вище середнього».

Це пояснюється тим, що розроблена соціальна мережа дозволяє створювати публікації, до яких буде прикріплюватися геолокація, що відображає поточне місце перебування користувача; дозволяє створювати публікації лише з того пристрою, з якого було зроблене фото, зняте відео або записаний звук тощо; дозволяє переглядати створені публікації на карті з відображенням геолокації; дозволяє отримувати сповіщення від певної локації, коли там публікується пост, а також отримувати сповіщення, якщо пост публікується поруч із користувачем.

4.2 Розрахунок витрат на розроблення мобільної клієнт-серверної системи соціальної мережі

Під час виконання даної роботи було зроблено такі витрати:

1. Основна заробітна плата виконавців $З_о$:

$$З_о = \frac{М}{Т_р} \cdot t \text{ грн}, \quad (4.1)$$

де М – місячний посадовий оклад конкретного виконавця, грн;

У 2023 році величини посадових окладів науковців знаходиться в межах (6700…24000) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р = 25$ днів.

Розрахунки основної заробітної плати виконавців зведемо до таблиці 4.4.

Таблиця 4.4 – Розрахунок основної заробітної плати виконавців (розробників)

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день (або за годину), грн	Число днів роботи	Витрати на оплату праці, грн	Примітка
1. Науковий керівник магістерської квалі-фікаційної роботи	21725	869,00	20 годин	≈ 2897	6 годин у день
2. Студент-розробник-магістрант	6700	268	80 днів	21440	
3. Консультант з економічної частини	16600	664	1,5 годин	166	6 годин у день
4. Інші консультанти	15000	600	3 дні	1800	
Всього				26303	

2. Додаткова заробітна плата виконавців $З_д$ розраховується як (10…12)% від величини основної заробітної плати виконавців, тобто:

$$З_Д = (0,1...0,12) \cdot З_о. \quad (4.2)$$

Для даного випадку отримаємо:

$$З_д = 0,1075 \cdot 26303 = 2827,57 \approx 2828 \text{ грн.}$$

3. Нарахування на заробітну плату $Н_{зп}$ розраховуються за формулою:

$$Н_{ЗП} = (З_о + З_Д) \cdot \frac{\beta}{100}, \quad (4.3)$$

де $З_о$ – основна заробітна плата виконавців, грн;

$З_д$ – додаткова заробітна плата виконавців, грн;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування; β = 22%.

Тоді отримаємо:

$$Н_{зп} = (26303+2828) \cdot 0,22 = 6408,82 \approx 6409 \text{ грн.}$$

4. Витрати на матеріали М розраховуються по кожному виду матеріалів:

$$М = \sum_1^n Н_i \cdot Ц_i \cdot К_i - \sum_1^n В_i \cdot Ц_в \text{ грн}, \quad (4.4)$$

де $Н_i$ – витрати матеріалу *i*-го найменування, кг; $Ц_i$ – вартість матеріалу *i*-го найменування, грн/кг; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); $В_i$ – маса відходів матеріалу *i*-го найменування, кг; $Ц_в$ – ціна відходів матеріалу *i*-го найменування, грн/кг; n – кількість видів матеріалів.

5. Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_1^n Н_i \cdot Ц_i \cdot К_i \text{ грн}, \quad (4.5)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.; $Ц_i$ – ціна комплектуючих *i*-го виду, грн; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів комплектуючих.

За аналогією з іншими розробками вартість всіх використаних матеріальних ресурсів становить приблизно 1475 грн.

6. Амортизацію А обладнання, комп'ютерів та приміщень А можна розрахувати за формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ грн}, \quad (4.6)$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань: $Н_а$ = (2...25)%;

Т – термін, використання обладнання, приміщень тощо, місяці.

Зроблені розрахунки зведено до таблиці 4.5.

Таблиця 4.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн	Норма амортиза-ції, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
1. Персональні комп'ютери, принтери тощо	59500	25	3,5 (60%)	2603,12
2. Приміщення кафедри та факультету	24500	2,5	3,5 (40%)	71,46
Всього				А = 2674,58 ≈ 2675

7. Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \quad (4.7)$$

де В – вартість 1 кВт-год. електроенергії, в 2023 р. В ≈ 4,5 грн/кВт;

П – установлена потужність обладнання, кВт; П = 0,85 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 275 годин;

$К_п$ – коефіцієнт використання потужності; $К_п < 1 = 0,91$.

$К_д$ – коефіцієнт корисної дії, $К_д = 0,83$.

Тоді витрати на електроенергію складуть:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{4{,}5 \cdot 0{,}85 \cdot 275 \cdot 0{,}91}{0{,}83} = 1153{,}26 \approx 1154 \text{ грн.}$$

8. Інші витрати $В_{ін}$ можна прийняти як (50…300)% від основної заробітної плати виконавців, тобто:

$$В_{ін} = К_{ін} \cdot З_о = (0{,}5 .. 3{,}0) \cdot З_о. \quad (4.8)$$

Для нашого випадку домовимося, що $К_{ін}$ = 3,00. Тоді:

$$В_{ін} = (3{,}0 \cdot 26303) = 78909 \text{ грн.}$$

9. Сума всіх попередніх статей витрат дає нам витрати на виконання етапу роботи безпосередньо розробником-магістрантом – В.

Для нашого випадку:

В = 26303 + 2828 + 6409 + 1475 + 2675 + 1154 + 78909 = 119753 грн.

10. Розрахунок загальних витрат на розробку та остаточне доопрацювання виконаної роботи здійснюється за формулою:

$$ЗВ = \frac{В}{\beta}, \quad (4.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Оскільки дана розробка ще потребує деякого дуже незначного доопрацювання, то можна прийняти, що β ≈ 0,7.

Тоді: $ЗВ = \frac{119754}{0{,}7} = 171075{,}71 \approx 172000$ грн або приблизно 172 тисячі грн.

Тобто прогнозовані витрати на розроблення та остаточне доопрацювання розробленої мобільної клієнт-серверної системи соціальної мережі, становлять приблизно 172 тисячі грн.

4.3 Розрахунок економічного ефекту від можливої комерціалізації даної розробки

Аналіз ринку показує, що розроблена мобільна клієнт-серверна система соціальної мережі безумовно знайде своє місце на сучасному ринку (не зважаючи на те, що цей ринок є досить конкурентним), оскільки дана розробка має значно кращі і ширші функціональні можливості.

Оскільки сьогодні, за підрахунками, даною клієнт-серверною системою можуть користуватися 1 млн користувачів, які разом з підпискою та рекламою можуть принести її власнику річний дохід у $ 3650 тисяч (реклама) та $ 4800 тисяч (підписка), що разом становить $ 8460 тисяч або 338,4 млн грн, то і ціна реалізації побідної системи також буде досить високою.

Якщо врахувати, що кількість потенційних користувачів даної клієнт-серверної системи становить 1 млн підписників, то тоді кожен з них у середньому протягом року «приносить дохід» у 110 млн грн / 1 млн = 110 грн/користувача.

Окрім того, при подальших розрахунках потрібно врахувати і зростання ціни «послуги», яку отримують користувачі цієї системи.

Аналіз місткості ринку також показує, що в Україні кількість потенційних користувачів подібних систем також буде зростати. Припустимо також, що дана розробка буде користуватися попитом на ринку протягом 3-х років після впровадження. Якщо дана розробка буде впроваджена з 1 січня 2024 року, то її результати будуть виявлятися протягом 2024-го, 2025-го та 2026-го років.

За висновками запрошених експертів, зростання ціни «послуги» для користувачів в середньому буде становити + 5% до попереднього року, а зростання користувачів даної системи буде зростати +10% до попереднього року.

Тоді очікуваний дохід Д від функціонування розробленої мобільної клієнт-серверної системи соціальної мережі становитиме:

2024-й рік: $Д_{2024} = 110 \cdot 1{,}05 \cdot 1000000 \cdot 1{,}1 = 127\,050\,000 \approx 127$ млн грн;

а зростання доходу $\Delta Д_{2024}$ = 127-110 = 17 млн грн.

2025-й рік: $Д_{2025}$ = 127 · 1,05 · 1000000 · 1,1 = 146 685 000 ≈ 147 млн грн;

а зростання доходу $\Delta Д_{2025}$ = 147-127 = 20 млн грн.

2026-й рік: $Д_{2026}$ = 147 · 1,05 · 1000000 · 1,1 = 169 785 000 ≈ 170 млн грн;

а зростання доходу $\Delta Д_{2026}$ = 170-147 = 23 млн грн.

Можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації даної розробки становитиме:

$$\Delta П_i = \sum_1^n \Delta Д_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}), \quad (4.10)$$

де $\Delta Д_i$ – збільшення величини доходу у кожному році;

λ – коефіцієнт, який враховує сплату податку на додану вартість; λ = 0,8333;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати ρ = (0,2...0,5); візьмемо ρ = 0,5;

υ – ставка податку на прибуток. У 2023-2025 роках υ = 18%.

Тоді зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого (2024) року може скласти:

$$\Delta П_1 = 17 \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) \approx 5,81 \text{ млн грн.}$$

Зростання чистого прибутку $\Delta П_2$ для потенційного інвестора протягом другого (2025) року може скласти:

$$\Delta П_2 = 20 \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) \approx 6,83 \text{ млн грн.}$$

Зростання чистого прибутку $\Delta П_3$ для потенційного інвестора протягом третього (2026) року може скласти:

$$\Delta П_3 = 23 \cdot 0,8333 \cdot 0,5 \cdot (1 - \frac{18}{100}) \approx 7,85 \text{ млн грн.}$$

За висновками експертів, теперішню вартість інвестицій PV, які можуть бути вкладені в придбання розробленої мобільної клієнт-серверної системи соціальної мережі становлять 5 млн грн, тобто: PV = 5 млн грн.

Далі розраховуємо абсолютний ефект від можливо вкладених в придбання розробленої мобільної клієнт-серверної системи соціальної мережі інвестицій $Е_{абс}$:

$$Е_{абс} = ПП - PV, \quad (4.11)$$

де ПП – приведена вартість зростання всіх можливих чистих прибутків потенційного інвестора, грн;

PV – теперішня вартість інвестицій, PV ≈ 5 млн грн.

