МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ СОЦІАЛЬНОЇ МЕРЕЖІ ЗАСОБАМИ PYTHON ТА JAVA SCRIPT»**

Виконала: студентка 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
О.М. Лутаєва
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Лутаєвій Ользі Миколаївні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку соціальної мережі засобами Python та Java Script

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування інформаційної системи.
4. Особливості реалізації та результати тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 01.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	03.05.2023	
2.	Збір вихідних даних.	22.05.2023	
3.	Обробка методичних та теоретичних	19.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2023	
5.	Розробка третього розділу.	30.10.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ О.М. Лутаєва
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку соціальної мережі засобами Python та Java Script»: 61 с., 22 рис., 3 табл., 17 джерел, 3 додатки.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ІНТЕРФЕЙС, СОЦІАЛЬНА МЕРЕЖА, JAVASCRIPT, LIAISE, PYTHON, SQL.

Об’єкт дослідження – процес розробки вебзастосунку соціальної мережі.

Мета роботи: розробка вебзастосунку соціальної мережі засобами Python та JavaScript.

Метод дослідження – методи об’єктно-орієнтованого програмування, методи програмної інженерії, системний аналіз.

Проведено огляд схожих проєктів, а також опис інструментів для розробки інформаційних систем, аналіз використаних технологій. Розроблено структуру програми та діаграми, що ілюструють класи, послідовності та прецеденти застосунку. Проведено проєктування бази даних і проілюстровано ER-діаграмою.

Реалізовано клієнтську і серверну частину, з подальшим тестуванням для завершення процесу розробки. Клієнтську частину вебзастосунку було розроблено з використанням HTML, CSS, вебфреймворку Django та JavaScript. Серверна частина розроблена з використанням Django та бази даних SQLite. Проведено ручне і автоматичне тестування інформаційної системи.

SUMMARY

Master’s qualifying paper «Development of the Social Network Web Application using Python and Java Script»: 61 pages, 22 figures, 3 tables, 17 references, 3 supplements.

DATABASE, WEB APPLICATION, INTERFACE, SOCIAL NETWORK, JAVASCRIPT, LIAISE, PYTHON, SQL.

Object of the study is the process of developing a social network web application.

Aim of the study: development of the social network web application using Python and JavaScript.

Methods of research are methods of object-oriented programming, software engineering methods, system analysis.

An overview of similar projects was carried out, as well as a description of tools for the development of information systems, an analysis of the technologies used. Developed application structure and diagrams illustrating application classes, sequences, and precedents. The design of the database was carried out and illustrated with an ER diagram.

The client and server parts have been implemented, with further testing to complete the development process. The client part of the web application was developed using HTML, CSS, Django web framework and JavaScript. The server part is developed using Django and SQLite database. Manual and automatic testing of the information system was carried out.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі, що швидко розвивається, ефективна комунікація стала першорядною, формуючи спосіб взаємодії та співпраці людей і компаній в онлайн-сфері. Сучасний розвиток технологій сприяє створенню вебпрограм різних видів.

Метою роботи є розробка доступного у використанні вебзастосунку соціальної мережі «Liaise» з використанням Python та JavaScript. Для досягнення поставленої мети визначено такі завдання дослідження: проаналізувати існуючі вебзастосунки для комунікації; розробити вебзастосунок з використанням Python та JavaScript; протестувати розроблену соціальну мережу.

Об’єкт дослідження – процес розробки вебзастосунку соціальної мережі.

Методами дослідження є методи об’єктно-орієнтованого програмування, методи програмної інженерії, системний аналіз.

Предметом дослідження є Python та JavaScript.

У ході вирішення поставлених завдань було отримано такі результати: проаналізовано ряд подібних існуючих проєктів, спроєктовано структуру застосунку, описано основні принципи роботи програмного забезпечення, реалізовано клієнтську та серверну частину і протестовано розроблену соціальну мережу.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань (17 найменувань) та трьох додатків.

1 ОГЛЯД МЕТОДІВ ТА ЗАСОБІВ ПРОЄКТУВАННЯ

1.1 Передумови та загальний опис проєкту

Сьогоднішній світ, безсумнівно, сформований подіями та інноваціями вчорашнього дня. Останні десятиліття знаходяться під значним впливом інформаційних технологій. За відносно короткий проміжок часу скромна іконка застосунку «витіснила» пошту, з традиційним процесом листування та численні інші засоби передачі інформації, які тепер увійшли в історію. Зараз для більшості людей є дещо незвичним отримувати інформацію, наприклад, поштою, адже це займає, за сьогоднішніми мірками, неймовірно довгий період часу. Людство прагнуло оптимізувати продуктивність, і тому використало всесвітню мережу як основний засіб передачі інформації. І це було рішення, яке принесло значний розвиток технологій.

З кожним роком час, необхідний для виконання завдань, пов’язаних із пошуком, отриманням і передачею інформації, скорочується, тоді як доступ до інформації постійно розширюється. Для міжособистісної комунікації або комунікації між групою людей достатньо простого натискання на піктограму цифрової програми, щоб надіслати повідомлення, процес який займає лічені секунди, проте можна досягти значного результату і проінформувати велику групу людей за досить короткий проміжок часу.

Вже існує екосистема різноманітних вебзастосунків, кожен з яких розроблений для певного аспекту спілкування.

Наприклад, універсальна платформа «Slack», спочатку розроблена для професійних та організаційних комунікацій. Однак її адаптивність і універсальність зробили її не лише інструментом для бізнесу, а й платформою для зручного спілкування. Подібним чином програма для чату «Discord», яка спочатку була прийнята спільнотою гравців комп’ютерних ігор за безперебійну голосову та текстову комунікацію, вийшла за межі свого коріння і тепер

резонує в багатьох сферах, включаючи освітні та професійні мережі.

Кожного дня база користувачів соціальних мереж, комунікаційних застосунків і цифрових сервісів продовжує збільшуватися, роблячи їх лідерами у категорії «Apps» у службах цифрового розповсюдження.

Використовуючи потужність сучасних вебтехнологій, таких як Python, JavaScript і SQL, ми маємо можливість створювати складні вебпрограми, які сприяють спілкування та співпрацю між користувачами. Тож, для вирішення задачі, пропонується саме цей інструментарій для створення вебзастосунку «Liaise» [1].

Python є популярною мовою програмування, яка використовується в широкому спектрі розробок, починаючи від вебзастосунків і закінчуючи аналізом даних. Тим часом JavaScript займає центральне місце, коли справа доходить до створення динамічних та інтерактивних вебсторінок, а SQL служить основою для ефективного керування базами даних.

Використовуючи переваги цих технологій, можна створити потужну комунікаційну платформу, з низкою функцій, таких як обмін повідомленнями в реальному часі, перегляд профілю тощо. Вебпроєкти такого роду можуть бути чудовим інструментом для компаній, щоб оптимізувати внутрішні комунікації, або для окремих людей, щоб зв’язуватися з іншими, хто має подібні інтереси.

Таким чином, метою цієї роботи є створення компактної, але високоефективної соціальної мережі «Liaise», створеної для спілкування між її користувачами. Ця мережа намагатиметься охопити суть сучасної комунікації, забезпечуючи продуктивний обмін інформацією у світі, який спирається на принципи зв’язку та взаємодії. Така платформа розроблена переважно на основі сучасного вебфреймворку Django з використанням JavaScript для покращення ефективності розробки деяких частин створюваного програмного забезпечення. Клієнтська частина використовує стандартні засоби, такі як HTML та CSS, з використанням мови Python, для відображення динамічного вмісту створених вебсторінок [2].

1.2 Огляд подібних проєктів

Сучасні служби цифрового розповсюдження пропонують широкий спектр програм, соціальних мереж, застосунків комунікації. Середньостатистична людина використовує приблизно чотири застосунки для своїх комунікаційних потреб. Незважаючи на чисельність застосунків такого типу ми маємо список найбільш популярних застосунків комунікації для особистого та комерційного спілкування. Досить довгий проміжок часу такі застосунки як «Facebook», «Instagram», «X» (до липня 2023 – «Twitter») перебувають у топі серед програм з поміткою «комунікація».

Нижче розглянуто детальніше кожну із названих платформ.

«Facebook» полегшує спілкування за допомогою персоналізованих профілів, оновлення статусу, обміну фотографіями та відео, а також можливості створювати і приєднуватися до різних груп і спільнот. Крім того, завдяки різноманітним варіантам обміну повідомленнями, включаючи «Messenger» і «WhatsApp», соціальна мережа надає комплексний пакет для особистої та професійної взаємодії.

Ключові елементи, які використовувалися на ранніх етапах розробки «Facebook»:

– мови програмування (спочатку мережа створювалася з використанням PHP для серверних сценаріїв і взаємодії з базами даних; на стороні клієнта для інтерфейсу користувача та інтерактивності використовувалися HTML, CSS і JavaScript);
– управління базами даних (MySQL була системою керування базами даних, яка використовувалася для зберігання та отримання даних користувача) [3].

Однією з сильних сторін «Facebook» є його здатність пропонувати безліч функцій, які задовольняють різноманітні інтереси та вимоги його користувачів. Здатність платформи адаптуватися до мінливих потреб користувачів гарантує, що вона ще довго залишатиметься популярною соціальною мережею [4].

«Instagram», у свою чергу, є своєрідною платформою для візуального оповідання, де користувачі діляться фотографіями та відео зі своїми підписниками. Функціонал даної соціальної мережі дещо простіший, ніж у «Facebook», проте можливо саме це і робить її популярнішою серед молоді. Застосунок має декілька основних екранів. Розглянемо декілька екранів та функцій «Instagram». Головний екран або «Стрічка» – це місце, де користувачі стикаються з підібраним потоком вмісту з облікових записів, за якими вони стежать. Також однією з головних функцій «Instagram» є функція «Історії», яка дозволяє користувачам ділитися тимчасовим вмістом, який зникає через 24 години. Цей формат заохочує спонтанність і обмін у реальному часі. Сторінка «Дослідження» є ще одним ключовим компонентом, де користувачі можуть відкривати новий контент, адаптований до їхніх інтересів за допомогою персоналізованого алгоритму. «Instagram» використовує машинне навчання для аналізу поведінки та вподобань користувачів, підбираючи різноманітні фото та відео з облікових записів, на які вони можуть не стежити, але, ймовірно, знайдуть їх привабливими. Функція прямого обміну повідомленнями, відома як «Direct», полегшує приватні розмови між користувачами. Ця функція дозволяє спілкуватися один на один або в групі за допомогою тексту, фотографій і відео, додаючи платформі більш особистий вимір. «Instagram» також підтримує голосові та відеодзвінки, що підсилює його роль як комплексного інструменту спілкування. Соціальна мережа дозволяє користувачам ділитися контентом, спілкуватися і стежити за людьми та брендами. Він відомий своєю яскравою спільнотою, що робить його невід’ємною частиною сучасної комунікації, особливо для компаній та впливових осіб [5].

Розробка «Instagram» передбачала використання різних інструментів і мов програмування. Спочатку соціальна мережа була запущена як програма для операційної системи iOS, тому ймовірно, використовувалися для розробки мови програмування Objective-C і пізніше Swift. Мова Java використана для розробки версії програми для операційної системи Android. Серверна частина «Instagram» була здебільшого побудована на Python. Django, високорівневий вебфреймворк

Python, як відомо, використовувався для певних аспектів розробки серверної частини. Також відбулося використання PostgreSQL, як основної системи керування реляційною базою даних (RDBMS) для зберігання та керування даними. Вебтехнології JavaScript, HTML, CSS використані для розробки вебкомпонентів платформи. А React.js для створення динамічних інтерфейсів користувача [6].

«X» (або «Twitter» до липня 2023) є платформою мікроблогів та займає важливе місце у спілкуванні в режимі реального часу та розповсюдженні екстрених новин. Одна з провідних комунікаційних платформ, що дозволяє користувачам брати участь у публічних бесідах, ділитися своїми думками та спілкуватися з глобальною аудиторією.

Користувачі можуть стежити за іншими обліковими записами, щоб бачити їхні твіти в їхній часовій шкалі. Так само користувачі можуть мати підписників, які бачать їхні твіти. Ця асиметрична система підписок дозволяє користувачам керувати своїм обліковим запис, вибираючи, чиї твіти вони хочуть бачити. Хештеги використовуються для класифікації твітів і надання їм доступності для широкої аудиторії. Користувачі можуть натискати або шукати хештеги, щоб знайти твіти, пов’язані з певними темами чи подіями. Користувачі можуть взаємодіяти з твітами, коментуючи їх або ставлячи лайки. Також користувачі можуть згадувати інших користувачів системи у своїх твітах, використовуючи символ «@», після якого йде ім’я користувача. Це створює сповіщення для згаданого користувача та полегшує комунікацію. «Twitter» є інструментом для спілкування в реальному часі, поширення новин, соціальної активності та маркетингу. Його простота та стислість роблять його унікальною та впливовою платформою у сфері соціальних мереж [7].

Технологічний стек, який використовувався для початкової розробки «Twitter», включав кілька інструментів і мов програмування. Серверна частина мережі спочатку була створена з використанням Ruby on Rails, фреймворку вебзастосунків, написаного на Ruby. Rails забезпечував зручне та швидке середовище розробки. Системою керування базою даних, яка

використовувалася для зберігання даних «Twitter», була MySQL. Для інтерфейсу Twitter використовував JavaScript для сценаріїв на стороні клієнта, а AJAX (асинхронний JavaScript і XML) використовувався для забезпечення динамічних і асинхронних оновлень інтерфейсу користувача. Це дозволило користувачам отримувати оновлення в реальному часі без необхідності оновлювати всю сторінку [8].