У свою чергу, приведена вартість зростання всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \quad (4.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої роботи, грн;

Т – період часу, протягом якого виявляються результати впровадженої розробки, роки; Т = 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний рівень інфляції в країні. Для України в 2023 році приймемо ставку = 0,10 (10%);

t – період часу (в роках) від моменту початку розробки мобільної клієнт-серверної системи соціальної мережі до моменту отримання потенційним інвестором чистих прибутків.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від можливої комерціалізації даної розробки, складе:

$$ПП = \frac{5,81}{(1+0,10)^2} + \frac{6,83}{(1+0,1)^3} + \frac{7,85}{(1+0,10)^4} = 4,80 + 5,13 + 5,36 = 15,29 \text{ млн грн.}$$

Абсолютний ефект від можливої комерціалізації даної розробки складе:

$$Е_{абс} = 15,29 – 5 = 10,29 \text{ млн грн.}$$

Далі розрахуємо відносну ефективність $E_в$ вкладених у дану розробку потенційних інвестицій:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.13)$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій, $E_{абс}$ = 10,29 млн грн;

PV – теперішня вартість початкових інвестицій, PV = 5 млн грн;

$Т_ж$ – життєвий цикл використання даної розробки, роки. $Т_ж$ = 4.

Для даного випадку:

$$E_в = \sqrt[4]{1 + \frac{10,29}{5}} - 1 = \sqrt[4]{1 + 2,058} - 1 = \sqrt[4]{3,058} - 1 = 1,322 - 1 =$$

$$= 0,322 \approx 32,2\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційний інвестор не буде зацікавлений вкладати кошти у комерціалізацію даної розробки.

Мінімальна дохідність або мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \quad (4.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках;

в 2023 році в Україні d = (0,10...0,12);

f – показник, що характеризує ризикованість вкладень;

зазвичай, величина f = (0,05...0,30), але може бути і значно більше.

Для даного випадку отримаємо:

$$\tau_{мін} = 0,10 + 0,20 = 0,30 \text{ або } \tau_{мін} = 30\%.$$

Оскільки величина $E_в$ = 32,2% > $\tau_{мін}$ = 30%, то потенційний інвестор (після проведення додаткових розрахунків) у принципі може бути зацікавлений у комерціалізації даної розробки.

Далі розраховуємо термін окупності коштів, які можуть бути вкладені у придбання та комерціалізацію розробленої мобільної клієнт-серверної системи соціальної мережі:

$$T_{ОК} = \frac{1}{E_В}. \quad (4.15)$$

Для даного випадку термін окупності можливих інвестицій $T_{ок}$ складе:

$$T_{ОК} = \frac{1}{0{,}323} \approx 3{,}09 \text{ років.}$$

Оскільки $T_{ок}$ < (3…5) років, то комерціалізація розробленої мобільної клієнт-серверної системи соціальної мережі в принципі є можливою.

Якщо рівень інфляції в країні зросте до 20%, то отримаємо:

$$ПП = \frac{5{,}81}{(1+0{,}20)^2} + \frac{6{,}83}{(1+0{,}2)^3} + \frac{7{,}85}{(1+0{,}20)^4} = 4{,}03 + 3{,}95 + 3{,}76 = 11{,}74 \text{ млн грн.}$$

Абсолютний ефект від можливої комерціалізації даної розробки складе:

$E_{абс}$ = 11,74 – 5 = 6,74 млн грн.

Далі розрахуємо відносну ефективність $E_в$ вкладених у дану розробку потенційних інвестицій:

$$E_В = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.16)$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій, $E_{абс}$ = 6,74 млн грн;

PV – теперішня вартість початкових інвестицій, PV = 5 млн грн;

$Т_ж$ – життєвий цикл використання даної розробки, роки. $Т_ж$ = 4.

Для даного випадку:

$$E_В = \sqrt[4]{1 + \frac{6{,}74}{5}} - 1 = \sqrt[4]{1 + 1{,}348} - 1 = \sqrt[4]{2{,}348} - 1 = 1{,}237 - 1 = 0{,}237 \approx 23{,}7\%.$$

Оскільки величина $E_в$ = 23,7% < $\tau_{мін}$ = 30%, то потенційний інвестор може бути не зацікавлений у комерціалізації даної розробки.

Зроблені розрахунки наведено на рисунку 4.1.

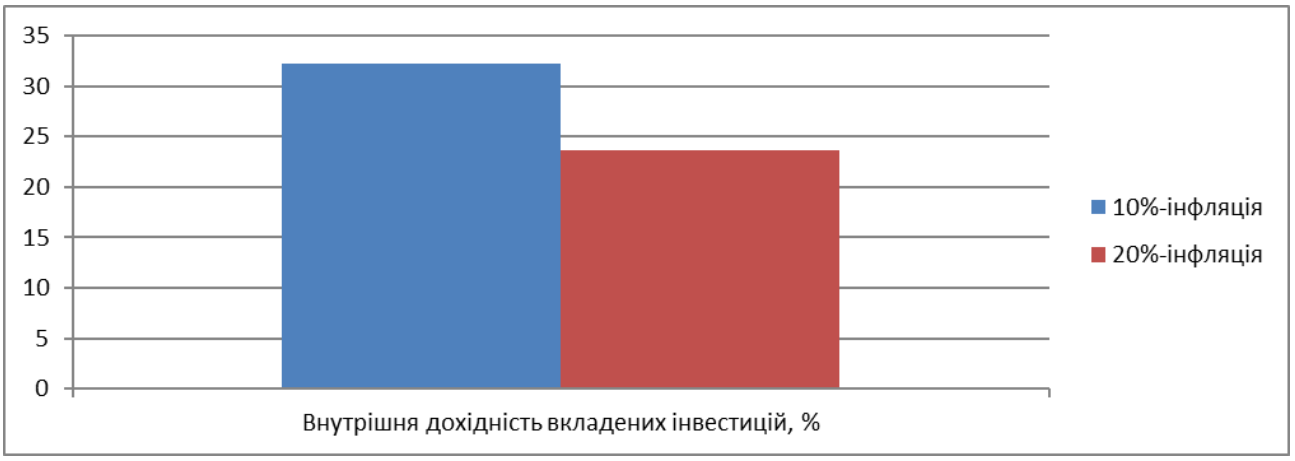

Рисунок 4.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій від рівня інфляції в країні

Аналіз діаграм на рисунку 4.1 показує, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в$ = 32,2%, що більше порогового значення $\tau_{мін}$ = 30% і тому комерціалізація даної розробки може бути доцільною. При рівні інфляції в 20% величина внутрішньої дохідності інвестицій, вкладених в комерціалізацію даної розробки, становить $E_в$ = 23,7%, що нижче порогового значення $\tau_{мін}$ = 30%, і тому комерціалізація даної розробки потенційним інвестором може бути проблематичною.

Остаточне рішення з цього питання потребує проведення додаткових розрахунків (можливо – зниження рівня ризикованості вкладень тощо).

Результати виконаної економічної частини магістерської кваліфікаційної роботи наведено у таблиці 4.6.

Таблиця 4.6 – Результати виконаної економічної частини магістерської кваліфікаційної роботи

Показники	Задані у ТЗ	Досягнуті у магістерській роботі	Висновок
1. Витрати на розроблення мобільної клієнт-серверної системи соціальної мережі	Не більше 200 тис. грн	172 тисяч грн	Виконано
2. Абсолютний економічний ефект від впровадження розробки, млн грн	не менше 10 млн грн	10,29 млн грн	Виконано
3. Внутрішня дохідність потенційних інвестицій, %	не менше 30%	32,3 %	Досягнуто
4. Термін окупності потенційних інвестицій, роки	до 3-х років	3,09 років	Практично досягнуто

Таким чином, заплановані у технічному завданні основні техніко-економічні показники розробленої мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації, практично виконані.

ВИСНОВКИ

В даній магістерській кваліфікаційній роботі було розглянуто актуальність та перспективи використання соціальних мереж, що базуються на основі геолокації. Також було проведено порівняльну характеристику існуючих систем та аналогів, таких як Foursquare, Instagram та Zenly, розглянуті їхні можливості.

Розглянуто технології, які необхідні для реалізації серверної частини подібних клієнт-серверних систем. Було досліджено та обґрунтовано вибір мови програмування, фреймворку, та серверу баз даних для реалізації системи та основні можливості і переваги цих технологій. Під час дослідження було порівняно серверні такі мови програмування, як Python, JavaScript, Java, а також фреймворки, зокрема Flask і Django, і сервери баз даних, такі як MySQL і PostgreSQL. Було ознайомлено з протоколом взаємодії між клієнтською та серверною частинами системи та обрано REST технологію для цих цілей. Для реалізації даної системи було обрано мову програмування Python, фреймворк – Flask, та сервер баз даних – MySQL.

Також було розглянуто архітектурні підходи та патерни, що використовуються для побудови клієнт-серверних систем. Ознайомлено із класифікацією патернів та їх призначенням. У якості патерну для розподілення структури серверної частини було обрано архітектуру на основі REST API та шаблон MVC, що відповідає за поділ відповідальності і управління між окремими компонентами системи. Було досліджено основні переваги використання даного підходу. Було розроблено структуру бази даних та наведено основні сутності системи та зв’язки між ними. В ході розробки даної системи було створено основні API та протестовано їх на предмет коректного функціонування.

В результаті даної роботи було розроблено та протестовано серверну частину клієнт-серверної системи, що базується на основі геолокації з допомогою обраних технологій.

В результаті розрахунку економічного ефекту від можливої комерціалізації даної розробки було обраховано наступні економічні показники: витрати на розроблення мобільної клієнт-серверної системи соціальної мережі - 172 тисяч грн; Абсолютний економічний ефект від впровадження розробки - 10,29 млн грн; Внутрішню дохідність потенційних інвестицій - 32,3 %; Термін окупності потенційних інвестицій - 3,09 років.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Достовірність інформації: питання честі чи рудимент? URL: https://detector.media/infospace/article/38822/2008-06-05-dostovirnist-informatsii-pytannya-chesti-chy-rudyment/ (дата звернення 09.09.2023).

2. Дусанюк О. С., Квєтний Р. Н. *ПІДХІД ДО ПОБУДОВИ СОЦІАЛЬНОЇ МЕРЕЖІ, ЩО БАЗУЄТЬСЯ НА ОСНОВІ ГЕОЛОКАЦІЇ* : Молодь в науці: дослідження, проблеми, перспективи, м. Вінниця, 22-23 червня 2023 р. Вінниця, 2023.

3. Дусанюк О. С., Коцюбинський В. Ю. *ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ТА ОСНОВНИХ АЛГОРИТМІВ СОЦІАЛЬНОЇ МЕРЕЖІ, ЩО БАЗУЄТЬСЯ НА ОСНОВІ ГЕОЛОКАЦІЇ* : LII Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації, м. Вінниця, 21-23 червня 2023 р. Вінниця, 2023.

4. Найпопулярніші соціальні мережі світу. URL: https://espreso.tv/news/2019/10/29/naypopulyarnishi_socialni_merezhi_svitu (дата звернення 10.09.2023).

5. Соціальні мережі як незамінний інструмент комунікації. URL: https://yur-gazeta.com/publications/practice/inshe/socialni-merezhi-yak-nezaminniy-instrument-komunikaciyi.html (дата звернення 10.09.2023).