Ці комунікаційні платформи не тільки сприяють особистим зв'язкам, але й служать функціями, які впливають на розвиток соціальної культури, формуючи те, як ми спілкуємося та взаємодіємо один з одним у цифрову епоху. Вони стали невід'ємними компонентами сучасної комунікації для компаній, впливових людей і широкої громадськості. Їх постійні інновації та здатність до адаптації гарантують їхню актуальність у цифровому середовищі, що постійно розвивається. По мірі того, як ми рухаємося вперед, ці платформи, ймовірно, залишаться на передовій, формуючи майбутнє того, як ми спілкуємося та взаємодіємо в епоху цифрових технологій.

1.3 Засоби розробки інформаційної системи

Розробка вебзастосунку «Liaise» поділяється на декілька етапів, кожен з яких відіграє вирішальну роль у створенні функціонального та надійного програмного забезпечення.

Першим етапом є збір та аналіз вимог, на якому визначаються цілі, завдання та етапи розробки. Також обираються методи розробки та інструменти. Цей початковий етап служить основою для всього проєкту. Ці рішення мають ключове значення для формування процесу розвитку застосунку.

Наступною стадією розробки є розробка дизайну програми. Дизайн поділяється на два основні аспекти: дизайн інтерфейсу, а також дизайн структури вебзастосунку, тобто його проєктування.

Процес розробки передбачає комплексний підхід до створення структури вебзастосунку. Це передбачає розробку архітектури, яка підтримуватиме функціональність програми. Етап структурного проєктування охоплює створення бази даних програми, серверних компонентів і загальної архітектури системи. Одним із аспектів проєктування структури вебзастосунку є використання діаграм уніфікованої мови моделювання (UML). Діаграми UML забезпечують стандартизоване, візуальне представлення архітектури системи, допомагаючи розробникам та зацікавленим сторонам краще зрозуміти зв’язки та взаємодію між різними компонентами. Ці діаграми включають:

– діаграми класів;
– діаграми послідовності;
– діаграми розгортання;
– діаграми компонентів [9].

Зв’язок між дизайном інтерфейсу та структурним дизайном має велике значення для загального успіху вебзастосунку. У той час як дизайн UX та інтерфейсу користувача забезпечує позитивну та привабливу взаємодію з користувачем, структурний дизайн гарантує, що програма є зручною для обслуговування та роботи.

Третій етап розробки це, власне, реалізація застосунку, яка містить у собі створення візуальної частини із попереднього етапу, реалізацію серверної частини, яка включає практичну реалізацію програми та розробку бази даних. Цей етап охоплює трансформацію концептуалізованих візуальних елементів попереднього етапу в матеріальні інтерактивні компоненти. Під час цієї фази впровадження фокус зміщується в бік ретельного написання коду, який виступає основою програмного забезпечення.

Це часто передбачає використання таких технологій, як JavaScript, універсальна мова програмування, яка широко використовується в сучасній веброзробці для покращення взаємодії з користувачем за допомогою динамічних і адаптивних елементів. JavaScript, як мова сценаріїв на стороні клієнта, має важливу роль у забезпеченні оновлень у реальному часі,

асинхронному спілкуванні та інтерактивних функціях інтерфейсу користувача. Це дає змогу розробникам створювати динамічні вебзастосунки, які не вимагають повного перезавантаження сторінки. Розгалужена екосистема бібліотек і фреймворків JavaScript, таких як React, Angular або Vue.js, додатково оптимізує процес розробки, забезпечуючи багаторазові компоненти та ефективні способи керування станом застосунку [10].

Водночас на стороні сервера такі технології, як Django, роблять значний внесок у процес розробки. Django – це високорівневий вебфреймворк Python, який забезпечує швидку розробку та чистий прагматичний дизайн. Його функції включають систему ORM (Object-Relational Mapping), яка спрощує взаємодію з базою даних, і вбудовану адміністративну панель, що зменшує зусилля, необхідні для розробки серверної частини. Фреймворк дотримується принципу «Не повторюйся» (DRY), покращуючи зручність обслуговування коду [11].

SQL (мова структурованих запитів) використовується для взаємодії з реляційними базами даних, ефективного керування зберіганням і пошуком даних. Це дозволяє розробникам визначати, маніпулювати та запитувати дані з точністю, забезпечуючи цілісність і узгодженість рівня даних програми.

Крім того, інтеграція JavaScript на інтерфейсі та Django на сервері формує надійний повноцінний підхід до розробки. Ця комбінація використовує сильні сторони обох технологій, забезпечуючи безперебійний і ефективний робочий процес для розробників. Спільне використання JavaScript і Django дозволяє створювати сучасні багатофункціональні вебпрограми з адаптивним інтерфейсом користувача та масштабованою серверною архітектурою.

Також, етап реалізації застосунку охоплює модульне тестування, яке передбачає виявлення та усунення проблем, які виникають під час процесу кодування.

Заключним етапом у нашій розробці є тестування, протягом якого, радше за все, час від часу доведеться повертатися до попереднього етапу реалізації застосунку задля усунення помилок, які виникли при розробці та тестуванні.

Важливо розуміти, що ці етапи можуть бути не суворо послідовними й часто можуть накладатися або повторюватися, щоб забезпечити відповідність кінцевого продукту найвищим стандартам якості та очікуванням користувачів.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Багато галузей промисловості мають певні законодавчі та нормативні вимоги щодо обробки даних, конфіденційності та безпеки. Проєктування інформаційної системи забезпечує відповідність системи цим нормам, щоб уникнути проблем. Це важливий крок у загальному процесі розробки, що впливає на успіх і довговічність впровадженої інформаційної системи. Цей розділ містить інформацію про структуру вебзастосунку, основні принципи роботи застосунку у вигляді діаграм, а також проєктування структури бази даних.

2.1 Проєктування структури застосунку

Перш ніж приступати до роботи, пов’язаної із вебзастосунком, необхідно ретельно проаналізувати його структуру. Структура сайту – це схема розташування його сторінок, категорій, підкатегорій і товарів. Це своєрідний план, в якому прослідковується логічний зв’язок між сторінками. З технічної точки зору навігація ресурсу являє собою набір URL, що розташовані в певній послідовності. Вона нерозривно пов’язана з семантичним ядром. А саме воно визначає, які папки та документи мають бути на сайті [12].

Тонкощі реалізації структури сайту відіграють важливу роль у впливі на рейтинг вебсайту в пошукових системах і формуванні сприйняття користувачів. Повне розуміння добре продуманої структури дає змогу зрозуміти нюанси того, чому два вебсайти, нібито схожі за різними параметрами, можуть демонструвати суттєві відмінності в своїх показниках ефективності.

Структурна основа вебсайту охоплює його архітектуру, навігацію та загальну організацію вмісту. Кожен елемент сприяє загальному досвіду користувача та може суттєво вплинути на те, як пошукові системи оцінюють

сайт. Адже вони використовують алгоритми, які аналізують організацію вмісту, ієрархію сторінок і зручність навігації. Добре структурований вебсайт, швидше за все, вважатиметься пошуковими системами релевантним і цінним, що призведе до підвищення рейтингу та збільшення видимості. А послідовна та інтуїтивно зрозуміла структура посприяє залученню користувачів.

Існує декілька видів структури сайтів. Якого виду може бути схема, залежить від мети сайту. Виходячи з цього обирають найбільш відповідний тип структури. У випадку даного проєкту соціальної мережі найбільше підійде гібридна структура сайту, яка об’єднує різні типи організаційних форматів, пропонуючи користувачам ефективний засіб швидкого та легкого пошуку необхідної інформації [12].

Отже, при розробці структури вебзастосунку «Liaise», було поставлено не менш важливе завдання, спроєктувати просту та зручну для користувача структуру застосунку гібридного вигляду (див. рис. 2.1).

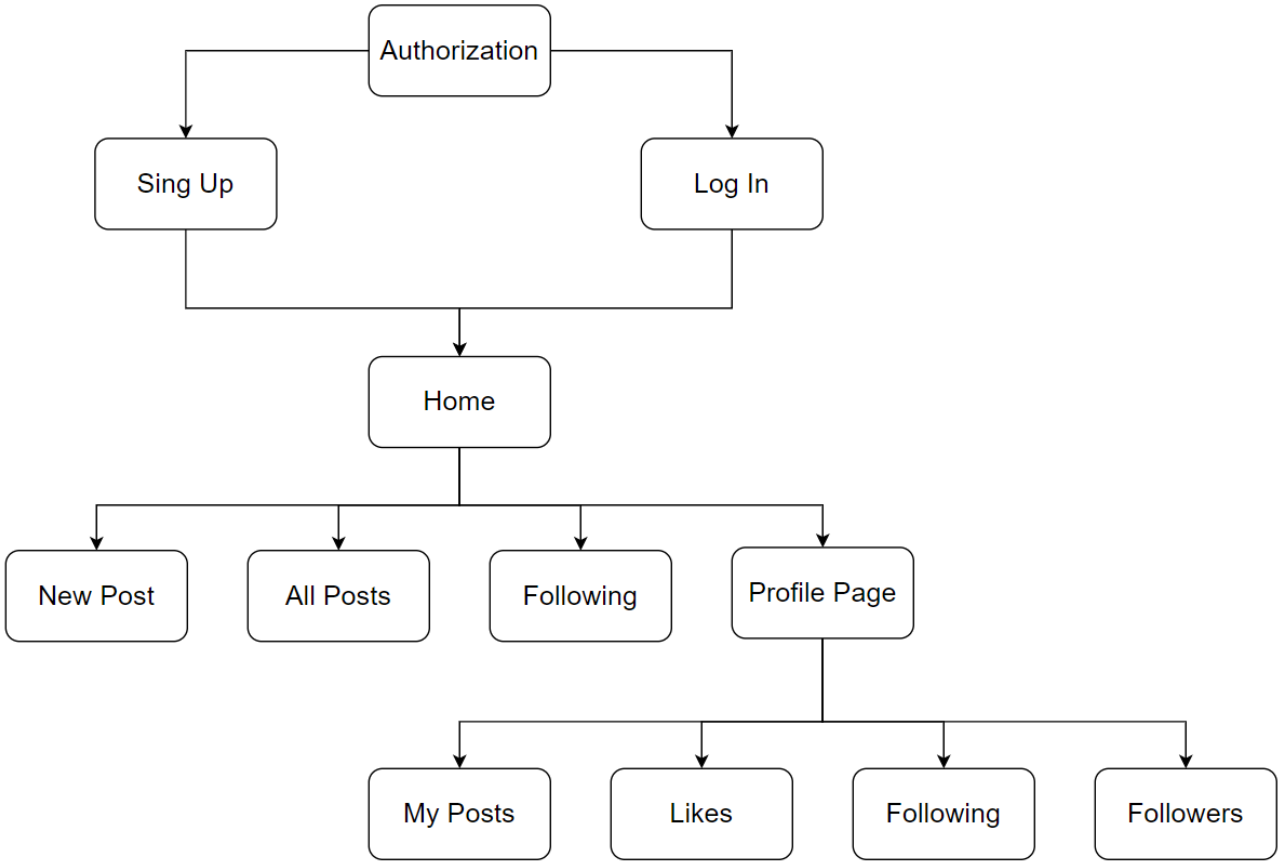

Рисунок 2.1 – Проєктування структури вебзастосунку «Liaise»

Головна сторінка є невід'ємною складовою сайту, тому її розробка вимагає особливої уваги. Як правило, ця сторінка пропонує адаптовану навігацію та різноманітні практичні функції для роботи.

Вебзастосунок «Liaise» містить основні елементи, такі як «шапка», панель навігації та основний інформаційний блок. Верхня частина головної сторінки містить «шапку», яка переважно повторюється на інших сторінках. Користувачі в першу чергу ідентифікують вебресурс через цей блок, оскільки він містить назву сайту і важливі компоненти, такі як панель навігації та налаштування.

Іншим ключовим елементом є панель навігації, яка містить посилання на основні розділи сайту. Горизонтальна розкладка є найпоширенішим форматом головного меню. Зважаючи на його важливість, меню має бути наочним, зручним і зрозумілим. Невідповідність цим критеріям може призвести до того, що користувачам буде важко ефективно орієнтуватися на сайті, що потенційно позбавить їх бажання витрачати на нього час.

Основний інформаційний блок адаптує свій контент відповідно до поточної сторінки користувача. Для головної сторінки зазвичай міститься «стрічка дописів», що містить останні публікації від усіх користувачів або підписок.

Ці елементи разом утворюють базову структуру соціальної мережі, надаючи користувачам можливість підключатися, ділитися вмістом і взаємодіяти з іншими в мережі. Крім того, такі функції, як сповіщення та налаштування конфіденційності, включені для покращення взаємодії з користувачем і контролю над своїм профілем.

2.2 Основні принципи роботи застосунку

Для проєктування функціонального і зрозумілого вебзастосунку використаємо UML. Уніфікована мова моделювання (UML) – це стандартизована мова моделювання, яка широко використовується в розробці

програмного забезпечення для візуалізації, специфікації, побудови та документування артефактів системи.

Основні принципи проєктування програми з використанням UML включають створення різних типів діаграм для представлення різних аспектів системи. Ось кілька ключових діаграм UML, використаних при проєктуванні роботи вебзастосунку «Liaise», і їх основні принципи:

– діаграма прецедентів;
– діаграма класів;
– діаграма послідовності.