6. Улічев О. С. Модель та методи поширення інформаційних впливів у соціальних мережах в умовах інформаційного протиборства: дис. на здобуття наукового ступеня кандидата технічних наук: 21.05.01. Київ, 2021. 170 с.

7. Найпопулярніші соціальні мережі в Україні та країнах світу у 2020. URL: https://uaspectr.com/2020/06/23/najpopulyarnishi-sotsialni-merezhi -v-ukrayini-takrayinah-svitu-2020/ (дата звернення 11.09.2023).

8. Ніколайчук М.Д., Озеранський В.С. *Технології створення соціальних мереж* : Тези XLV науково-технічної конференції професорсько-викладацького складу, співробітників та студентів університету, м. Вінниця, 21-23 березня 2018 р. Вінниця, 2018.

9. Соціальні мережі як ефективний інструмент маркетингу в індустрії зустрічей. URL: https://infotour.in.ua/kusina.htm (дата звернення 11.09.2023).

10. What Is Foursquare & How Does It Work?. URL: https://smallbusiness.chron.com/foursquare-work-28728.html (дата звернення 12.09.2023).

11. КОМУНІКАЦІЙНА ПЛАТФОРМА FOURSQUARE ЯК ЗАСІБ ПРОСУВАННЯ ЗАКЛАДІВ РЕСТОРАННОГО ГОСПОДАРСТВА. URL: https://tourlib.net/statti_ukr/zhurahovska.htm (дата звернення 12.09.2023).

12. What Is Foursquare and How Does It Work?. URL: https://techboomers.com/t/what-is-foursquare-how-it-works (дата звернення 12.09.2023).

13. How Foursquare Works. URL: https://computer.howstuffworks.com/internet/social-networking/networks/foursquare.htm (дата звернення 12.09.2023).

14. What Is Foursquare And How Do I Use It?. URL: https://www.businessinsider.com/how-hit-location-based-social-app-foursquare-works-2010-1 (дата звернення 12.09.2023).

15. Все, що вам потрібно знати про Instagram. URL: https://blog.comfy.ua/ua/vse-shcho-vam-potribno-znati-pro-instagram/#yakor_01 (дата звернення 13.09.2023).

16. 41 Instagram Features, Hacks, & Tips Everyone Should Know About. URL: https://blog.hubspot.com/marketing/instagram-features-tricks (дата звернення 13.09.2023).

17. Головне про Instagram Reels. URL: https://sostav.ua/publication/golovnepro-instagram-reels-abo-yak-shche-mozhna-p-dnyati-okhoplennya-prof-lyu89302.html (дата звернення 13.09.2023).

18. ZENLY APP: WHAT YOU NEED TO KNOW. URL: https://www.onlineoptimism.com/blog/zenly-app-what-you-need-to-know/ (дата звернення 13.09.2023).

19. What is Zenly? Reasons Behind the Shutdown and Alternative App. URL: https://blog.jagat.io/hangouts/what-is-zenly/ (дата звернення 13.09.2023).

20. Mark L. Learning Python: Sebastopol : O'Reilly Media, 2009. 1213 с.

21. C.H. Swaroop, James Z. A Byte of Python: Scotts Valley : CreateSpace Independent Publishing Platform, 2017. 162 с.

22. Що таке Python? URL: http://www.plug.org.ua/documentation/aboutpython (дата звернення 14.09.2023).

23. Мова програмування Python. URL: https://edu.cbsystematics.com/ua/blog/python-start-blog (дата звернення 14.09.2023).

24. David F. JavaScript: The Definitive Guide: Sebastopol : O'Reilly and Associates, 2011. 1093 с.

25. Marijn H. Eloquent JavaScript: San Francisco : No Starch Press, 2018. 472 с.

26. Starting your Software Developer Career: Java vs. JavaScript vs. Python. URL: https://medium.com/@learnstuff.io/starting-your-software-developer-career-java-vs-javascript-vs-python-954ed27b3157 (дата звернення 15.09.2023).

27. Your Guide to The Top 15 Backend Languages For 2023. URL: https://www.ishir.com/blog/75047/your-guide-to-the-top-15-backend-languages-for-2023.htm (дата звернення 15.09.2023).

28. JavaScript. URL: https://astwellsoft.com/uk/blog/tehnology/javascript .html (дата звернення 15.09.2023).

29. Joshua B. Effective Java 3rd Edition: Boston : Addison-Wesley Professional, 2017. 416 с.

30. What is Java? Definition, Meaning & Features of Java Platforms. URL: https://www.guru99.com/java-platform.html (дата звернення 16.09.2023).

31. Що таке Java і де вона використовується. URL: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/ (дата звернення 16.09.2023).

32. Pros and Cons of Java | Advantages and Disadvantages of Java. URL: https://data-flair.training/blogs/pros-and-cons-of-java/ (дата звернення 16.09.2023).

33. Advantages and disadvantages of Java. URL: https://www.javatpoint.com/advantages-and-disadvantages-of-java (дата звернення 16.09.2023).

34. Miguel G. Flask Web Development: Sebastopol : O'Reilly Media, 2014. 258 с.

35. Daniel G., Jack S. Mastering Flask Web Development Second Edition: Birmingham : Packt Publishing Ltd., 2018. 327 с.

36. What Is Flask and How Do Developers Use It? A Quick Guide. URL: https://careerfoundry.com/en/blog/web-development/what-is-flask/ (дата звернення 17.09.2023).

37. FLASK PYTHON FRAMEWORK. URL: https://quintagroup.com/cms/python/flask (дата звернення 17.09.2023).

38. Samuel D., Aidas B., Arun R. Django: Web Development with Python: Birmingham : Packt Publishing, 2016. 730 с.

39. ARTICLE ON DJANGO FRAMEWORK IN PYTHON. URL: https://www.linkedin.com/pulse/article-django-framework-python-prasanth-m- (дата звернення 18.09.2023).

40. Flask vs Django: Which Python Web Framework to Use in 2023?. URL: https://hackr.io/blog/flask-vs-django (дата звернення 18.09.2023).

41. Marc D. Creating your MySQL Database: Practical Design Tips and Techniques: Birmingham : Packt Publishing Ltd., 2014. 105 с.

42. Saied T., Hugh W. Learning MySQL: Sebastopol : O'Reilly, 2015. 622 с.

43. Що таке MySQL як і де використовують MySQL. URL: http://ruszura.in.ua/ihry-i-ihrovi-konsoli/scho-take-mysql-yak-i-de-vykorystovuyutmysql.html (дата звернення 19.09.2023).

44. Що таке база даних MySQL. URL: https://uk.education-wiki.com/2686691-what-is-mysql-database (дата звернення 19.09.2023).

45. SQL/Типи даних MySQL. URL: https://uk.wikibooks.org/wiki/SQL/Типи_даних_MySQL (дата звернення 19.09.2023).

46. Regina O. Obe PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database 3rd Edition: Sebastopol : O'Reilly, 2017. 314 с.

47. Silas B. Learn PostgreSQL From Basics 2023, 2022. 100 с.

48. What is PostgreSQL? Introduction, Advantages & Disadvantages. URL: https://www.guru99.com/introduction-postgresql.html (дата звернення 20.09.2023).

49. PostgreSQL vs MySQL: Explore Their 12 Critical Differences. URL: https://kinsta.com/blog/postgresql-vs-mysql/ (дата звернення 20.09.2023).

50. What is PostgreSQL?. URL: https://www.postgresql.org/about/ (дата звернення 20.09.2023).

51. Mark M. REST API Design Rulebook: Sebastopol : O'Reilly, 2011. 114 с.

52. What is REST. URL: https://restfulapi.net (дата звернення 21.09.2023).

53. Що таке RESTful API? URL: https://codeguida.com/post/601 (дата звернення 21.09.2023).

54. Шпаргалка по кодам відповіді стану HTTP. URL: https:// sebweo.com/shpargalka-po-kodam-vidpovidi-stanu-http/ (дата звернення 21.09.2023).

55. Ajit K. Sencha MVC Architecture: Birmingham : Packt Publishing, 2012. 126 с.

56. Everything you need to know about MVC architecture. URL: https:// towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c82 7930b4c1 (дата звернення 29.09.2023).

57. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А (обов'язковий).

Технічне завдання на магістерську кваліфікаційну роботу

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
_____ д.т.н., проф. Олег БІСІКАЛО
«____» _____ 2023 року

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
РОЗРОБКА МОБІЛЬНОЇ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ СОЦІАЛЬНОЇ МЕРЕЖІ, ЩО БАЗУЄТЬСЯ НА ОСНОВІ ГЕОЛОКАЦІЇ
08-31.МКР.006.02.000 ТЗ

Керівник: к.т.н., доц. каф. АІІТ
_____ Володимир КОЦЮБИНСЬКИЙ
«____» _____ 2023 р.
Розробив студент гр. 1АКІТ-22м
_____ Олександр ДУСАНЮК
«____» _____ 2023 р.

Вінниця 2023

1. Назва та галузь застосування

Розробка мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації. У даній роботі розглядається створення мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації. Дана система дасть можливість користувачам спілкуватися, обмінюватися своїми інтересами та подіями зі свого життя, знаходити своїх однодумців, формувати групи із загальними інтересами та відстежувати актуальні події, що відбуваються навколо. Така функціональність дасть змогу користувачам розширювати свій світогляд та брати активну участь у житті свого міста. Крім того, дана система може бути використана компаніями для представлення контекстної реклами з використанням інформації про геолокацію, а також для визначення потенційних споживачів в контексті місця та часу, і більш ефективно надавати цю інформацію постачальникам послуг.

2. Підстави для проведення робіт

Підставою для виконання роботи є наказ №__ по ВНТУ від «__» _____ 2023р., та індивідуальне завдання на МКР, затверджене протоколом №__ засідання кафедри АІІТ від «__» _____ 2023р.

3. Мета та призначення роботи

Метою роботи є покращення комунікації між користувачами та підвищення достовірності інформації, яка публікується, шляхом створення мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації.

4. Джерела розробки

4.1 Mark L. Learning Python / Mark L. – Sebastopol: O'Reilly Media, 2009. – 1213 p. – ISBN 9781449355739.

4.2 Miguel G. Flask Web Development / Miguel G. – Sebastopol: O'Reilly Media, 2014. – 258 p. – ISBN 9781491991732.

4.3 Mark M. REST API Design Rulebook / Mark M. – Sebastopol: O'Reilly, 2011. – 114 p. – ISBN 9781449310509.

5. Показники призначення

5.1 Мінімальні вимоги до техніки:

- ОС: Ubuntu 20.04;
- Процесор: Intel Core i5 3 GHz або вище;
- Об’єм оперативної пам’яті: 8 Gb або більше;
- Жорсткий диск: 120 Gb SATA2 або вище.

5.2 Середовище розробки та запуску PyCharm;

5.3 СУБД MySQL.