Принципи наведених діаграм можна сформувати наступним чином.

Діаграма прецедентів представляє функціональність системи з точки зору користувача. Включає акторів (користувачів або зовнішні системи) і випадки використання (функціональні можливості системи). Описує, як користувачі взаємодіють із системою.

Діаграма класів ілюструє статичну структуру системи з точки зору класів та їхніх зв’язків. Включає класи, атрибути, методи та асоціації між класами. Представляє схему для об’єктів у системі.

Діаграма послідовності зображує взаємодію між об’єктами в часі. Показує потік повідомлень і порядок взаємодії між об’єктами. Корисно для моделювання динамічних аспектів системи [9].

Під час проєктування програми з використанням UML зазвичай використовують комбінацію цих діаграм, щоб забезпечити повне уявлення про систему з різних точок зору. Вибір діаграм, які використовувати, залежить від характеру системи та інформації, яку потрібно передати. Крім того, важливо підтримувати узгодженість діаграм і оновлювати їх у міру розвитку системи.

Спершу розробимо діаграму класів. На рисунку 2.2 зображена діаграма класів вебзастосунку «Liaise». Діаграма містить основні класи: користувач, публікація, фото, коментар, уподобання, підписки.

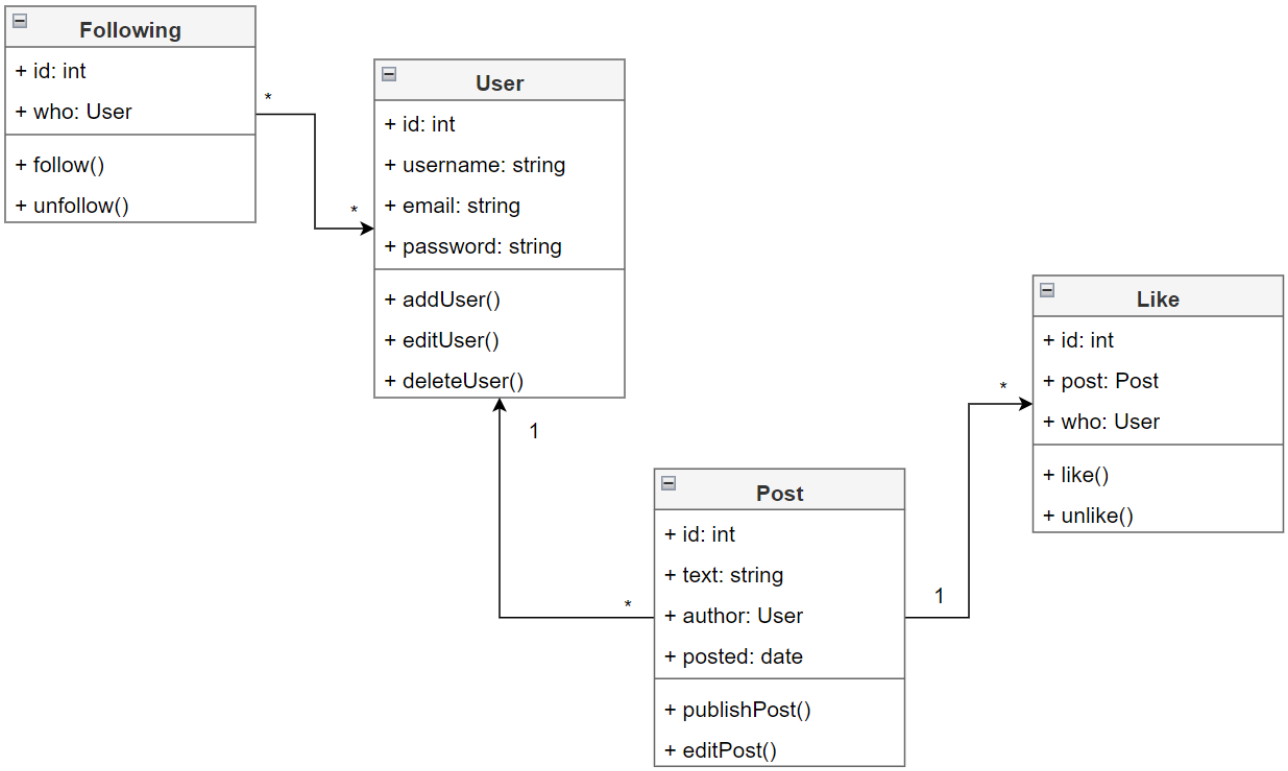

Рисунок 2.2 – Діаграма класів вебзастосунку «Liaise»

До діаграми класів соціальної мережі «Liaise» створено пояснювальну таблицю 2.1 з класами, атрибутами та методами, створеними для розробки вебзастосунку.

Таблиця 2.1 – Опис класів вебзастосунку «Liaise»

Параметр	**Значення**
Клас User	
Коментар	Клас, що є користувачем застосунку
Атрибути	id – ідентифікаційний номер користувача username – ім’я користувача email – поштова адреса для входу password – пароль для входу
Методи	addUser() – додати користувача editUser() – редагувати інформацію користувача deleteUser() – видалити користувача

Продовження табл. 2.1

Клас User	
Методи	searchUser() – пошук користувача
Клас Post	
Коментар	Клас, що є публікацією
Атрибути	id – ідентифікаційний номер публікації text – текст публікації author – ім’я користувача (автора публікації) posted – дата публікації
Методи	publishPost() – додати публікацію editPost() – редагувати публікацію deletePost() – видалити публікацію
Клас Like	
Коментар	Клас, що відповідає за «Like»
Атрибути	post – публікація, до якої залишений «Like» who – користувач, який залишив «Like»
Методи	like() – залишити «Like» unlike() – прибрати «Like»
Клас Following	
Коментар	Клас, що відповідає за підписки
Атрибути	id – ідентифікаційний номер who – користувач, який підписався або за яким почали стежити
Методи	follow() – підписатися на користувача unfollow() – відписатися від користувача

Для ілюстрації інтерактивності об’єктів поведінки системи у мові UML використовується діаграма послідовності. Його перевага полягає в представленні зв’язків між об’єктами послідовним способом.

На рисунку 2.3 зображено діаграму послідовності процесу «Авторизація». На цій діаграмі розміщені об’єкти: User, Sing up/Log In, Validation та DB.

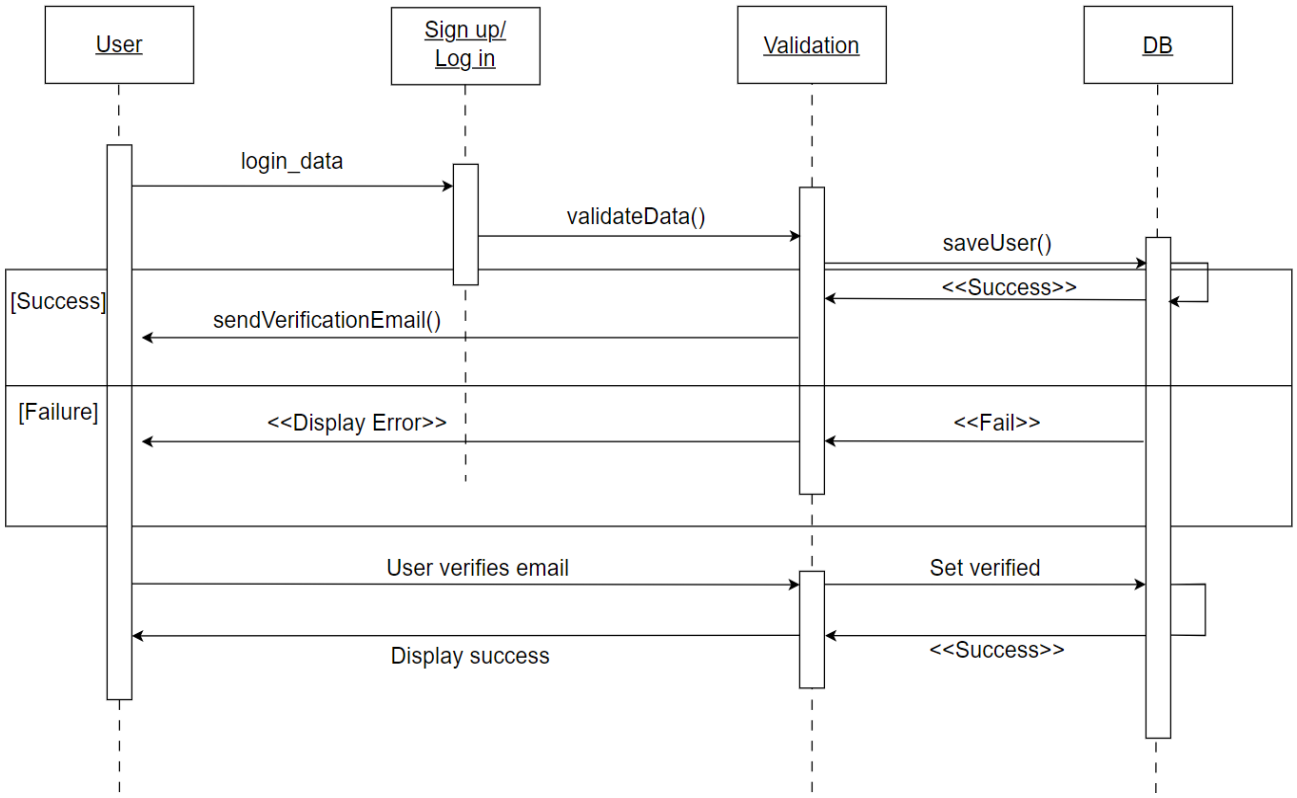

Рисунок 2.3 – Діаграма послідовності для процесу «Авторизація»

Послідовність подій розгортається наступним чином: користувач ініціює процес авторизації, натиснувши на кнопку авторизації. Згодом користувач перенаправляється до вебзастосунку, який потім надсилає запит сторінки на сервер. Сервер, у свою чергу, відповідає видачою сторінки, яка відображається користувачеві. Користувач переходить до заповнення форми авторизації та натискає кнопку підтвердження. Після цього запит передається назад на сервер, який пересилає його в базу даних. Сервер обробляє запит, а наступний крок включає отримання результату від сервера бази даних. Після обробки даних сервер повертає користувачеві підтвердження про успішну авторизацію.

На рисунку 2.4 зображено діаграму послідовності процесу «Редагувати публікацію». На цій діаграмі розміщено наступні об’єкти: User, Home, Post та DB.

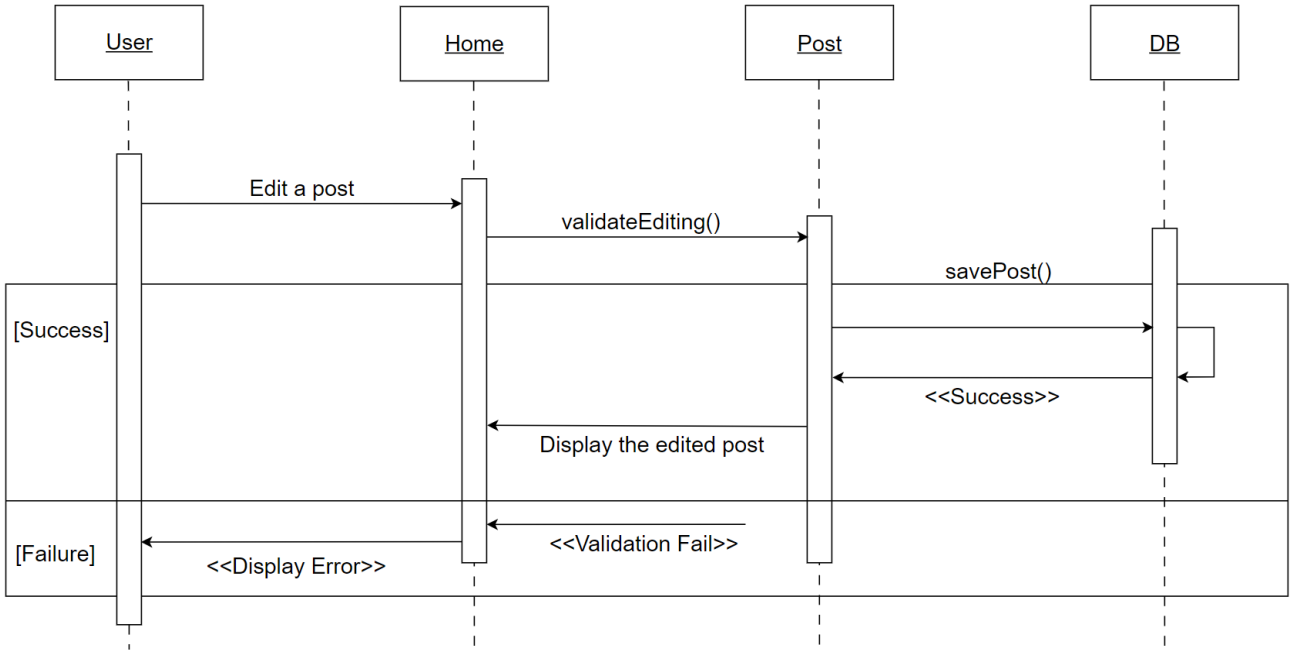

Рисунок 2.4 – Діаграма послідовності для процесу «Редагувати публікацію»

На діаграмі послідовності дій для процесу «Редагувати публікацію» відбувається наступне: користувач натискає на кнопку «Edit», відкривається вікно редагування публікації. Після цього користувач здійснює редакцію публікації та натискає кнопку «Save changes». Далі запит іде на сервер, сервер надсилає запит до бази даних, потім здійснюється обробка запиту, наступним є отримання результату, після чого відбувається опрацювання даних, а як результат сервер повертає підтвердження про успішну редакцію і редагована публікація оновлюється.

Діаграма використання, також відома як діаграма прецедентів або Use Case, надає візуальне представлення ролей у системі та їх взаємодії. Вона зосереджена на ілюстрації типів ролей та їхніх відносин із системою. Однак не зображує послідовний порядок кроків. Натомість діаграма представляє функціональні вимоги, демонструючи, які дії може виконувати система з точки зору користувача. Цю інформацію можна передати як у текстовому вигляді, так і за допомогою схеми [13].

На рисунку 2.5 зображена діаграма варіантів використання вебзастосунку «Liaise», яка містить трьох акторів: адміністратора, авторизованого та неавторизованого користувача.

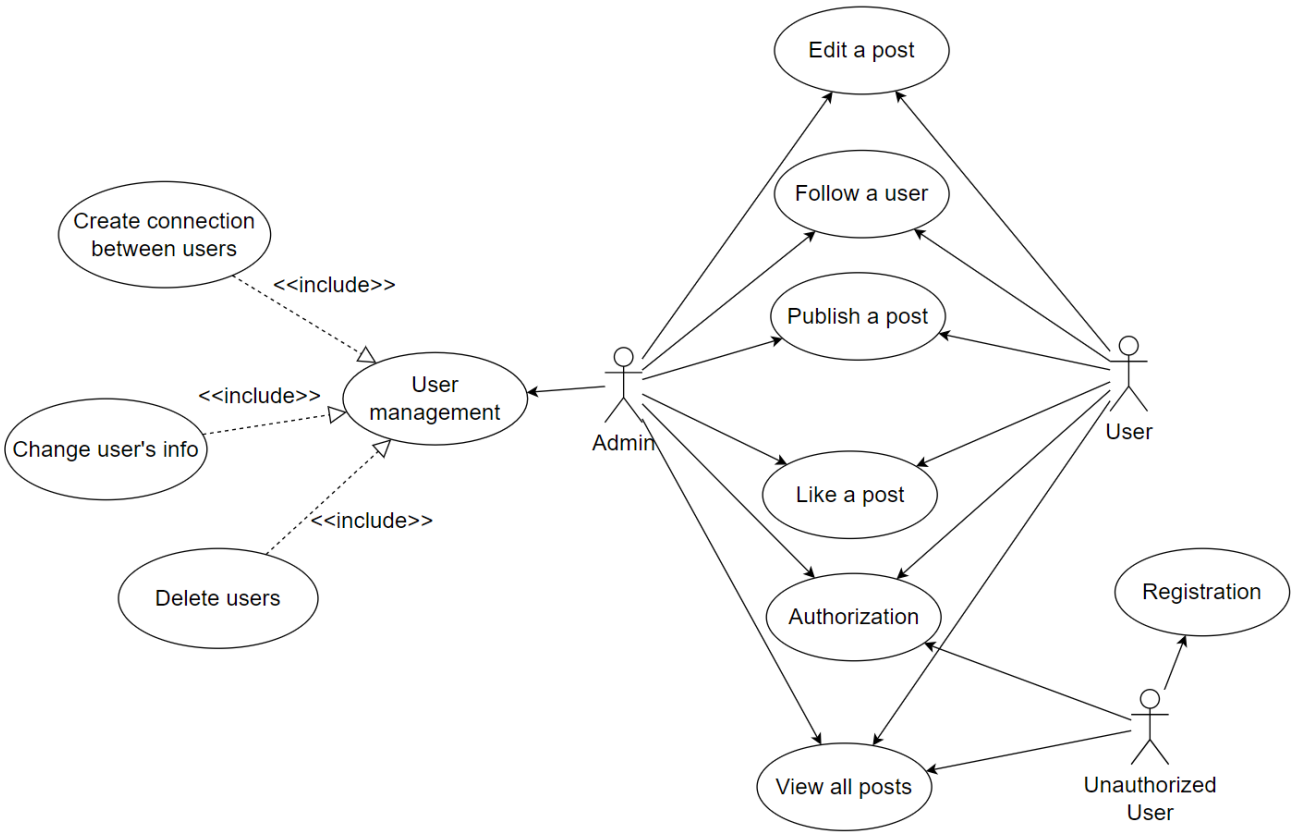

Рисунок 2.5 – Діаграма прецедентів вебзастосунку «Liaise»

Кожен актор має певні відмінності у доступному йому функціоналі. Найбільш обмежений функціонал у неавторизованого користувача: реєстрація, авторизація і перегляд сторінки «All Posts». Щодо авторизованого користувача, то він має додаткові функції, такі як: редагування публікації, підписка, публікація допису, вподобання, авторизація та перегляд всіх постів. Реєстрація для такого користувача вже недоступна. Користувач з правами адміністратора має доступ до всіх функцій користувача, окрім реєстрації, а також додатковий розширений функціонал для управління користувачами. Який включає у себе ще декілька додаткових функцій:

– створення зв’язків між користувачами (редагування вподобань і підписок);
– редагування інформації користувачів;
– видалення користувачів.

На діаграмі прецедентів (див. рис. 2.6) зображено двох акторів. Користувач (автор публікації) і підписник.

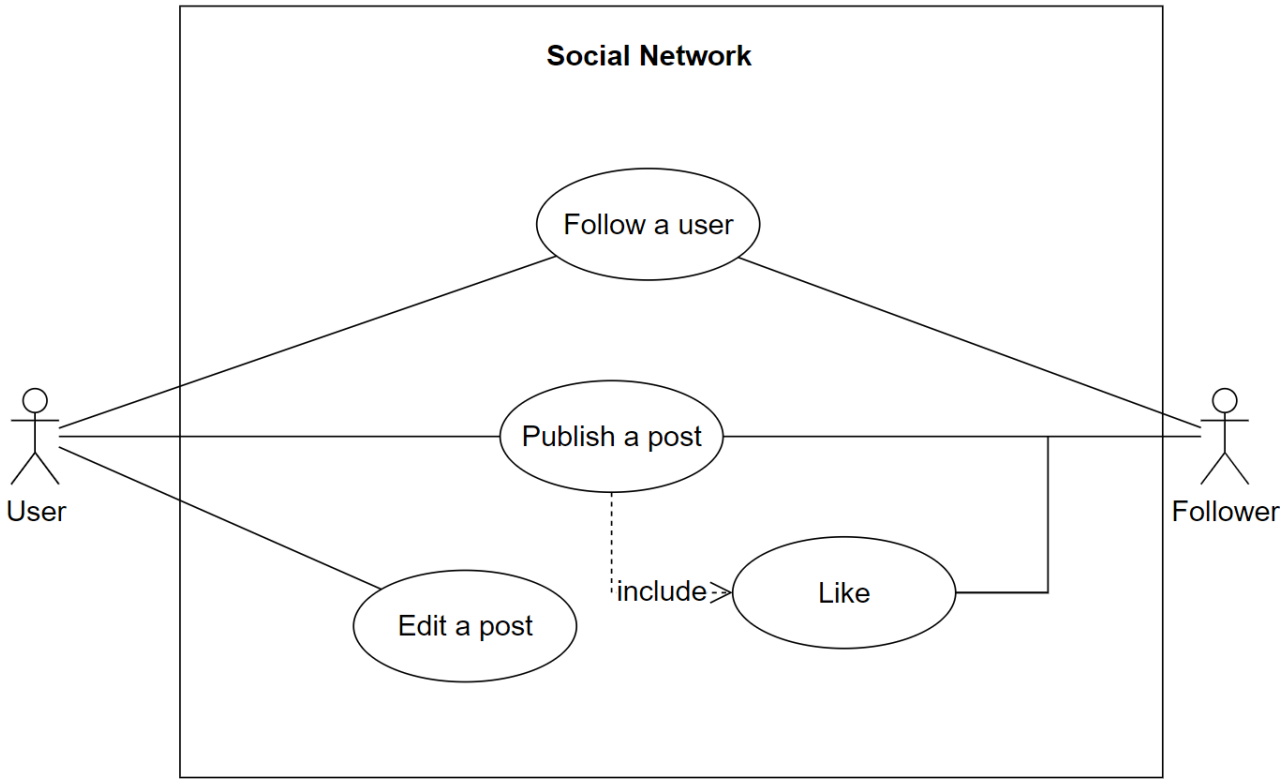

Рисунок 2.6 – Діаграма прецедентів взаємодії між користувачами

Автор публікації може опублікувати допис. У системі є варіант використання «Publish a post», а всередині є кілька елементів, які він включає. Написання публікації може бути з нуля. Також є можливість залишити «Like». Це частина функціональності, яка міститься в сценарії використання «Publish a post».

Також є у автора публікації можливість слідкувати за іншим користувачем. Ідентичну можливість має інший користувач (наприклад, підписник).

У системі є користувач «Підписник». Він має змогу створити публікацію, поставити «Like». Технічно це простий варіант використання, тому що користувач і підписник технічно взаємозамінні. Якщо користувач стежить за іншим користувачем і є підписником, він все ще є автором. Кожен користувач є автором, і тому функціональні можливості поширюються на всі випадки використання. Зображення другорядного актора на діаграмі, полягає в тому, щоб показати, до чого підписник має доступ.

Під час практичного застосування UML до вебзастосунку «Liaise» продемонстровано розробку діаграми класів, що представляє ключові класи, такі як користувач, публікація, фотографія, коментар, вподобання і підписка. Крім того, за допомогою діаграми послідовності проілюстровано інтерактивність об’єктів поведінки системи, демонструючи такі процеси, як процедури «Авторизація» та «Редагувати публікацію». Діаграма прецедентів забезпечила візуальне представлення ролей у системі, наголошуючи на взаємодії між акторами та системою.

2.3 Проєктування структури бази даних

Entity-relationship діаграми складаються з сутностей, відносин та атрибутів. Ці діаграми широко використовуються для проєктування або налагодження реляційних баз даних у різних областях, включаючи бізнес-інформаційні та дослідницькі системи [14].

Щоб встановити систематичний підхід до ефективної обробки даних, важливо спланувати структуру бази даних. Це передбачає проведення ретельного аналізу предметної області, вирішення потенційних проблем і подальше створення концептуальної моделі бази даних на основі цих міркувань.

Сутністю вважають визначений об’єкт, який може бути представлений у вигляді людини, концепції, або ж події. Сутність може містити дані, які в ній зберігаються. В діаграмі відображається у вигляді прямокутника.

Відносини – це зв’язок між суб’єктами. Вони показують як суб’єкти діють один на одного або зображують зв’язок між ними. Відносини мають вигляд ромбу.

Атрибутом є властивість або характеристика відносини порівняно з сутністю. Відображається як овал або круг.

Відношення між сутностями в ER-діаграмі зображується у вигляді лінії,

яка з’єднує ці сутності. За допомогою відношень – можна вказувати типи зв’язку між сутностями. Під час проєктування ER-діаграми до вебзастосунку «Liaise» використано два типи зв’язку.

Зв’язок «один-до-багатьох» складається з позначення від «одиниці» до «багатьох» або від «нуля» до «багатьох» на одній стороні зв’язку та нотації «один і тільки один» або «нуль» або «один на іншій».

Наступним зв’язком є «багато-до-багатьох». Він складається з «нуля» до «багатьох» або від «одного» до «багатьох» з обох сторін зв’язку. Ця конструкція існує лише в логічних моделях даних, оскільки бази даних не можуть реалізувати зв’язок безпосередньо [15].

ER-модель є зручним інструментом для розуміння структури та проєктування бази даних. Тому було спроєктовано діаграму «сутність-зв’язок» для вебзастосунку «Liaise».

На рисунку 2.7 зображено ER-діаграму вебзастосунку соціальної мережі, яка містить 5 сутностей.

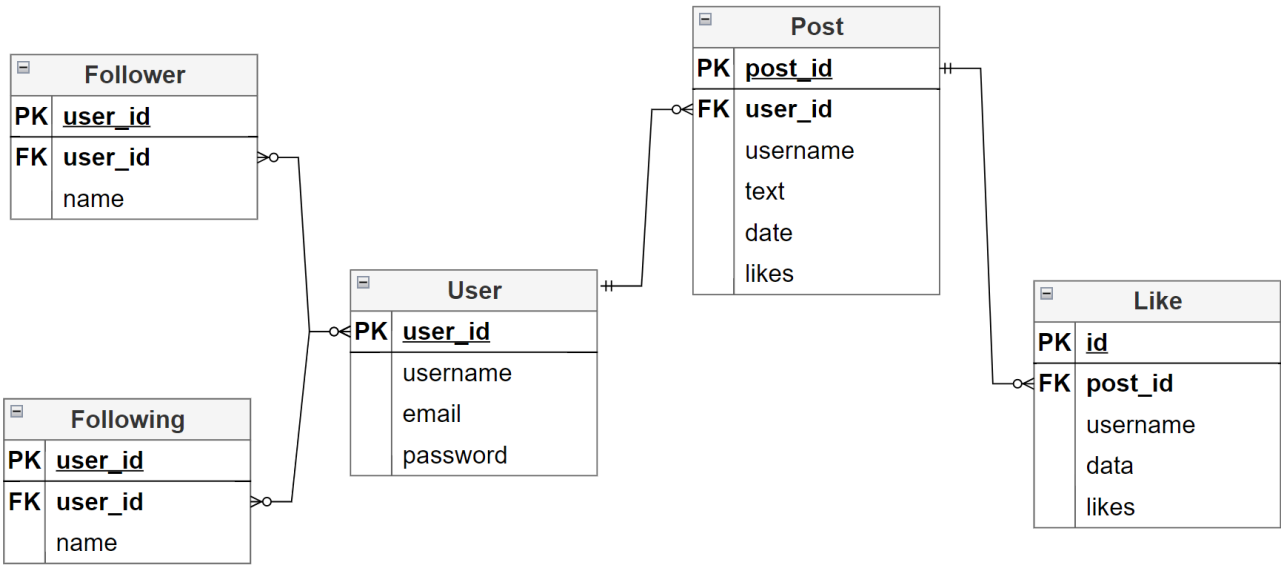

Рисунок 2.7 – ER-діаграма вебзастосунку «Liaise»

У даній ER-моделі зображено такі відношення та сутності:

– «User» та «Following» мають відношення «багато-до-багатьох», оскільки користувач має змогу підписуватися на багатьох

користувачів;

– «User» та «Follower» мають відношення «багато-до-багатьох», оскільки користувач може мати багато підписників;
– «User» і «Post» – відношення «один-до-багатьох», тому що один користувач може мати багато публікацій;
– «Post» та «Like» також має відношення «один-до-багатьох» оскільки одна публікація може мати багато вподобань.