6. Економічні показники

До економічних показників входять:

- витрати на розробку – не більше 200 тис. грн;
- абсолютний економічний ефект від впровадження розробки – не менше 10 млн грн;
- внутрішня дохідність потенційних інвестицій – не менше 30%;
- термін окупності потенційних інвестицій – до 3-ох років;

7. Стадії розробки

7.1 Розділ 1 «ЗАГАЛЬНІ ВІДОМОСТІ» має бути виконаний до 30.09.2023.

7.2 Розділ 2 «ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ» має бути виконаний до 04.10.2023.

7.3 Розділ 3 «РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ» має бути виконаний до 12.11.2023.

7.4 Економічний розділ має бути виконаний до 15.11.2023.

8. Порядок контролю та приймання

8.1 Рубіжний контроль провести до 20.11.2023.

8.2 Попередній захист магістерської кваліфікаційної роботи провести до 21.11.2023.

8.3 Захист магістерської кваліфікаційної роботи провести до 14.12.2023.

Розробив студент групи 1АКІТ-22м _____ Олександр ДУСАНЮК

Додаток Б (обов'язковий).

ІЛЮСТРАТИВНА ЧАСТИНА

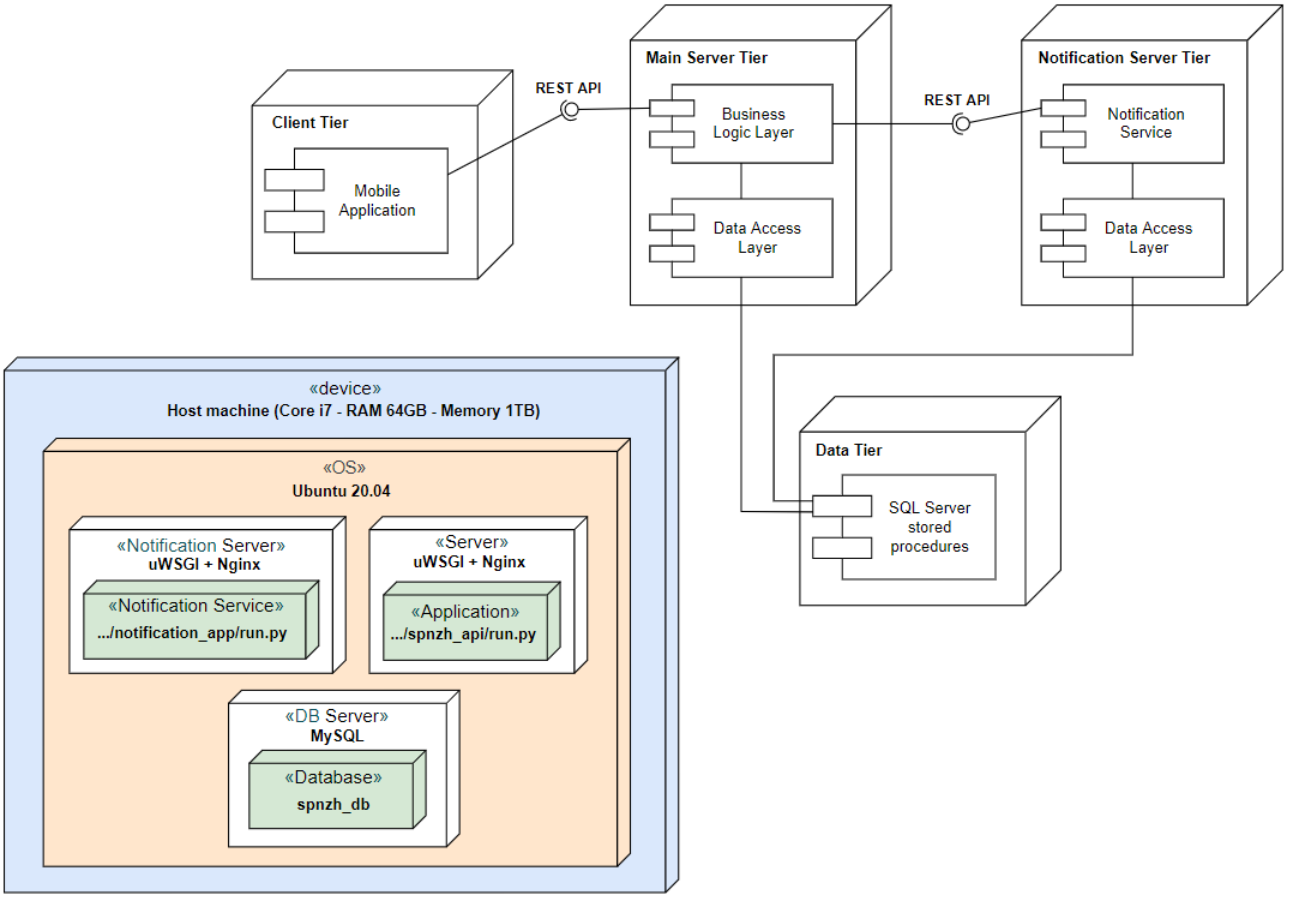

Рисунок Б.1 – Deployment діаграма

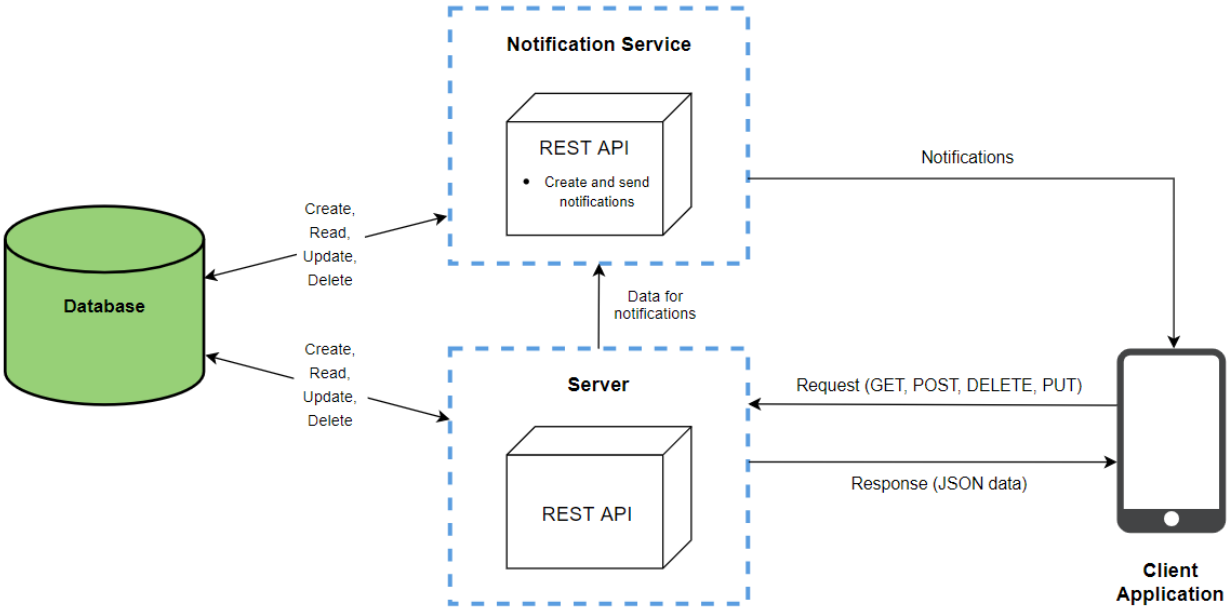

Рисунок Б.2 – Data Flow діаграма

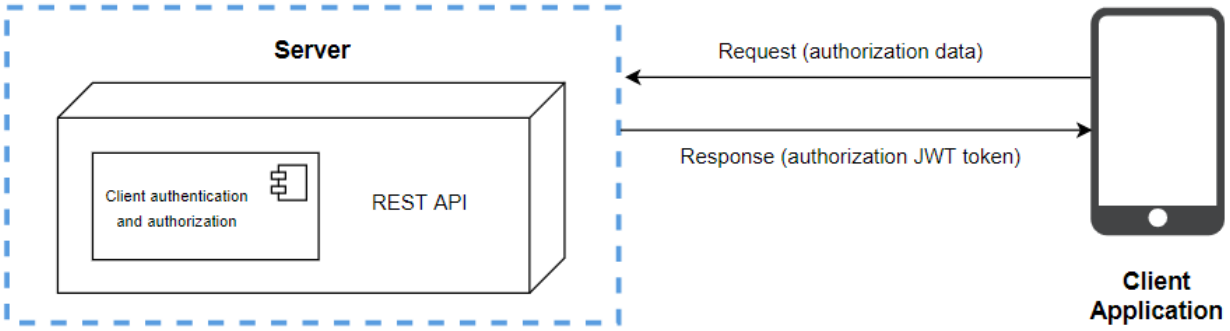

Рисунок Б.3 – Принцип роботи аутентифікації та авторизації

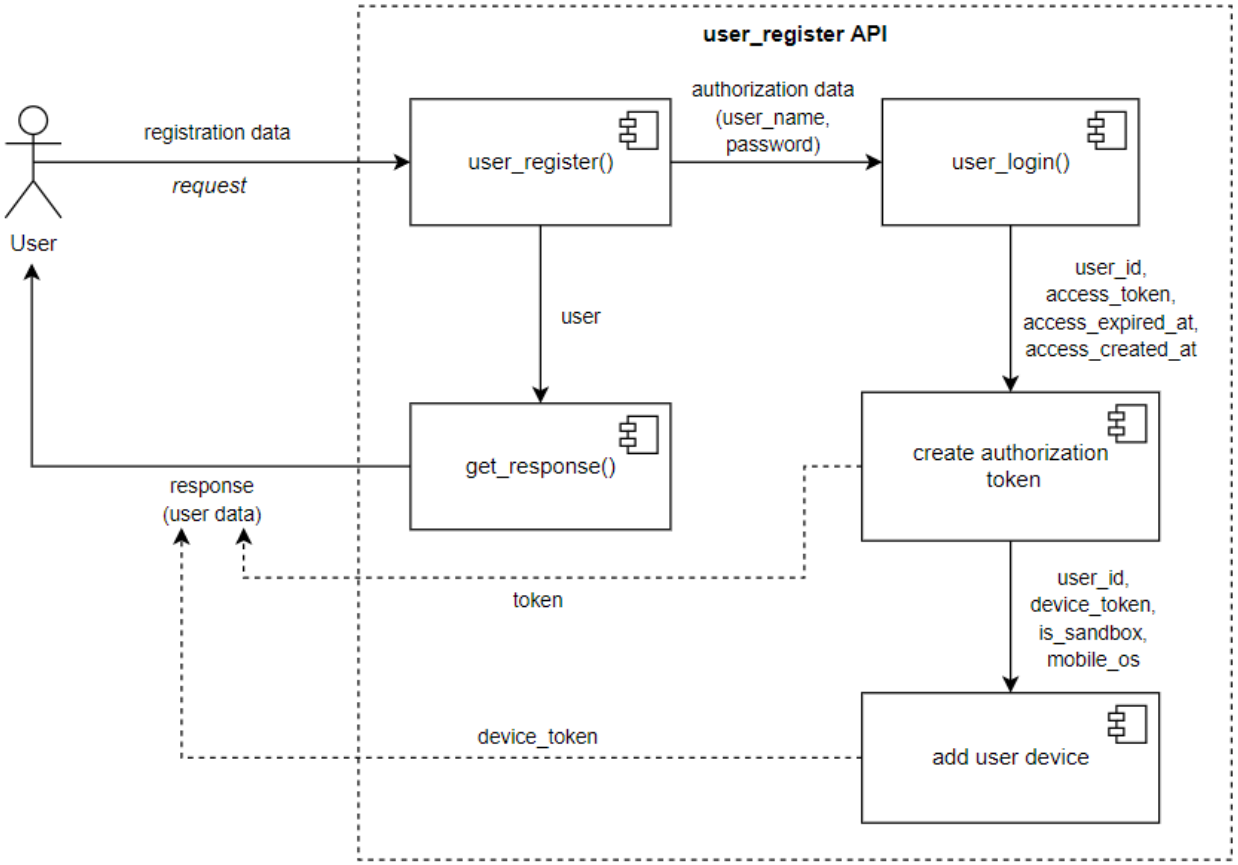

Рисунок Б.4 – Data Flow діаграма роботи API для реєстрації користувача

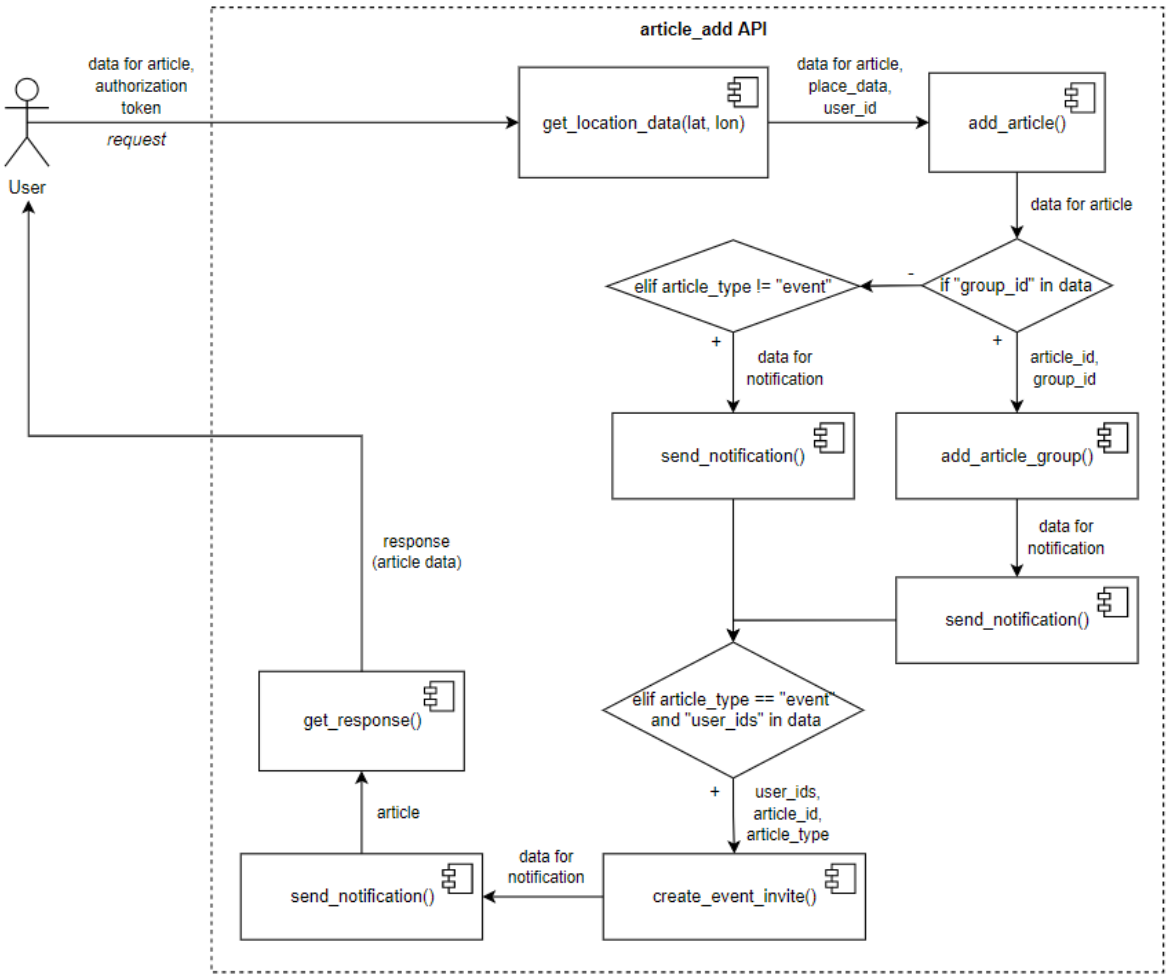

Рисунок Б.5 – Data flow діаграма роботи API для створення поста

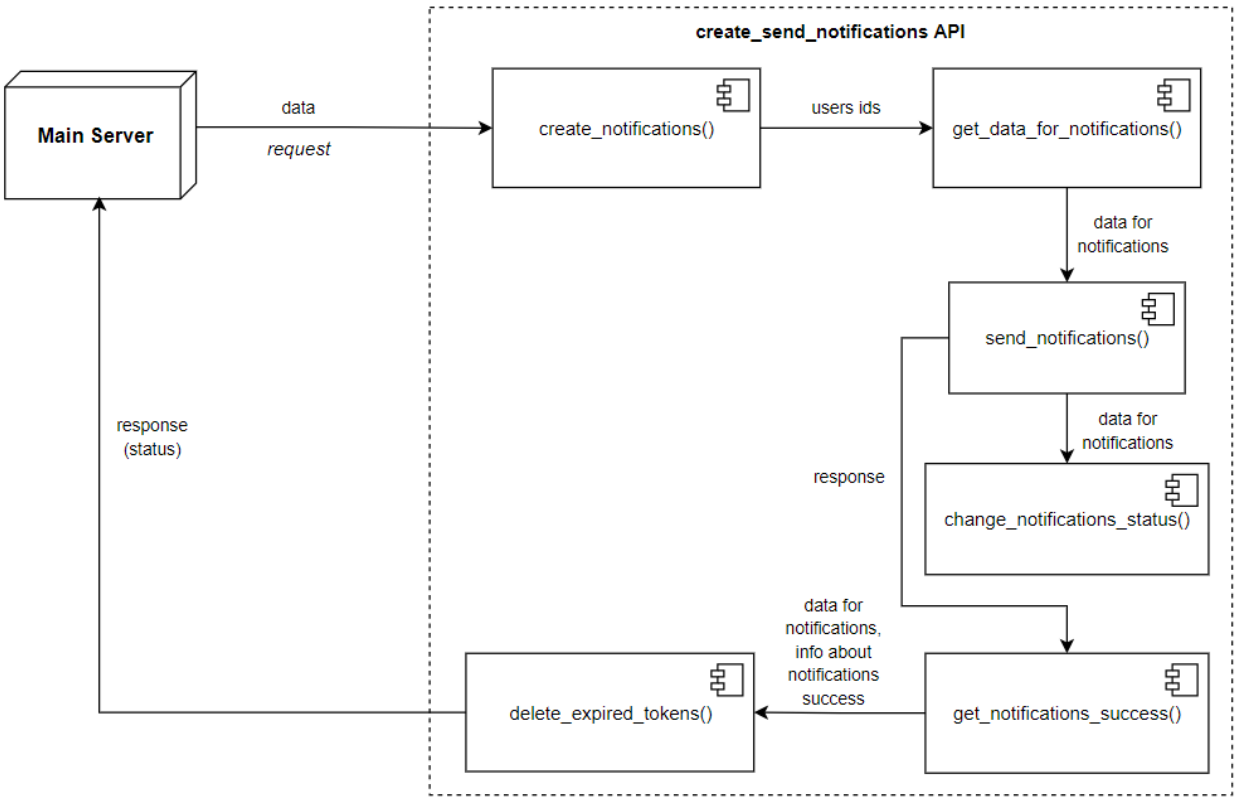

Рисунок Б.6 – Data Flow діаграма роботи API для створення та відправки нотифікацій

Додаток В (обов'язковий).

Лістинг основних сутностей системи основного серверу

Article Model

```
class Article(db.Model):
    __tablename__ = 'articles'
    id = db.Column(db.Integer(), primary_key=True)
    name = db.Column(db.String(200), nullable=False)
    …
    …
    …
    sound_duration = db.Column(db.Integer(), nullable=True, default=None)
    time_delay = db.Column(db.Integer(), nullable=False, default=0)
```

ArticlesGroups Model

```
class ArticlesGroups(db.Model):
    __tablename__ = 'articles_groups'
    …
    …
    …

    __table_args__ = (
        db.PrimaryKeyConstraint(
            article_id, group_id,
        ),
    )
```

Groups Model

```
class Groups(db.Model):
    __tablename__ = 'groups'
    id = db.Column(db.Integer(), primary_key=True)
    name = db.Column(db.String(200), nullable=False, unique=True)
    …
    …
    …
    messaging_type = db.Column(db.String(20), nullable=True, default='')
```

UsersGroups Model

```
class UsersGroups(db.Model):
    __tablename__ = 'users_groups'
    user_id = db.Column(db.Integer())
    group_id = db.Column(db.Integer())
    …
    …
    …
    @property
    def is_admin(self):
        if self.user_role == 'admin':
            return True
        else:
            return False
```

GroupFollower Model

```
class GroupFollower(db.Model):
    __tablename__ = 'groups_followers'
    user_id = db.Column(db.Integer())
    group_id_follow = db.Column(db.Integer())
    …
```

```
        …
        …
        __table_args__ = (
            db.PrimaryKeyConstraint(
                user_id, group_id_follow,
            ),
        )
```

ImageStorage Model

```
    class ImageStorage(db.Model):
        __tablename__ = 'fts_images'
        id = db.Column(db.Integer(), primary_key=True)
        local_url = db.Column(db.String(200), nullable=False, default='')
        …
        …
        …
        datetime = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp())
```

VideoStorage Model

```
    class VideoStorage(db.Model):
        __tablename__ = 'videos'
        id = db.Column(db.Integer(), primary_key=True)
        local_url = db.Column(db.String(200), nullable=False, default='')
        …
        …
        …
        datetime = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp())
```