3 ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ТА РЕЗУЛЬТАТИ ТЕСТУВАННННЯ

Розробка вебзастосунку соціальної мережі «Liaise» відбувається з використанням Django, високорівневого вебфреймворка Python, мови сценаріїв JavaScript та автономної реляційної бази даних SQLite. Цей розділ містить у собі опис реалізації інтерфейсу вебзастосунку, а також його серверної частини і тестування.

3.1 Реалізація інтерфейсу вебзастосунку

Створення інтерфейсу вебзастосунку за допомогою HTML, CSS, JavaScript і Django включає в себе поєднання зовнішніх технологій (HTML, CSS, JS) із серверною платформою (Django) для реалізації динамічного-оновлюваних вебсторінок.

Спочатку відбувається реалізація інтерфейсу для входу та реєстрації у соціальну мережу. На рисунку 3.1 зображено сторінку авторизації до вебзастосунку. Вона містить навігаційну панель з логотипом, розділом «All Posts», який доступний не лише користувачам системи, та посиланнями з входом до сайту чи реєстрацією. На основній частині сторінки розміщено форму для заповнення даних користувача. Для реєстрації у системі необхідно вказати наступні дані:

– username – ім’я користувача;
– email – адреса електронної скриньки користувача;
– password – пароль для входу у обліковий запис;
– confirm password – підтвердження паролю до щойно створеного облікового запису.

Після реєстрації користувач має змогу продовжувати роботу із вебзастосунком з розширеним функціоналом.

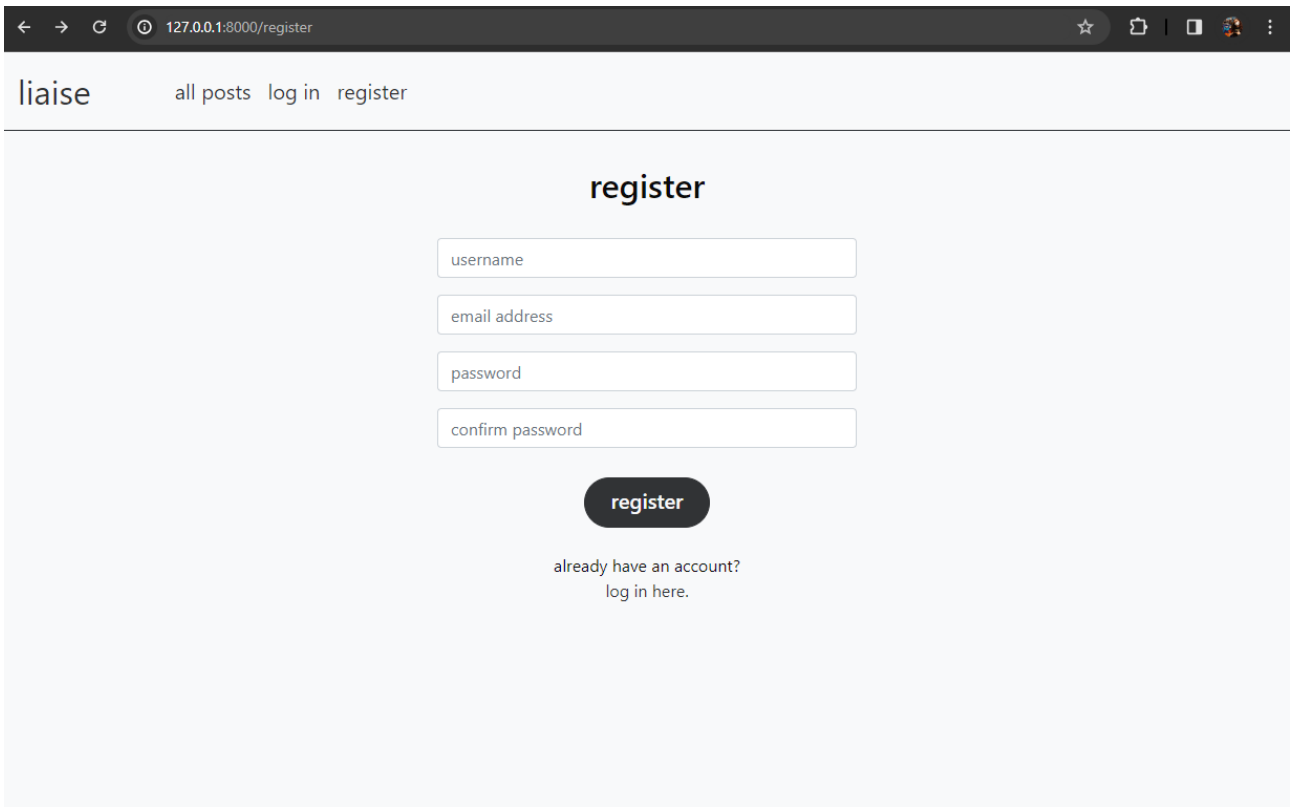

Рисунок 3.1 – Сторінка реєстрації до вебзастосунку

На рисунку 3.2 зображено код розмітки авторизації до соціальної мережі.

```
{% extends "network/layout.html" %}

{% block body %}

    <h2>Login</h2>

    {% if message %}
        <div>{{ message }}</div>
    {% endif %}

    <form action="{% url 'login' %}" method="post">
        {% csrf_token %}
        <div class="form-group">
            <input autofocus class="form-control" type="text" name="username" placeholder="Username">
        </div>
        <div class="form-group">
            <input class="form-control" type="password" name="password" placeholder="Password">
        </div>
        <input class="btn btn-primary" type="submit" value="Login">
    </form>

    Don't have an account? <a href="{% url 'register' %}">Register here.</a>

{% endblock %}
```

Рисунок 3.2 – Код розмітки авторизації вебзастосунку

Задля її створення використано HTML та функціонал фреймворку Django, для створення динамічної розмітки вебсторінки і використання CSRF-токену для безпечної авторизації.

Реалізація «домашньої сторінки» застосунку показана на рисунку 3.3, на якій розміщені публікації всіх користувачів, і яка доступна авторизованим та неавторизованим користувачам.

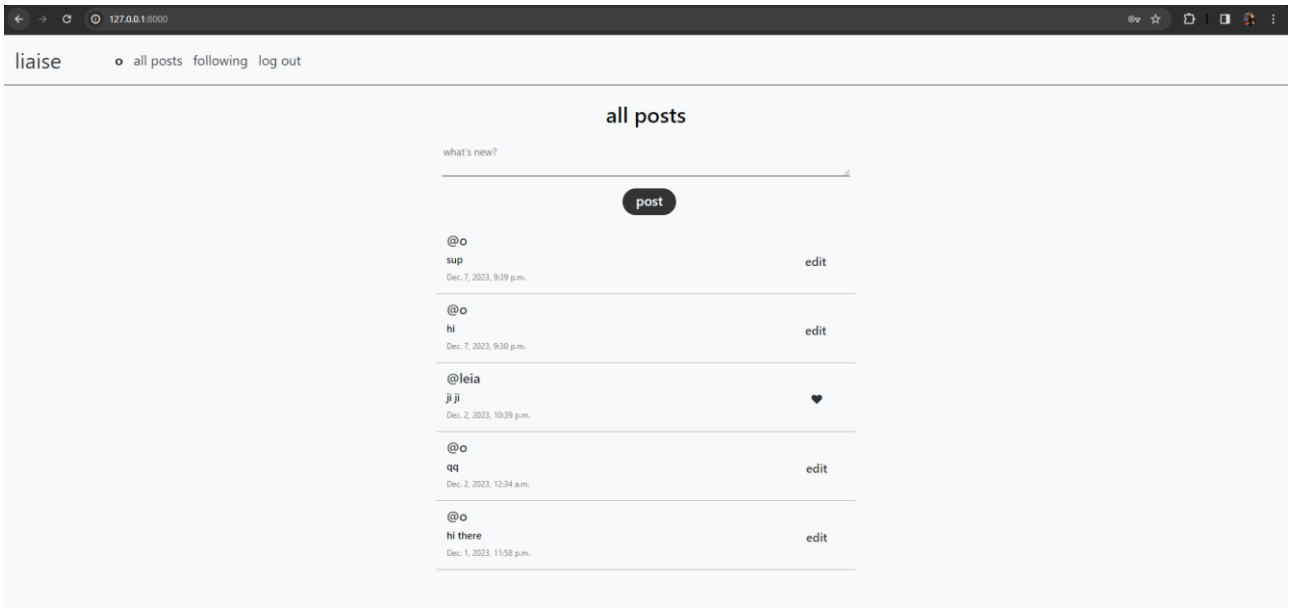

Рисунок 3.3 – Домашня сторінка вебзастосунку

На цій сторінці авторизований користувач може спостерігати навігаційну панель, на якій розміщено назву соціальної мережі, ім’я авторизованого користувача, з можливістю перейти до сторінки свого облікового запису, вкладки «All Posts» та «Following» та функція виходу із облікового запису. Також поле вводу нової публікації. Нижче розміщені публікації користувачів. Аналогічний дизайн має вкладка «Following», проте вона містить публікації тільки тих користувачів, за якими стежить авторизований користувач, на відмінну від вкладки «All Posts» і доступна також тільки користувачам, які увійшли до системи. Окрім публікації, також є функції «Like» та «Edit». Користувач вебзастосунку за бажанням може редагувати свої публікації та залишати «Like» на публікаціях інших користувачів.

Для неавторизованого користувача «домашня сторінка» має подібний вигляд, проте для нього відсутні вкладки «Профіль» та «Following». Але присутні функції входу та реєстрації, аналогічні до тих, що розміщено на сторінці з авторизацією.

На рисунку 3.4 зображено розмітку домашньої сторінки, а саме для зареєстрованого користувача.

Авторизований користувач має додаткові функції, такі як публікація дописів, їх редагування, слідкування за іншими користувачами та функція поставити «Like» на дописи інших користувачів.

```
<div class="all-posts">
 {% for post in posts_of_the_page %}
   <div class="row post col-4 ">
    <h5 class="username"><a href="{% url 'profile' user_id=post.user.id %}">@{{ post.user }}</a></h5>
    <h6 class="content" id="content_{{ post.id }}">{{ post.content }}</h6>
    <p class="date">{{ post.date }}</p>
    {% if user.is_authenticated %}
     {% if user == post.user %}
      <div class="d-flex justify-content-around">
       <button class="btn btn-primary" data-toggle="modal" data-target="#modal_edit_post_{{ post.id }}">Edit</button>
      </div>
      <div class="modal fade" id="modal_edit_post_{{ post.id }}" tabindex="-1" role="dialog" aria-labelledby="modal_edit_post_{{ post.id }}_label"
       <div class="modal-dialog" role="document">
        <div class="modal-content">
         <div class="modal-header">
          <h5 class="modal-title">Edit Post</h5>
          <button type="button" class="close" data-dismiss="modal" aria-label="Close">
           <span aria-hidden="true">&times;</span>
          </button>
         </div>
         <div class="modal-body">
          <textarea row="5" id="textarea_{{ post.id }}" class="form-control" name="content">{{ post.content }}</textarea>
         </div>
         <div class="modal-footer">
          <button type="button" class="btn btn-primary" onclick="submitHandler({{ post.id }})">Save changes</button>
          <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
         </div>
        </div>
       </div>
      </div>
     {% else %}
```

Рисунок 3.4 – Код розмітки домашньої сторінки «All Posts»

Фрагмент коду вище містить у собі розмітку модального вікна для функції редагування власної публікації для авторизованого користувача. Для реалізації даної частини програми, необхідно використати функціонал мови JavaScript. Оскільки для роботи програми достатньо трьох JavaScript-функцій, їх можна розмістити у HTML-файлі з розміткою сторінки, на якій використовуються ці функції. Код розміщується між двома тегами <script>. На рисунку 3.5 продемонстровано частину коду JavaScript у HTML-файлі.

```
<script>
  function getCookie(name){
    const value = `; ${document.cookie}`;
    const parts = value.split(`; ${name}=`);
    if(parts.length == 2) return parts.pop().split(';').shift();
  }

  function submitHandler(id){
    const textareaValue = document.getElementById(`textarea_${id}`).value;
    const content = document.getElementById(`content_${id}`);
    const model = document.getElementById(`modal_edit_post_${id}`);
    fetch(`/edit/${id}`, {
      method: "POST",
      headers: {"Conten-type": "application/json", "X-CSRFToken": getCookie("csrftoken")},
      body: JSON.stringify({
        content: textareaValue
      })
    })
    .then(response => response.json())
    .then(result => {
        content.innerHTML = result.data;
        modal.classList.remove('show');
        modal.setAttribute('aria-hidden', 'true');
        modal.setAttribute('style', 'display: none');

        // get modal backdrops
        const modalsBackdrops = document.getElementsByClassName('modal-backdrop');

        // remove every modal backdrop
        for(let i=0; i<modalsBackdrops.length; i++) {
          document.body.removeChild(modalsBackdrops[i]);
        }
    })
```

Рисунок 3.5 – Код JavaScript

Рисунок 3.5 містить дві функції, написані мовою JavaScript. Функції розроблені для коректної роботи модального вікна редагування публікацій користувачів.