SoundStorage Model

```
    class SoundStorage(db.Model):
        __tablename__ = 'sounds'
        id = db.Column(db.Integer(), primary_key=True)
        local_url = db.Column(db.String(200), nullable=False, default='')
        …
        …
        …
        datetime = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp())
```

Notification Model

```
    class Notification(db.Model):
        __tablename__ = 'notifications'
        id = db.Column(db.Integer(), primary_key=True)
        from_user_id = db.Column(db.Integer(), nullable=False)
        to_user_id = db.Column(db.Integer(), nullable=False)
        …
        …
        …
        expired_at = db.Column(db.DateTime(), nullable=True, default=None)
        from_obj = db.Column(db.Enum('user', 'group', 'location', 'nearby'),
nullable=True, default=None)
```

NotificationSubscription Model

```
    class NotificationSubscription(db.Model):
        __tablename__ = 'notifications_subscriptions'
        user_id = db.Column(db.Integer())
        obj_id = db.Column(db.Integer())
        obj_type = db.Column(db.Enum('profile', 'group', 'location'), nullable=False,
default='profile')
        …
```

```
        …
        …
        __table_args__ = (
            db.PrimaryKeyConstraint(
                user_id, obj_id, obj_type,
            ),
        )
```

NotificationsTimeVisit Model

```
    class NotificationsTimeVisit(db.Model):
        __tablename__ = 'notifications_time_visit'
        user_id = db.Column(db.Integer(), primary_key=True)
        last_visit_time = db.Column(db.DateTime(), nullable=True, default=None)
```

NotificationViewed Model

```
    class NotificationViewed(db.Model):
        __tablename__ = 'notifications_viewed'
        notification_id = db.Column(db.Integer(), primary_key=True)
        user_id = db.Column(db.Integer(), nullable=False)
        read_time = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp())
```

Location Model

```
    class Location(db.Model):
        __tablename__ = 'locations'
        id = db.Column(db.Integer(), primary_key=True)
        name = db.Column(db.String(200), nullable=False)
        …
        …
        …
        updated_at = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp(), onupdate=db.func.current_timestamp())
```

User Model

```
    class User(db.Model, UserMixin):
        __tablename__ = 'fts_users'
        id = db.Column(db.Integer, primary_key=True)
        user_name = db.Column(db.Unicode(50), nullable=False, server_default=u'',
unique=True)
        first_name = db.Column(db.Unicode(100), nullable=True, server_default=None)
        …
        …
        …
        def check_password_correction(self, attempted_password):
                return  hashlib.md5(attempted_password.encode('utf-8')).hexdigest()  ==
self.password

        def check_email_correction(self, attempted_email):
            return attempted_email == self.email
```

Profile Model

```
    class Profile(db.Model):
        __tablename__ = 'fts_user_profile'
        id = db.Column(db.Integer, primary_key=True)
        user_id = db.Column(db.Integer(), nullable=False, index=True)
        …
        …
        …
        last_lat = db.Column(db.Float(), nullable=True, default=None)
        last_lon = db.Column(db.Float(), nullable=True, default=None)
```

UserDevice Model

```
class UserDevice(db.Model):
    __tablename__ = 'fts_user_devices'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer(), nullable=False)
    device_token = db.Column(db.Unicode(200), nullable=False, server_default=u'',
unique=True)
    …
    …
    …
    datetime = db.Column(db.DateTime(timezone=False),
default=db.func.current_timestamp())
```

UserSettings model

```
class UserSettings(db.Model):
    __tablename__ = 'user_settings'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, nullable=False, index=True)
    …
    …
    …
    measurements = db.Column(db.String(20), nullable=False, default='')
```

UserToken Model

```
class UserToken(db.Model):
    __tablename__ = 'user_token'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer(), nullable=False, index=True)
    …
    …
    …
    access_created_at = db.Column(db.DateTime(timezone=False),
default=db.func.current_timestamp())
```

UserFollower Model

```
class UserFollower(db.Model):
    __tablename__ = 'users_followers'
    id = db.Column(db.Integer(), primary_key=True)
    user_id = db.Column(db.Integer(), nullable=False)
    user_id_follow = db.Column(db.Integer(), nullable=False)
    datetime = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp())
```

Friend Model

```
class Friend(db.Model):
    __tablename__ = 'friends'
    user_id = db.Column(db.Integer())
    friend_user_id = db.Column(db.Integer())
    …
    …
    …
    __table_args__ = (
        db.PrimaryKeyConstraint(
            user_id, friend_user_id,
        ),
    )
```

Додаток Г (обов'язковий).

Лістинг контролерів основного серверу

UserController

```
class UserController:
   user_model = User
   profile_model = Profile
   profile_controller = ProfileController
   image_controller = ImageStorageController
   logged_by = {'common': 0, 'FB': 1, 'AppleID': 2, 'Google': 3}

   def get_by_id(self, id):
       {...}

   def get_with_profile_by_id(self, id):
       {...}

   def get_by_username(self, username):
       {...}

   def get_by_email(self, email):
       {...}

   def get_list(self):
       {...}

   def add_db(self, data):
       {...}

   def upd_by_id(self, id, raw):
       {...}

   def upd_with_profile_by_user_id(self, user_id, raw):
       {...}

   def is_username_exists(self, username):
       {...}

   @staticmethod
   def create_user_uuid(email: str) -> str:
       {...}

   def create_unique_username(self, user):
       {...}

   def username_change(self, user, username):
       {...}

   def user_register(self, data):
       {...}

   def user_login(self, data):
       {...}

   def avatar_delete(self, user_id):
       {...}

   def avatar_add_update(self, user_id, image, folder_to_store, domain=None,
storage_name=None):
       {...}

   def profile_background_delete(self, bg_file, user_id,
background_type='profile_bg'):
       {...}
```

```
    def profile_background_add_update(self, user_id, image, folder_to_store,
background_type='profile_bg', domain=None, storage_name=None, image_extension=None):
        {...}

    def profile_background_map_add_update(self, user_id, mapbox_image,
folder_to_store, background_type='profile_bg', domain=None, storage_name=None,
image_extension=None):
        {...}

        @staticmethod
        def user_logout():
            {...}

        @staticmethod
        def dict_one(obj):
            {...}

        @staticmethod
        def dict_with_profile_one(obj, another_user_id=None):
            {...}

        @staticmethod
        def dict_list(objs):
            {...}

        def get_last_post_coords(self, user_id):
            {...}

        def get_post_count(self, user_id):
            {...}

        def get_is_follow(self, user_id, followed_user_id):
            {...}

        def get_following_count(self, user_id):
            {...}

        def get_followers_count(self, followed_user_id):
            {...}

        def get_friends_count(self, user_id):
            {...}

        def get_is_friends(self, user_id, another_user_id):
            {...}

        def get_is_sent_friend_request(self, user_id, another_user_id):
            {...}
```

ProfileController

```
    class ProfileController:

        db_model = Profile

        def get_by_user_id(self, user_id):
            {...}

    def add_db(self, data):
            {...}

        def upd_by_id(self, id, raw):
            {...}

        def upd_by_user_id(self, user_id, raw):
            {...}

        @staticmethod
        def dict_one(obj):
            {...}
```

UserSettingsController

```
class UserSettingsController:

   db_model = UserSettings

   def get_by_user_id(self, user_id):
       {...}

   def add_db(self, data):
       {...}

   def upd_by_id(self, id, raw):
       {...}

   def upd_by_user_id(self, user_id, raw):
       {...}

   def user_settings_update(self, user_id, data):
       {...}

   @staticmethod
   def dict_one(obj):
       {...}
```

UserDeviceController

```
class UserDeviceController:

   db_model = UserDevice

   def get_by_user_id(self, user_id):
       {...}

   def add_db(self, data):
       {...}

   def upd_by_id(self, id, raw):
       {...}

   def upd_by_user_id(self, user_id, raw):
       {...}

   def update_on_login(self, user_id, device_data):
       {...}

   def del_by_user_id(self, user_id):
       {...}

@staticmethod
   def dict_one(obj):
       {...}
```

UserTokenController

```
class UserTokenController:
   db_model = UserToken

   def add_db(self, data):
       {...}

   def get_by_user_id(self, user_id):
       {...}

   @staticmethod
   def dict_one(obj):
       {...}
```

FriendController

```
class FriendController:
   friend_model = Friend
   user_model = User

   def add_db(self, current_user_id, friend_user_id):
       {...}

   def upd_friend_status(self, friend, status):
       {...}

   def del_db(self, friend):
       {...}

   def add_friendship_request(self, data, current_user_id):
       {...}

   def del_friendship_request(self, data, current_user_id):
       {...}

   def approve_friendship(self, data, current_user_id):
       {...}

   def block_friendship(self, data, current_user_id):
       {...}

   def unblock_friendship(self, data, current_user_id):
       {...}

   def reject_friendship(self, data, current_user_id):
       {...}

   def del_friend(self, data, current_user_id):
       {...}

   def get_friends_list(self, user_id, status):
       {...}

   def get_request_direction(self, current_user_id, user_id, friend_user_id):
       {...}

   @staticmethod
   def dict_one(obj):
       {...}

   @staticmethod
   def dict_list(objs):
       {...}

   @staticmethod
   def list_to_dict(friends_list, user_id=None):
       {...}
```

UserFollowerController

```
class UserFollowerController:
   db_model = UserFollower
   user_db_model = User

   def db_add_follow(self, user_id, user_id_follow):
       {...}

   def add_follow(self, data, user_id):
       {...}

   def del_follow(self, data, user_id):
       {...}
```

```
    def get_followers_list(self, user_id, **params):
        {...}

    def get_followed_list(self, user_id, **params):
        {...}

    @staticmethod
    def dict_one(obj):
        {...}

    @staticmethod
    def dict_list(objs):
        {...}

    @staticmethod
    def query_result_to_dict_list(query_result):
        {...}

    @staticmethod
    def get_final_result(data, user_id=None):
        {...}
```

ArticleController

```
class ArticleController:
    db_model = Article

    def get_by_id(self, id):
        {...}

    def get_list(self, **params):
        {...}

    def get_articles_by_tag(self, **params):
        {...}

    def add_db(self, data, user_id):
        {...}

    def upd_by_id(self, id, raw):
        {...}

    def del_by_id(self, id):
        {...}

    def add_article(self, data, user_id):
        {...}

    def parse_tags(self, text):
        {...}

    def upd_views_counter(self, article):
        {...}

    @staticmethod
    def dict_one(obj, current_user_id=None):
        {...}

    @staticmethod
    def dict_list(objs):
        {...}

    @staticmethod
    def query_result_to_dict_list(query_result, current_user_id=None):
        {...}

    @staticmethod
    def query_search_tags_to_dict_list(query_result, search_tag=None):
        {...}
```

```
    @staticmethod
    def fix_domain(url_link):
        {...}

    @staticmethod
    def fix_domains_in_links(url_links):
        {...}
```