Також розроблено розмітку для сторінки «Following», яка є майже ідентичною до сторінки «All Posts», за виключенням того, що вона містить лише дописи користувачів за якими стежить авторизований користувач, тобто публікації його підписок.

Наступною необхідною до розробки була сторінка «Profile». Вона містить у собі ім’я користувача, кількість його підписників та підписок, а також всі опубліковані дописи (див. рис. 3.6).

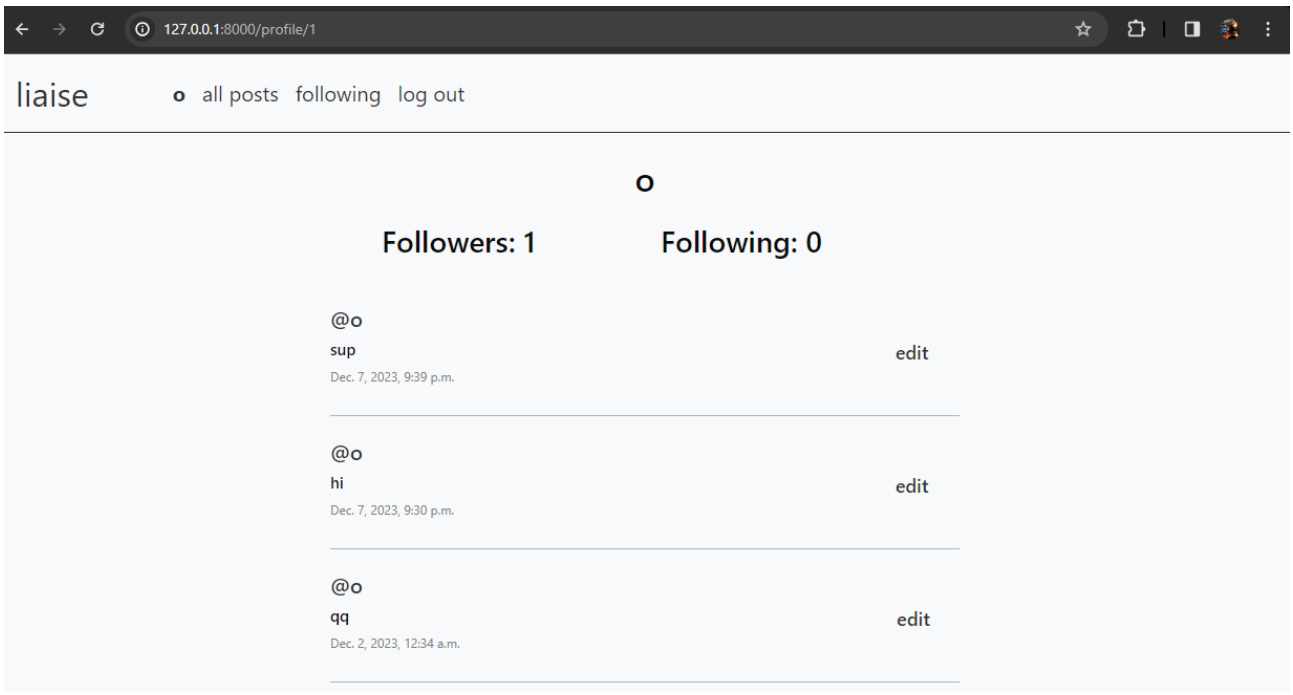

Рисунок 3.6 – Сторінка профілю

Також, якщо користувач відвідує сторінку іншого користувача системи, він має функції «Follow» або «Unfollow», реалізовані у вигляді кнопки, які дають змогу підписатися на відвідуваний обліковий запис або відписатися від нього.

3.2 Реалізація серверної частини вебзастосунку

Далі, задля коректної роботи вебзастосунку, необхідна розробка серверної частини. У даній роботі для серверної реалізації обрано високорівневий вебфреймворк Django, який дозволяє швидко і ефективно створювати різні вебсервіси.

Під час виконання даної роботи було створено Django-проєкт командою «django-admin startproject». Цей метод полегшує початок створення проєкту і у ньому вже присутні необхідні файли для функціонування програми.

У проєкті «Liaise» основними файлами які були використані під час роботи є:

– models.py;
– views.py;
– urls.py;
– admin.py.

Файл models.py містить основні поля та поведінку даних, які потрібно зберегти у базі даних. Як правило, кожна модель відображається у вигляді однієї таблиці з бази даних. Кожен атрибут моделі представляє поле бази даних.

Рисунок 3.7 містить фрагмент коду із файлу «models.py», який використовується у вебзастосунку «Liaise». У ньому міститься чотири реалізованих класи, які відповідають за таблиці у базі даних. Для роботи програми достатньо чотири класи, які містять інформацію про користувача, публікацію, підписки та вподобання.

```python
from django.contrib.auth.models import AbstractUser
from django.db import models


class User(AbstractUser):
    pass


class Post(models.Model):
    content = models.CharField(max_length=140)
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="author")
    date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Post {self.id} made {self.user} on {self.date.strftime('%d %b %Y %H:%M:%S')}"

class Follow(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="user_who_is_following")
    user_follower = models.ForeignKey(User, on_delete=models.CASCADE, related_name="user_who_is_being_followed")

    def __str__(self):
        return f"{self.user} is following {self.user_follower}"
```

Рисунок 3.7 – Фрагмент коду із файлу «models.py»

Наступний файл – views.py. По суті, це функція Python, яка приймає вебзапит і повертає вебвідповідь. Цією відповіддю може бути HTML-вміст вебсторінки, перенаправлення, помилка 404, документ XML або зображення. Або будь-що, насправді [16].

Для початку імпортуємо моделі із файлу «models.py». Далі відбувається розробка функцій. У даній роботі цей файл містить дванадцять функцій:

– register;
– logout_view;
– login_view;
– unfollow;
– follow;
– following;
– profile;
– newPost;
– index;
– edit;
– add_like;
– remove_like.

Перша функція містить у собі логіку необхідну для реєстрації нового користувача. Працює за допомогою запиту методу «POST». Перевіряє чи співпадають паролі вказані при реєстрації, а також містить перевірку імені користувача. Якщо введені паролі не співпали або ім’я користувача вже зареєстроване у системі, у користувача не вийде створення облікового запису.

Наступні дві функції відповідають за вхід і вихід із системи. Перша функція logout_view() використовує лише метод logout та відправляє HTTP-відповідь. Друга функція login_view() має дещо подібний функціонал до функції register(). Містить перевірку на існування імені користувача, а також правильності введеного пароля.

На рисунку 3.8 продемонстровано фрагмент коду із функціями follow() і unfollow(), що відповідають за підписку і відписку від користувача, відповідно.

Функції follow() та unfollow() опрацьовують дані авторизованого користувача і користувачів системи, на яких потенційно може підписатися поточний користувач, або які можуть мати роль підписника.

```python
def follow(request):
    userfollow = request.POST['userfollow']
    currentUser = User.objects.get(pk=request.user.id)
    userFollowData = User.objects.get(username=userfollow)
    f = Follow(user=currentUser, user_follower=userFollowData)
    f.save()
    user_id = userFollowData.id
    return HttpResponseRedirect(reverse(profile, kwargs={'user_id': user_id}))

def unfollow(request):
    userfollow = request.POST['userfollow']
    currentUser = User.objects.get(pk=request.user.id)
    userFollowData = User.objects.get(username=userfollow)
    f = Follow.objects.get(user=currentUser, user_follower=userFollowData)
    f.delete()
    user_id = userFollowData.id
    return HttpResponseRedirect(reverse(profile, kwargs={'user_id': user_id}))
```

Рисунок 3.8 – Фрагмент коду із файлу «views.py», який містить функції «follow» і «unfollow»

Використані функції мають доволі подібний вигляд, але використовують різні методи «save» та «delete» для підписки та відписки.

На наступному рисунку 3.9 продемонстровано функцію following(), яка відповідає за відображення вмісту сторінки «Following», що містить усі публікації підписок авторизованого користувача.

```python
def following(request):
    currentUser = User.objects.get(pk=request.user.id)
    followingPeople = Follow.objects.filter(user=currentUser)
    allPosts = Post.objects.all().order_by('id').reverse()
    followingPosts = []

    for post in allPosts:
        for person in followingPeople:
            if person.user_follower == post.user:
                followingPosts.append(post)

    # Pagination
    paginator = Paginator(followingPosts, 10)
    page_number = request.GET.get('page')
    posts_of_the_page = paginator.get_page(page_number)

    return render(request, "network/following.html", {
        "posts_of_the_page": posts_of_the_page
    })
```

Рисунок 3.9 – Фрагмент коду із файлу «views.py», який містить функцію «Following»

Також слід відмітити реалізацію «Pagination». «Pagination» відповідає за відображення контенту на сторінці. У функції Paginator(), яка є однією із вбудованих функцій Django міститься інформація з усіма постами, які мають бути показані на сторінці, а також їхня кількість на одну сторінку. У цій роботі було обрано формат відображення десяти публікацій на одну сторінку.

Використання функції reverse() допомагає у сортування публікацій від найновіших до найдавніших, для відображення нових дописів зверху сторінки.

Наступною є невеличка функція newPost(), що відповідає за публікацію нових дописів. Її вміст продемонстровано на рисунку 3.10.

```
60  def newPost(request):
61      if request.method == "POST":
62          content = request.POST['content']
63          user = User.objects.get(pk=request.user.id)
64          post = Post(content=content, user=user)
65          post.save()
66          return HttpResponseRedirect(reverse(index))
67
```

Рисунок 3.10 – Фрагмент коду із файлу «views.py», який містить функцію «newPost»

Слідуючою згаданою функцією у переліку вище є функція index(). Вона відповідальна за відображення вмісту головної сторінки «Home». Має майже ідентичний вміст до функції following(), проте у ній відсутнє обмеження на публікації. Тобто відображаються публікації усіх користувачів системи. Має такий ж спосіб сортування дописів – від найновіших до найдавніших, а також включає функцію Paginator(). І містить у собі логіку функції «Like» з перевіркою про вподобання публікації авторизованим користувачем. Якщо користувач вже залишив «Like» на дописі, то він має змогу його прибрати. Або ж навпаки вподобати обраний допис, якщо він цього ще не зробив.

Однією із важливих функцій є наступна функція «Edit» (див. рис. 3.11), яка відповідає за редагування користувачем власних публікацій.

Функція зберігає відредагований допис. Проте перевірка на дозвіл редагування реалізована у HTML-файлі за допомогою мови Python.

```python
def edit(request, post_id):
    if request.method == "POST":
        data = json.loads(request.body)
        edit_post = Post.objects.get(pk=post_id)
        edit_post.content = data["content"]
        edit_post.save()
        return JsonResponse({"message": "Change successful", "data": data["content"]})
```

Рисунок 3.11 – Фрагмент коду із файлу «views.py», який містить функцію «edit»

Останніми є функції add_like() та remove_like(), які відповідають за роботу з вподобаннями користувача. Дані функції аналогічно до функції edit() зберігають дані про дії користувача, але основна логіка реалізована у HTML-файлі.

Наступним файлом, який слід розглянути є файл urls.py. Він містить у собі посилання з усіма функціями із файлу «views.py» для коректної роботи застосунку (див. рис. 3.12).

```python
from django.urls import path

from . import views

urlpatterns = [
    path("", views.index, name="index"),
    path("login", views.login_view, name="login"),
    path("logout", views.logout_view, name="logout"),
    path("register", views.register, name="register"),
    path("newPost", views.newPost, name="newPost"),
    path("profile/<int:user_id>", views.profile, name="profile"),
    path("unfollow", views.unfollow, name="unfollow"),
    path("follow", views.follow, name="follow"),
    path("following", views.following, name="following"),
    path("edit/<int:post_id>", views.edit, name="edit"),
    path("remove_like/<int:post_id>", views.remove_like, name="remove_like"),
    path("add_like/<int:post_id>", views.add_like, name="add_like"),
]

```

Рисунок 3.12 – Вміст файлу «urls.py»

Посилання на views.index створене автоматично про створенні проєкту Django. А інші посилання створюються у процесі розробки вебзастосунку у

відповідності з функціями з «views.py».

Файл проєкту «admin.py» (див. рис. 3.13) використовується для відображення існуючих моделей на панелі адміністратора Django. Також з його допомогою розробник має можливість налаштовувати панель адміністратора.

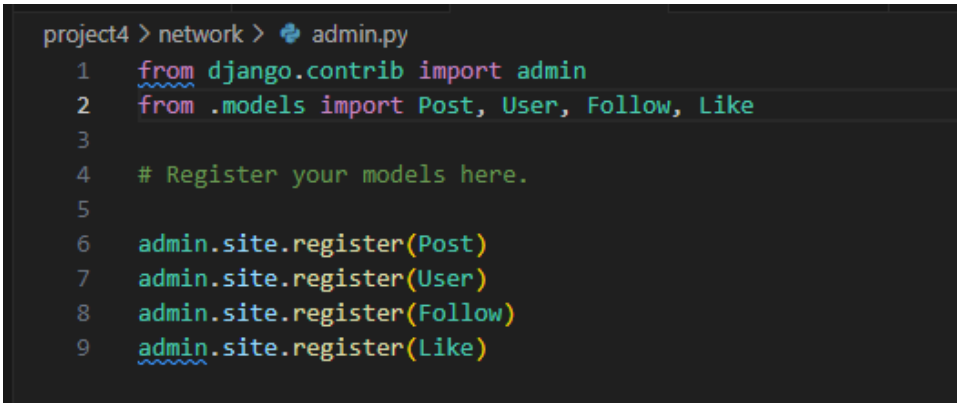

Рисунок 3.13 – Вміст файлу «admin.py»

Файл проєкту вебзастосунку «Liaise» використовує всі класи наявні у файлі «models.py» і відображає їх на панелі адміністратора Django (див. рис. 3.14).

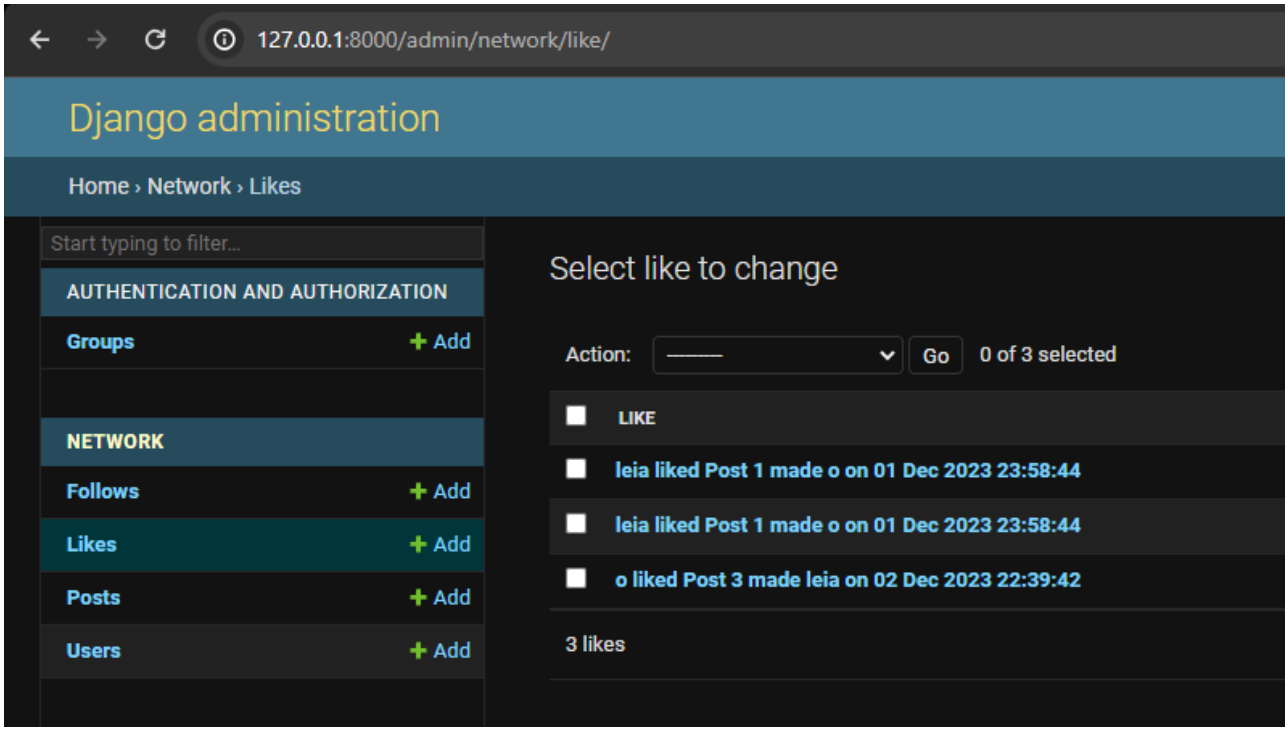

Рисунок 3.14 – Панель адміністратора Django вебзастосунку «Liaise»

Як і очікувалося на панелі адміністратор має доступ до користувачів, публікацій, вподобань на зв’язків між користувачами мережі. Також рисунок 3.14 містить частину інтерфейсу панелі адміністратора на якій відображено дії користувачів з вподобанням публікацій.

3.3 Тестування роботи інформаційної системи

Тестування роботи інформаційної системи – це важливий етап у розробці програмного забезпечення будь-якого виду, адже саме воно забезпечує безперебійну роботу систем різного рівня складності та предметної області.
Хоча цей етап зазвичай показаний як останній на схемах циклу розробки програмного забезпечення, насправді, таким не є. Бо розробка – це циклічний процес, який містить у собі повторення всіх етапів за час існування розроблюваних продуктів.

Тестування включає оцінку програмного забезпечення для виявлення будь-яких помилок, помилок або дефектів. Воно допомагає виявляти та виправляти дефекти на ранніх етапах процесу розробки, зменшуючи ймовірність того, що помилки досягнуть кінцевих користувачів. Також слід відмітити, що тестування допомагає виявити потенційні ризики та вразливі місця в програмному забезпеченні. Вирішуючи ці проблеми на ранній стадії, розробники можуть зменшити ризики, пов’язані зі збоями ПЗ, порушеннями безпеки та іншими несподіваними проблемами. У багатьох галузях існують нормативні вимоги та вимоги відповідності, яким має відповідати програмне забезпечення. Тестування гарантує, що програмне забезпечення відповідає цим стандартам, уникаючи юридичних проблем і штрафів.

Тестування оптимізованої продуктивності оцінює чуйність, швидкість і загальну ефективність програми. Це гарантує, що програмне забезпечення добре працює за різних умов і справляється з очікуваним навантаженням. Також існує тестування зручності використання, яке зосереджується на інтерфейсі користувача та загальному досвіді користувача. Позитивний досвід

користувача є важливим для успіху програмного продукту, а тестування допомагає виявити та покращити проблеми зручності використання [17].

Загалом, тестування – це ітеративний процес, який дозволяє розробникам постійно вдосконалювати програмне забезпечення. Відгуки від тестування допомагають удосконалити код, покращити функції та оптимізувати загальну продуктивність програмного забезпечення.

Таким чином, тестування програмного забезпечення має важливе значення для надання високоякісних, надійних і безпечних програмних продуктів, які відповідають очікуванням користувачів, відповідають стандартам і сприяють загальному успіху проєкту розробки.

Перш ніж розпочати створення автоматизованої системи тестування, необхідно сформулювати комплексний набір тестів, оскільки вони відіграють ключову роль у формулюванні автоматизованих тестів. Важливо оцінити реакцію системи як на позитивні, так і на несприятливі сценарії, наприклад випадки введення помилкових даних. Відповідно, для кожного окремого модуля були сформульовані окремі тести.

Таблиця 3.1 містить дані про перевірку реєстрації нового користувача системи, а саме перевірку правильності усіх заповнених полів, необхідних для створення облікового запису.

Таблиця 3.1 – Тесткейс «Реєстрація нового користувача, позитивний сценарій»

№	Крок	Очікуваний результат
1	Відкрити сторінку «Login»	Відображається сторінка «Login»
2	Натиснути на кнопку «Register here»	Користувача перенаправлено на сторінку «Sign Up»
3	Заповнити поле «Username» коректними даними	В полі «Username» відображено введені дані
4	Заповнити поле «Email Address» коректними даними	В полі «Email Address» відображено введені дані

Продовження табл. 3.1

№	Крок	Очікуваний результат
5	Заповнити поле «Password» коректними даними	В полі «Password» відображено введені дані
6	Заповнити поле «Confirm Password» коректними даними	В полі «Confirm Password» відображено введені дані
7	Натиснути на кнопку «Register»	Користувача перенаправлено на сторінку авторизації, над полем «Login» написано «Register complete!»

Таблиця 3.2 містить перевірку авторизації зареєстрованого користувача системи, тобто правильність логіну та паролю.

Таблиця 3.2 – Тесткейс «Авторизація користувача»

№	Крок	Очікуваний результат
1	Відкрити сторінку «Login»	Відображається сторінка «Login»
2	Ввести дані в поле «Username» розміром більше 30 символів.	Над полем «Username» червоним текстом відображено помилку «Username is too long»
3	Ввести дані в поле «Username» коректної довжини.	В полі «Username» відображено введені дані, помилки не відображаються
4	Ввести дані в поле «Password», несумісні з полем «Username»	В полі «Password» відображено введені дані
6	Натиснути кнопку «Login»	Над кнопкою «Login» відображено помилку «Incorrect login data!»
5	Заповнити поле «Password» коректними даними	В полі «Password» відображено введені дані
7	Натиснути кнопку «Login»	Користувача перенаправлено на сторінку головну сторінку застосунку. Авторизацію завершено.

Завдяки автоматизованому тестуванню тести можна виконувати набагато швидше, ніж вручну, що дає змогу швидше отримати відгук про якість програмного забезпечення. Також автоматичні тести зменшують ймовірність людських помилок, таких як пропуск тестових випадків або помилки під час виконання тесту.

Автоматичні тести можуть охоплювати велику кількість тестів, забезпечуючи більш ретельну перевірку програмного забезпечення порівняно з ручним тестуванням.

На рисунку 3.15 продемонстровано частину коду із автоматизованим тестуванням функцій add_like() та remove_like() із файлу «models.py».

```
12  class NetworkAppTests(TestCase):
13      def setUp(self):
14          # Set up test data (e.g., create a user)
15          self.user = User.objects.create_user(username='testuser', password='testpass')
16
17      def test_add_like(self):
18          post = Post.objects.create(content="Test post", user=self.user)
19          response = add_like(self._create_request(self.user), post.id)
20          self.assertEqual(response.status_code, 200)
21          self.assertEqual(json.loads(response.content)['message'], 'Like added')
22
23      def test_remove_like(self):
24          post = Post.objects.create(content="Test post", user=self.user)
25          like = Like.objects.create(user=self.user, post=post)
26          response = remove_like(self._create_request(self.user), post.id)
27          self.assertEqual(response.status_code, 200)
28          self.assertEqual(json.loads(response.content)['message'], 'Like removed')
29
30      # Add tests for other functions
31
32      def _create_request(self, user):
33          # Helper method to create a mock request
34          request = self.client.post(reverse('login'), {'username': user.username, 'password': 'testpass'})
35          request.user = user
36          return request
```

Рисунок 3.15 – Фрагмент коду із файлу «tests.py»

І хоча автоматизоване тестування пропонує численні переваги, важливо зазначити, що воно може не підходити для кожного сценарію тестування, і для комплексного забезпечення якості часто рекомендується збалансований підхід, який включає як автоматичне, так і ручне тестування.

ВИСНОВКИ

Під час роботи над розробкою вебзастосунку соціальної мережі «Liaise» було отримано завдання на розробку, проведено дослідження схожих проєктів, таких як «Facebook», «Instagram», «X», а також огляд засобів розробки інформаційної системи. Наведено порівняльний аналіз технологій Python та JavaScript.

У другому розділі спроєктовано структуру застосунку, описано основні принципи роботи програмного забезпечення, які було проілюстровано відповідними діаграмами класів, послідовності та прецедентів. Також додано ER-діаграму бази даних вебзастосунку.

Третій розділ включає у себе програмну реалізацію клієнтської та серверної частин і тестування розробленої соціальної мережі, які є завершальними етапами процесу розробки. При розробці клієнтської частини реалізовано інтерфейс вебзастосунку засобами HTML, CSS, вебфреймворку Django і мови JavaScript. Серверна частина розроблена за допомогою Django і бази даних SQLite. Проведено ручне тестування застосунку і автоматизоване тестування розроблених функцій.