ArticleGroupController

```
class ArticleGroupController:
    db_model = ArticlesGroups

    def add_db(self, data):
        {...}

    def del_by_article_id_group_id(self, article_id, group_id):
        {...}
```

GroupsController

```
class GroupsController:
    groups_model = Groups
    users_groups_model = UsersGroups
    groups_followers_model = GroupFollower
    image_controller = ImageStorageController

    def get_by_id(self, id):
        {...}

    def get_list(self, **params):
        {...}

    def add_db_group(self, data, user_id):
        {...}

    def add_db_creator_group(self, user_id, group_id):
        {...}

    def add_group_and_creator(self, data, user_id):
        {...}

    def upd_by_id(self, id, raw, user_id):
        {...}

    def avatar_delete(self, group_id):
        {...}

     def avatar_add_update(self, group_id, image, folder_to_store, domain=None,
storage_name=None):
        {...}

    def profile_background_delete(self, bg_file, group_id,
background_type='group_profile_bg'):
        {...}

    def profile_background_add_update(self, group_id, image, folder_to_store,
background_type='group_profile_bg', domain=None, storage_name=None):
        {...}

    def get_followers_count(self, group_id):
        {...}

    def get_posts_count(self, group_id):
        {...}

    def get_follow_status(self, group_id, user_id):
        {...}
```

```
    def get_member_info(self, group_id, user_id):
        {...}

    @staticmethod
    def dict_one(obj, user_id=None):
        {...}

    @staticmethod
    def dict_list(objs, user_id=None):
        {...}

    @staticmethod
    def dict_with_creator(objs):
        {...}
```

UsersGroupsController

```
class UsersGroupsController:
    users_groups_model = UsersGroups
    groups_model = Groups

    def get_members_count(self, group_id):
        {...}

    def get_member(self, current_user_id, group_id):
        {...}

    def is_creator_or_admin(self, current_user_id, group_id):
        {...}

    def is_not_creator(self, member):
        {...}

    def is_creator(self, member):
        {...}

    def is_admin(self, member):
        {...}

    def is_not_blocked(self, member):
        {...}

    def add_db_user_group(self, data, user_id):
        {...}

    def del_db(self, obj):
        {...}

    def join_group(self, data, user_id):
        {...}

    def approve_user(self, data, current_user_id):
        {...}

    def reject_user(self, data, current_user_id):
        {..}

    def get_members_list_from_group(self, group_id, current_user_id):
        {...}

    def del_member(self, data, current_user_id):
        {...}

    def remove_join_request(self, data):
        {...}

    def set_admin(self, data, current_user_id):
        {...}
```

```
    def unset_admin(self, data, current_user_id):
        {...}

    def block_member(self, data, current_user_id):
        {...}

    def unblock_member(self, data, current_user_id):
        {...}

    def assign_creator(self, data, current_user_id):
        {...}

    def leave_group(self, data, current_user_id):
        {...}

    @staticmethod
    def dict_one(obj):
        {...}

    @staticmethod
    def dict_list(objs):
        {...}

    @staticmethod
    def get_final_result(data):
        {...}
```

GroupFollower

```
class GroupFollowerController:
    group_model = Groups
    group_follower_model = GroupFollower
    users_groups_model = UsersGroups
    user_model = User

    def add_db(self, user_id, group_id_follow):
        {...}

    def add_group_follower(self, data, user_id):
        {...}

    def del_group_follower(self, data, user_id):
        {...}

    def get_list_groups_follow(self, user_id):
        {...}

    def get_group_followers_list(self, group_id):
        {...}

    def upd_status_group_follower(self, data, current_user_id):
        {...}

    @staticmethod
    def dict_one(obj):
        {...}

    @staticmethod
    def dict_list(objs):
        {...}

    @staticmethod
    def dict_group_followers_list(objs):
        {...}
```

BaseStorageController

```
class BaseStorageController:
    db_model = None
    domain_name = None
```

```
        storage_name = None

        def get_by_id(self, id):
            {...}

        def get_list(self, **params):
            {...}

        def add_db(self, data):
            {...}

        def del_by_id(self, id):
            {...}

        @staticmethod
        def store_file_obj(file_obj, folder_path, filename=None):
            {...}

        @staticmethod
        def delete_file_obj(file_path):
            {...}

        @staticmethod
        def dict_one(obj):
            {...}

        @staticmethod
        def dict_list(objs):
            {...}
```

ImageStorageController

```
    class ImageStorageController(BaseStorageController):
        db_model = ImageStorage
        domain_name = app.config['APP_DOMAIN']
        storage_name = app.config['STORAGE_IMAGES']
        folder_group_name = app.config['FOLDER_GROUP_NAME']

        def __init__(self):
            {...}

        def get_by_file_name(self, file_name, user_id=None, used_for=None):
            {...}

        def get_users_avatar(self, user_id):
            {...}

        def get_users_profile_backgrounds(self, user_id, background_type='profile_bg'):
            {...}

        def get_group_avatar(self, group_id):
            {...}

        def get_group_profile_backgrounds(self, group_id,
background_type='group_profile_bg'):
            {...}

        @staticmethod
        def local_url_to_http_url(local_url, obj_id, obj_type='user', used_for=None):
            {...}
```

VideoStorageController

```
    class VideoStorageController(BaseStorageController):
        db_model = VideoStorage
        domain_name = app.config['APP_DOMAIN']
        storage_name = app.config['STORAGE_IMAGES']

        def __init__(self):
            {...}
```

SoundStorageController

```
class SoundStorageController(BaseStorageController):
   db_model = SoundStorage
   domain_name = app.config['APP_DOMAIN']
   storage_name = app.config['STORAGE_IMAGES']

   def __init__(self):
       {...}
```

NotificationSubscriptionController

```
class NotificationSubscriptionController:
   db_model = NotificationSubscription
   db_model_notification = Notification
   db_model_notification_viewed = NotificationViewed

   def add_db(self, data):
       {...}

   def del_notification_by_id(self, id):
       {...}

   def get_notification_id(self, user_id, type, obj_id, obj_type):
       {...}

   def set_for_profile(self, user_id, profile_user_id):
       {...}

   def unset_for_profile(self, user_id, profile_user_id):
       {...}

   def is_set_for_profile(self, user_id, profile_user_id):
       {...}

   def get_subscribed_user_ids(self, user_id):
       {...}

   def set_for_group(self, user_id, group_id):
       {...}

   def unset_for_group(self, user_id, group_id):
       {...}

   def is_set_for_group(self, user_id, group_id):
       {...}

   def get_subscribed_group_ids(self, user_id):
       {...}

   def set_for_location(self, user_id, location_id):
       {...}

   def unset_for_location(self, user_id, location_id):
       {...}

   def is_set_for_location(self, user_id, location_id):
       {...}

   def get_subscribed_location_ids(self, user_id):
       {...}

   def get_list_notifications_by_user_id(self, user_id):
       {...}

   def add_or_update_notifications_time_visit(self, user_id):
       {...}
```

```
    def get_count_unread_notifications(self, user_id):
        {...}

    def add_notification_viewed(self, user_id, data):
        {...}

    @staticmethod
    def dict_one(obj):
        {...}

    @staticmethod
    def dict_list(objs):
        {...}

    @staticmethod
    def dict_notifications_list(data):
        {...}

    def get_article_by_id(self, article_id):
        {...}
```

LocationController

```
class LocationController:
    db_model = Location

    def add_db(self, data, user_id):
        {...}

    def add_location(self, data, user_id):
        {...}

    def upd_location_by_id(self, data, user_id, location_id):
        {...}

    def del_location_by_id(self, user_id, location_id):
        {...}

    def get_location_by_id(self, location_id):
        {...}

    def get_list_locations(self, **params):
        {...}

    @staticmethod
    def dict_one(obj, user_id=None):
        {...}

    @staticmethod
    def dict_list(objs, user_id=None):
        {...}
```

Додаток Д (обов'язковий).

Лістинг API шару основного серверу

User APIs

```
user_controller = UserController
token_controller = UserTokenController
user_device_controller = UserDeviceController
user_settings_controller = UserSettingsController


@api_user.route('/user/register', methods=['POST'])
def api_user_register():
   {...}


@api_user.route('/user/login', methods=['POST'])
def api_user_login():
   {...}


@api_user.route('/user/logout', methods=['GET'])
@login_required
def api_user_logout():
   {...}


@api_user.route('/user/me/profile', methods=['GET'])
@login_required
def api_user_profile_me():
   {...}


@api_user.route('/user/me/profile/edit', methods=['POST'])
@login_required
def api_user_profile_me_update():
   {...}


@api_user.route('/user/avatar/add', methods=['POST'])
@login_required
def api_user_avatar_add():
  {...}


@api_user.route('/user/profile-bg/add', methods=['POST'])
@login_required
def api_user_profile_bg_add():
   {...}


@api_user.route('/user/avatar/delete', methods=['GET'])
@login_required
def api_user_avatar_delete():
  {...}


@api_user.route('/user/profile-bg/delete', methods=['GET'])
@login_required
def api_user_profile_bg_delete():
   {...}

@api_user.route('/user/<int:user_id>/profile', methods=['GET'])
@login_required
def api_user_profile_get(user_id):
   {...}
```

```
@api_user.route('/user/me/settings', methods=['GET'])
@login_required
def api_user_settings_get():
   {...}


@api_user.route('/user/me/settings', methods=['PUT'])
@login_required
def api_user_settings_update():
   {...}


@api_user.route('/user/profile-bg/map/add', methods=['POST'])
@login_required
def api_user_profile_bg_map_add():
   {...}
```

Article APIs

```
article_controller = ArticleController
article_group_controller = ArticleGroupController
friend_controllers = FriendController
groups_controller = GroupsController


@api_article.route('/article/<int:article_id>', methods=['GET'])
@login_required
def api_article_get(article_id):
   {...}


@api_article.route('/articles', methods=['GET'])
@login_required
def api_articles_list():
   {...}


@api_article.route('/article/add', methods=['POST'])
@login_required
def api_article_add():
   {...}


@api_article.route('/article/edit/<int:article_id>', methods=['POST'])
@login_required
def api_article_update(article_id):
   {...}
```