Результатом проведеного у даній роботі дослідження є розробка вебзастосунку соціальної мережі «Liaise» з використанням Python та JavaScript. Вплив соціальних мереж продовжує зростати разом із розвитком технологій. Це впливає на те, як ми зв’язуємося, спілкуємося та співпрацюємо в різних аспектах нашого особистого та професійного життя. Тому результати роботи можуть бути використані у дослідженні міжособистісних стосунків між людьми чи групами людей, вивченні соціальної поведінки та взаємодій, аналізі тенденцій і закономірностей поведінки людини. Також результати розробки можуть сприяти поширенню та просуванню контенту, новин та інформації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Лутаєва О. М., Кривохата А. Г. Розробка системи комунікації «Liaise» засобами Python та JavaScript. *Актуальні проблеми математики та інформатики*: збірка тез та доповідей Чотирнадцятої Всеукраїнської, двадцять першої регіональної наукової конференції молодих дослідників. (Запоріжжя, 27-28 квітня 2023). Запоріжжя: ЗНУ, 2023. С. 62.
2. Sagar Poudel. Dynamic website with Django. URL: https://faun.pub/dynamic-website-with-django-55db56c1068a (дата звернення 12.08.2023).
3. Hack – The Language Behind Facebook. URL: https://www.hongkiat.com/blog/hack-language-facebook/ (дата звернення 12.08.2023).
4. Соціальна мережа «Facebook». URL: https://www.facebook.com/ (дата звернення: 14.08.2023).
5. Соціальна мережа «Instagram». URL: https://www.instagram.com/ (дата звернення: 14.08.2023).
6. Famous Websites & Apps Programmed in What Languages? URL: https://www.hackersclub.io/single-post/2016-1-26-famous-websites-apps-programmed-in-what-languages (дата звернення: 15.08.2023).
7. Соціальна мережа «X». URL: https://twitter.com/ (дата звернення: 15.08.2023).
8. Charles Humble. Twitter Shifting More Code to JVM, Citing Performance and Encapsulation As Primary Drivers. URL: https://www.infoq.com/articles/twitter-java-use/ (дата звернення: 16.08.2023).
9. UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples. URL: https://creately.com/blog/diagrams/uml-diagram-types-examples/ (дата звернення: 19.08.2023).
10. Mastering the Basics of JavaScript Web Development. URL: https://www.appypie.com/javascript-web-development (дата звернення:

03.09.2023).

11. Django. URL: https://www.studysmarter.co.uk/explanations/computer-science/computer-network/django/ (дата звернення: 13.09.2023).
12. Структура сайту: основні види та правила їх розробки. URL: https://webtune.com.ua/statti/web-rozrobka/struktura-sajtu/ (дата звернення 24.09.2023).
13. Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 27.09.2023).
14. ER-діаграма. Опис, види, правила побудови. URL: http://hi-news.pp.ua/kompyuteri/14668-er-dagrama-ce-opis-vidi-pravila-pobudovi.html (дата звернення 29.09.2023).
15. An Entity Relationship Diagram Example. URL: https://www.leerichardson.com/2007/06/entity-relationship-diagram-example.html (дата звернення 30.09.2023).
16. Making queries. Django documentation. URL: https://docs.djangoproject.com/en/4.2/topics/http/views/ (дата звернення 06.10.2023).
17. Aaron Thomas. Compliance Testing: Everything You Need to Know in 2023. URL: https://testsigma.com/blog/compliance-testing/ (дата звернення 10.10.2023).

ДОДАТОК А

Лістинг програмного коду «views.py»

```
from django.contrib.auth import authenticate, login, logout
from django.db import IntegrityError
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render
from django.urls import reverse
from django.core.paginator import Paginator
import json
from django.http import JsonResponse
from .models import User, Post, Follow, Like
def remove_like(request, post_id):
    post = Post.objects.get(pk=post_id)
    user = User.objects.get(pk=request.user.id)
    like = Like.objects.filter(user=user, post=post)
    like.delete()
    return JsonResponse({"message": "Like removed"})
def add_like(request, post_id):
    post = Post.objects.get(pk=post_id)
    user = User.objects.get(pk=request.user.id)
    newLike = Like(user=user, post=post)
    newLike.save()
    return JsonResponse({"message": "Like added"})
def edit(request, post_id):
    if request.method == "POST":
        data = json.loads(request.body)
        edit_post = Post.objects.get(pk=post_id)
        edit_post.content = data["content"]
```

```
        edit_post.save()
        return JsonResponse({"message": "Change successful", "data": data["content"]})
def index(request):
    allPosts = Post.objects.all().order_by("id").reverse()
    # Pagination
    paginator = Paginator(allPosts, 10)
    page_number = request.GET.get('page')
    posts_of_the_page = paginator.get_page(page_number)
    allLikes = Like.objects.all()
    whoYouLiked = []
    try:
        for like in allLikes:
            if like.user.id == request.user.id:
                whoYouLiked.append(like.post.id)
    except:
        whoYouLiked = []
    return render(request, "network/index.html", {
        "allPosts": allPosts,
        "posts_of_the_page": posts_of_the_page,
        "whoYouLiked": whoYouLiked
    })
def newPost(request):
    if request.method == "POST":
        content = request.POST['content']
        user = User.objects.get(pk=request.user.id)
        post = Post(content=content, user=user)
        post.save()
        return HttpResponseRedirect(reverse(index))
def profile(request, user_id):
```

```
    user = User.objects.get(pk=user_id)
    allPosts = Post.objects.filter(user=user).order_by("id").reverse()
    following = Follow.objects.filter(user=user)
    followers = Follow.objects.filter(user_follower=user)
    try:
        checkFollow = followers.filter(user=User.objects.get(pk=request.user.id))
        if len(checkFollow) !=0:
            isFollowing = True
        else:
            isFollowing = False
    except:
        isFollowing = False
    # Pagination
    paginator = Paginator(allPosts, 10)
    page_number = request.GET.get('page')
    posts_of_the_page = paginator.get_page(page_number)
    return render(request, "network/profile.html", {
        "allPosts": allPosts,
        "posts_of_the_page": posts_of_the_page,
        "username": user.username,
        "following": following,
        "followers": followers,
        "isFollowing": isFollowing,
        "user_profile": user
    })
def following(request):
    currentUser = User.objects.get(pk=request.user.id)
    followingPeople = Follow.objects.filter(user=currentUser)
    allPosts = Post.objects.all().order_by('id').reverse()
    followingPosts = []
```

```
    for post in allPosts:
        for person in followingPeople:
            if person.user_follower == post.user:
                followingPosts.append(post)
    # Pagination
    paginator = Paginator(followingPosts, 10)
    page_number = request.GET.get('page')
    posts_of_the_page = paginator.get_page(page_number)
    return render(request, "network/following.html", {
        "posts_of_the_page": posts_of_the_page
    })
def follow(request):
    userfollow = request.POST['userfollow']
    currentUser = User.objects.get(pk=request.user.id)
    userFollowData = User.objects.get(username=userfollow)
    f = Follow(user=currentUser, user_follower=userFollowData)
    f.save()
    user_id = userFollowData.id
    return HttpResponseRedirect(reverse(profile, kwargs={'user_id': user_id}))
def unfollow(request):
    userfollow = request.POST['userfollow']
    currentUser = User.objects.get(pk=request.user.id)
    userFollowData = User.objects.get(username=userfollow)
    f = Follow.objects.get(user=currentUser, user_follower=userFollowData)
    f.delete()
    user_id = userFollowData.id
    return HttpResponseRedirect(reverse(profile, kwargs={'user_id': user_id}))
def login_view(request):
    if request.method == "POST":
        # Attempt to sign user in
```

```
        username = request.POST["username"]
        password = request.POST["password"]
        user = authenticate(request, username=username, password=password)
        # Check if authentication successful
        if user is not None:
            login(request, user)
            return HttpResponseRedirect(reverse("index"))
        else:
            return render(request, "network/login.html", {
                "message": "Invalid username and/or password."
            })
    else:
        return render(request, "network/login.html")
def logout_view(request):
    logout(request)
    return HttpResponseRedirect(reverse("index"))
def register(request):
    if request.method == "POST":
        username = request.POST["username"]
        email = request.POST["email"]
        # Ensure password matches confirmation
        password = request.POST["password"]
        confirmation = request.POST["confirmation"]
        if password != confirmation:
            return render(request, "network/register.html", {
                "message": "Passwords must match."
            })
        # Attempt to create new user
        try:
            user = User.objects.create_user(username, email, password)
```

```
            user.save()
        except IntegrityError:
            return render(request, "network/register.html", {
                "message": "Username already taken."
            })
        login(request, user)
        return HttpResponseRedirect(reverse("index"))
    else:
        return render(request, "network/register.html")
```

ДОДАТОК Б

Лістинг програмного коду index.html

```
{% extends "network/layout.html" %} {% block body %}
<script>
  function getCookie(name) {
    const value = `; ${document.cookie}`;
    const parts = value.split(`; ${name}=`);
    if (parts.length == 2) return parts.pop().split(";").shift();
  }
  function submitHandler(id) {
    const textareaValue = document.getElementById(`textarea_${id}`).value;
    const content = document.getElementById(`content_${id}`);
    const model = document.getElementById(`modal_edit_post_${id}`);
    fetch(`/edit/${id}`, {
      method: "POST",
      headers: {
        "Conten-type": "application/json",
        "X-CSRFToken": getCookie("csrftoken"),
      },
      body: JSON.stringify({
        content: textareaValue,
      }), })
      .then((response) => response.json())
      .then((result) => {
        content.innerHTML = result.data;
        modal.classList.remove("show");
        modal.setAttribute("aria-hidden", "true");
        modal.setAttribute("style", "display: none");
```

```
      const modalsBackdrops =
        document.getElementsByClassName("modal-backdrop");
      for (let i = 0; i < modalsBackdrops.length; i++) {
        document.body.removeChild(modalsBackdrops[i]);
      } }); }
function likeHandler(id, whoYouLiked) {
  const btn = document.getElementById(`${id}`);
  let liked;
  fetch(`/getLikes/${id}`)
    .then((response) => response.json())
    .then((result) => {
      liked = result.answer;
      if (liked) {
        fetch(`/remove_like/${id}`)
          .then((response) => response.json())
          .then((result) => {
            console.log(result);
            btn.classList.remove("fa-heart");
            btn.classList.add("fa-heart-o");
            liked = false;
          });
      } else {
        fetch(`/add_like/${id}`)
          .then((response) => response.json())
          .then((result) => {
            console.log(result);
            btn.classList.remove("fa-heart-o");
            btn.classList.add("fa-heart");
            liked = true;
          });
```

```
      }
    });
  }
</script>
<h1>all posts</h1>
{% if user.is_authenticated %}
<div class="new-post">
  <form action="{% url 'newPost' %}" method="post">
    {% csrf_token %}
    <textarea placeholder="what's new?" name="content" row="4"
cols="50"></textarea>
    <br />
    <input type="submit" value="Post" class="btn btn-primary btnpost"  />
  </form>
</div>
{% endif %}
<div class="all-posts">
  {% for post in posts_of_the_page %}
  <div class="row post col-4">
    <h5 class="username">
      <a href="{% url 'profile' user_id=post.user.id %}">@{{ post.user }}</a>
    </h5>
    <h6 class="content" id="content_{{ post.id }}">{{ post.content }}</h6>
    <p class="date">{{ post.date }}</p>
    {% if user.is_authenticated %} {% if user == post.user %}
    <div class="d-flex justify-content-around">
      <button
        class="btn btn-primary"
        data-toggle="modal"
        data-target="#modal_edit_post_{{ post.id }}">
```

```
        Edit
      </button>
    </div>
    <div
      class="modal fade"
      id="modal_edit_post_{{ post.id }}"
      tabindex="-1"
      role="dialog"
      aria-labelledby="modal_edit_post_{{ post.id }}_label"
      aria-hidden="true">
      <div class="modal-dialog" role="document">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title">Edit Post</h5>
            <button
              type="button"
              class="close"
              data-dismiss="modal"
              aria-label="Close">
              <span aria-hidden="true">&times;</span>
            </button>
          </div>
          <div class="modal-body">
            <textarea
              row="5"
              id="textarea_{{ post.id }}"
              class="form-control"
              name="content">
{{ post.content }}</textarea>
          </div>
```

```
      <div class="modal-footer">
        <button
          type="button"
          class="btn btn-primary"
          onclick="submitHandler({{ post.id }})">
          Save changes
        </button>
        <button
          type="button"
          class="btn btn-secondary"
          data-dismiss="modal">
          Close
        </button>
      </div>
    </div>
  </div>
</div>
{% else %} {% if post.id in whoYouLiked %}
<!-- This means that we already liked the post -->
<div class="d-flex justify-content-around">
<button
  class="btn btn-info fa fa-heart-o"
  onclick="likeHandler({{ post.id }}, {{ whoYouLiked }})"
  id="{{ post.id }}"
></button></div>
{% else %}
<div class="d-flex justify-content-around">
<button
  class="btn btn-info fa fa-heart"
  onclick="likeHandler({{ post.id }}, {{ whoYouLiked }})"
```

```
      id="{{ post.id }}"
    ></button></div>
    {% endif %} {% endif %} {% endif %}
  </div>
  {% endfor %}
</div>
<nav aria-label="Page navigation example">
  <ul class="pagination d-flex justify-content-center">
    {% if posts_of_the_page.has_previous %}
    <li class="page-item">
      <a
        class="page-link"
        href="?page={{ posts_of_the_page.previous_page_number }}"
        >Previous</a>
    </li>
    {% endif %} {% if posts_of_the_page.has_next %}
    <li class="page-item">
      <a class="page-link" href="?page={{ posts_of_the_page.next_page_number }}"
        >Next</a>
    </li>
    {% endif %}
  </ul>
</nav>
{% endblock %}
```

ДОДАТОК В

Лістинг програмного коду models.py

```
from django.contrib.auth.models import AbstractUser
from django.db import models
class User(AbstractUser):
    pass
class Post(models.Model):
    content = models.CharField(max_length=140)
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="author")
    date = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return f"Post {self.id} made {self.user} on {self.date.strftime('%d %b %Y %H:%M:%S')}"
class Follow(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="user_who_is_following")
    user_follower = models.ForeignKey(User, on_delete=models.CASCADE, related_name="user_who_is_being_followed")
    def __str__(self):
        return f"{self.user} is following {self.user_follower}"
class Like(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="user_like")
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name="post_like")
    def __str__(self):
        return f"{self.user} liked {self.post}"
```