Group APIs

```
groups_controller = GroupsController
group_follower_controller = GroupFollowerController
users_groups_controller = UsersGroupsController
notification_subscription_controller = NotificationSubscriptionController

followers_controller = UserFollowerController


@api_groups.route('/group/<int:group_id>', methods=['GET'])
@login_required
def api_group_get(group_id):
  {...}


@api_groups.route('/groups', methods=['GET'])
@login_required
def api_groups_list_get():
   {...}
```

```
@api_groups.route('/group/add', methods=['POST'])
@login_required
def api_group_add():
   {...}


@api_groups.route('/group/edit/<int:group_id>', methods=['POST'])
@login_required
def api_group_update(group_id):
   {...}


@api_groups.route('/group/avatar/add', methods=['POST'])
@login_required
def api_group_avatar_add():
   {...}


@api_groups.route('/group/profile-bg/add', methods=['POST'])
@login_required
def api_group_profile_bg_add():
   {...}


@api_groups.route('/group/avatar/delete', methods=['DELETE'])
@login_required
def api_group_avatar_delete():
   {...}


@api_groups.route('/group/profile-bg/delete', methods=['DELETE'])
@login_required
def api_group_profile_bg_delete():
   {...}


@api_groups.route('/group/follower/add', methods=['POST'])
@login_required
def api_group_follower_add():
   {...}


@api_groups.route('/group/follower/delete', methods=['DELETE'])
@login_required
def api_group_follower_delete():
   {...}

@api_groups.route('/groups/follow', methods=['GET'])
@login_required
def api_groups_follow_get_list():
   {...}


@api_groups.route('/group/follower/status_edit', methods=['PUT'])
@login_required
def api_status_group_follower_edit():
   {...}


@api_groups.route('/group/<int:group_id>/followers', methods=['GET'])
@login_required
def api_group_followers_list_get(group_id):
   {...}
```

Followers APIs

```
followers_controller = UserFollowerController
```

```
@api_followers.route('/user/follow', methods=['POST'])
@login_required
def api_user_follow():
   {...}


@api_followers.route('/user/unfollow', methods=['POST'])
@login_required
def api_user_unfollow():
   {...}


@api_followers.route('/user/followers/<int:user_id>', methods=['GET'])
@login_required
def api_followers_get_list(user_id):
   {...}


@api_followers.route('/user/followed/<int:user_id>', methods=['GET'])
@login_required
def api_followed_get_list(user_id):
   {...}
```

Friend APIs

```
friend_data_controller = FriendController


@api_friend.route('/user/friend/add', methods=['POST'])
@login_required
def api_friendship_request_add():
   {...}


@api_friend.route('/user/friend/request/delete', methods=['DELETE'])
@login_required
def api_friendship_request_delete():
   {...}

@api_friend.route('/user/friend/approve', methods=['PUT'])
@login_required
def api_approve_friendship():
   {...}


@api_friend.route('/user/friend/reject', methods=['PUT'])
@login_required
def api_reject_friendship():
   {...}


@api_friend.route('/user/friend/block', methods=['PUT'])
@login_required
def api_block_friendship():
   {...}


@api_friend.route('/user/friend/unblock', methods=['PUT'])
@login_required
def api_unblock_friendship():
   {...}


@api_friend.route('/user/friend/delete', methods=['DELETE'])
@login_required
def api_friend_delete():
   {...}
```

```
@api_friend.route('/user/friends', methods=['GET'])
@login_required
def api_friends_list_get():
   {...}
```

Files_Storage APIs

```
@route_files_storage.route('/images/<path:image_path>', methods=['GET'])
def get_app_images(image_path):
   {...}


@route_files_storage.route('/videos/<path:video_path>', methods=['GET'])
def get_app_videos(video_path):
   {...}


@route_files_storage.route('/sounds/<path:sound_path>', methods=['GET'])
def get_app_sounds(sound_path):
   {...}


@api_common.route('/article/file/upload', methods=['POST'])
@login_required
def api_file_upload():
   {...}
```

Notification APIs

```
notification_subscription_controller = NotificationSubscriptionController

@api_notification.route('/notification/user/set', methods=['POST'])
@login_required
def api_notification_user_set():
   {...}


@api_notification.route('/notification/user/unset', methods=['POST'])
@login_required
def api_notification_user_unset():
   {...}


@api_notification.route('/notification/group/set', methods=['POST'])
@login_required
def api_notification_group_set():
   {...}


@api_notification.route('/notification/group/unset', methods=['DELETE'])
@login_required
def api_notification_group_unset():
   {...}


@api_notification.route('/notification/location/set', methods=['POST'])
@login_required
def api_notification_location_set():
   {...}


@api_notification.route('/notification/location/unset', methods=['DELETE'])
@login_required
def api_notification_location_unset():
   {...}
```

```
@api_notification.route('/notifications', methods=['GET'])
@login_required
def api_get_list_notifications():
   {...}


@api_notification.route('/notifications/unread/count', methods=['GET'])
@login_required
def api_get_count_unread_notifications():
   {...}


@api_notification.route('/notification/viewed/add', methods=['POST'])
@login_required
def api_add_viewed_notification():
   {...}


@api_notification.route('/notification/delete', methods=['DELETE'])
@login_required
def api_delete_notification():
   {...}
```

Location APIs

```
location_data_controller = LocationController

@api_location.route('/location/name', methods=['GET'])
@login_required
def api_get_location_name():
   {...}


@api_location.route('/location/add', methods=['POST'])
@login_required
def api_add_location():
   {...}


@api_location.route('/location/edit/<int:location_id>', methods=['PUT'])
@login_required
def api_edit_location(location_id):
   {...}


@api_location.route('/location/<int:location_id>', methods=['GET'])
@login_required
def api_get_location(location_id):
   {...}


@api_location.route('/location/delete/<int:location_id>', methods=['DELETE'])
@login_required
def api_delete_location(location_id):
   {...}


@api_location.route('/locations', methods=['GET'])
@login_required
def api_get_list_locations():
   {...}
```

Додаток Е (обов'язковий).

Лістинг основних сутностей системи сервісу нотифікацій

Notification Model

```
class Notification(db.Model):
    __tablename__ = 'notifications'
    id = db.Column(db.Integer(), primary_key=True)
    from_user_id = db.Column(db.Integer(), nullable=True, default=None)
    to_user_id = db.Column(db.Integer(), nullable=True, default=None)
    …
    …
    …
    expired_at = db.Column(db.DateTime(), nullable=True, default=None)
    from_obj = db.Column(db.Enum('user', 'group', 'location', 'nearby'),
nullable=True, default=None)
```

UserDevice Model

```
class UserDevice(db.Model):
    __tablename__ = 'fts_user_devices'
    id = db.Column(db.Integer(), primary_key=True)
    user_id = db.Column(db.Integer(), nullable=False)
    device_token = db.Column(db.String(200), nullable=False)
    …
    …
    …
    datetime = db.Column(db.DateTime(), nullable=False,
default=db.func.current_timestamp())
```

Додаток Ж (обов'язковий).

Лістинг шару контролера сервісу нотифікацій

NotificationController

```
class NotificationController:
   db_model_notification = Notification
   db_model_user_device = UserDevice
   domain_name = app.config['APP_DOMAIN']
   storage_images_name = app.config['STORAGE_IMAGES']
   folder_group_name = app.config['FOLDER_GROUP_NAME']


@staticmethod
def get_followers_of_user(user_id):
   {...}


@staticmethod
def get_following_of_user(user_id):
   {...}


@staticmethod
def get_friends_of_user(user_id):
   {...}


@staticmethod
def get_followers_of_group(group_id):
   {...}


@staticmethod
def get_members_of_group(group_id, user_id):
   {...}


@staticmethod
def get_users_around(post_lat, post_lon):
   {...}


@staticmethod
def get_location_followers(post_lat, post_lon, user_id):
   {...}


def get_avatar(self, obj_id, used_for):
   {...}


@staticmethod
def  add_post_notification_db(from_user_id,  type,  name,  content,  status,  obj_id,
obj_type, expired_at, *users_ids):
   {...}


@staticmethod
def create_notifications_for_user_add_post(type_event, data):
   {...}

@staticmethod
def create_notifications_for_group_add_post(type_event, data):
   {...}
```

```
@staticmethod
def create_notifications(data):
   {...}


@staticmethod
def get_data_all_messages(user_ids):
   {...}


@staticmethod
def change_notifications_status(data):
   {...}


@staticmethod
def del_expired_tokens(data, response):
   {...}


@staticmethod
def get_list_ids(sql_result):
   {...}


@staticmethod
def get_list_users_ids_and_locations_names(sql_result):
   {...}


@staticmethod
def get_result(all_responses):
   {...}


def send_all_notifications(data):
   {...}
```

Додаток К (обов'язковий).

Лістинг API шару сервісу нотифікацій

Notification API

```
from notifications_app.controllers import NotificationController


notification_controller = NotificationController


def func_notifications_thread(**data):
   user_ids = notification_controller.create_notifications(data)
   if user_ids:
       data_for_messages = notification_controller.get_data_all_messages(user_ids)
       response = send_all_notifications(data_for_messages)
       notification_controller.change_notifications_status(data_for_messages)
       result = notification_controller.get_result(response)
       notification_controller.del_expired_tokens(data_for_messages, result)


@notifications_blueprint.route('/notifications/add', methods=['POST'])
def create_send_notifications():
   data = request.get_json()
   with app.app_context():
                   notifications_thread  =  Thread(target=func_notifications_thread,
kwargs=data)
       notifications_thread.start()
   return {"status": True}
```

Додаток Л (обов'язковий).

Лістинг методів тестування API

```
      def test_user_register():
         response = requests.post(url=f'{BASE_URL}/user/register', json=data_user)
         data = response.json()['data']
         headers['x-access-tokens'] = data['token']
         logging.error(response.text)
         assert response.status_code == 200

      def test_user_login():
         response = requests.post(url=f'{BASE_URL}/user/login', json=data_for_login)
         if 'error' in response.json():
             logging.error(response.text)
         else:
             data = response.json()['data']
             headers['x-access-tokens'] = data['token']
         assert response.status_code == 200

      def test_user_logout():
         response = requests.get(url=f'{BASE_URL}/user/logout', headers=headers)
         logging.error(response.text)
         assert response.status_code == 200

      def test_article_add():
          response = requests.post(url=f'{BASE_URL}/article/add', json=data_article_add,
headers=headers)
         logging.error(response.text)
         assert response.status_code == 200

      def test_group_add():
            response = requests.post(url=f'{BASE_URL}/group/add', json=data_group_add,
headers=headers)
         logging.error(response.text)
         assert response.status_code == 200

      def test_add_location():
         response = requests.post(url=f'{BASE_URL}/location/add', headers=headers,
json=data_add_location)
         logging.error(response.text)
         assert response.status_code == 200
```

Додаток М (обов'язковий).

Протокол перевірки

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільної клієнт-серверної системи соціальної мережі, що базується на основі геолокації.

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації

Показники звіту подібності Unicheck

Оригінальність 96.7% Схожість 3.3%

Аналіз звіту подібності (відмітити потрібне):

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

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

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

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ
(підпис) (Ім'я, ПРІЗВИЩЕ)

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

Автор роботи _____ Олександр ДУСАНЮК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Володимир КОЦЮБИНСЬКИЙ
(підпис) (Ім'я, ПРІЗВИЩЕ)