ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків Spring та Nextjs»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Іванна АНДРІЙЧУК
(підпис)

Виконав: здобувачка вищої освіти групи ПД-43

Іванна АНДРІЙЧУК

Керівник: Світлана ШЕВЧЕНКО
к.п.н., доц.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Андрійчук Іванні Ростиславівні

1. Тема кваліфікаційної роботи: «Розробка клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків Spring та Nextjs»
керівник кваліфікаційної роботи к.п.н., доц., доцент кафедри ІПЗ Світлана ШЕВЧЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про проблему інформаційної бульбашки, опис алгоритмів фільтрації та їхній вплив на дану проблему, опис стратегій її пом’якшення.

4. Зміст розрахунково-пояснювальної записки:
1. Провести аналіз літературних джерел з проблеми алгоритмічної фільтрації підбору контенту у соціальних мережах, визначити та описати вплив алгоритмів фільтрації на проблему інформаційної бульбашки, оглянути запропоновані варіанти її розв'язання.

2. Здійснити огляд та аналіз існуючих соціальних мереж, що використовуються для розповсюдження фотографій, визначити їх переваги та недоліки.
3. Провести огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій.
4. Сформувати вимоги до розробки клієнт-серверного застосунку.
5. Спроектувати архітектуру клієнт-серверного застосунку соціальної мережі.
6. Розробити клієнт-серверний застосунок соціальної мережі для розповсюдження фотографій.
7. Провести тестування клієнт-серверного застосунку соціальної мережі.

5. Перелік графічного матеріалу:
1. Титульний слайд.
2. Мета, об’єкт та предмет роботи
3. Аналіз аналогів
4. Задачі дипломної роботи
5. Аналіз аналогів
6. Вимоги до додатку
7. Програмні засоби реалізації
8. Діаграма варіантів використання
9. Діаграми послідовності авторизації
10. Структура бази данних
11. Архітектура додатку
12. Діаграма класів основної логіки
13. Діаграма класів авторизації
14. Мапи веб-застосунку
15. Екранні форми
16. Демонстрація роботи програми
17. Апробація результатів дослідження
18. Висновки

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження програм аналогів	07.03-13.03.2024	
3	Аналіз засобів реалізації	14.03-21.03.2024	
4	Проектування архітектури системи	22.03-28.03.2024	
5	Розробка програмного забезпечення	29.03-22.04.2024	
6	Тестування системи	23.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувачка вищої освіти _____ Іванна АНДРІЙЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Світлана ШЕВЧЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра:50 стор., 21 табл., 31 рис., 25 джерел.

Мета роботи – покращення процесу розповсюдження фотографій за допомогою клієнт-серверного застосунку соціальної мережі.

Об'єкт дослідження – процес розповсюдження фотографій за допомогою соціальних мереж.

Предмет дослідження – клієнт-серверний застосунок для розповсюдження фотографій.

*Короткий зміст роботи***:** В роботі проведено аналіз літературних джерел на тему роботи соціальних мереж. Проведено огляд та аналіз існуючих соціальних мереж, що використовуються для розповсюдження фотографій. Однією із основних проблем існуючих додатків є проблема інформаційної бульбашки. Визначено та описано вплив алгоритмів фільтрації на дану проблему та проаналізовано стратегії її пом'якшення.

Було наведено огляд технологій, що були використані для розробки клієнт-серверного застосунку. Основними технологіями є мова програмування Java, та бекенд фреймворк Spring у поєднанні з об'єктно-реляційною системою керування базами даних PostgreSQL, та хмарним сховищем Amazon S3 для розробки бекенду додатку. Також, мова програмування TypeScript та фреймворк Next.js для розробки фронтенду додатку. Окрім цього, було описано архітектуру програмного застосунку та структру інтерфейсу користувача.

Результатом роботи є додаток соціальної мережі з можливістю прозорого вибору способу фільтрації контенту. Також, було проведено тестування програмного застосунку, для підтвердження його роботоспроможності.

Цільовою аудиторією додатку є люди будь-якого віку та інтересів, які цінують можливості усвідомленого вибору контенту в свому інфо просторі.

Наукова новизна - демонстрація можливості реалізації соціальної мережі з прозорим вибором фільтрів для пом’якшення проблеми інформаційної бульбашки.

КЛЮЧОВІ СЛОВА: СОЦІАЛЬНА МЕРЕЖА, ІНФОРМАЦІЙНА БУЛЬБАШКА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

REST – Representational State Transfer

API – Application Programming Interface

JSON – Java Script Object Notation

JWT – JSON Web Token

Amazon S3 – Amazon Simple Storage Service

VS Code – Visual Studio Code

CSRF – Міжсайтова підробка запитів

JPA – Java Persistence API

DAL – Data Access Layer

BSL – Business Service Layer

API – Application Programming Interface

SPSP – Simple Photo Sharing Platform

ВСТУП

Сучасний світ характеризується стрімким розвитком в галузі інформаційних технологій, що супроводжується швидким поширенням соціальних мереж. Одним із основних способів вираження, та спілкування у соціальних мережах є розповсюдження фотографій. Однак, існуючі соціальні мережі не завжди можуть задовольнити вимоги та потреби своїх користувачів. Це викликає необхідність пошуку нових рішень проблеми якісного надання контенту користувачам.

У ході розвитку галузі соціальних мереж постала проблема надмірного використання персоналізованої та групової алгоритмічної фільтрації контенту, що у результаті сприяло погіршенню проблеми інформаційної бульбашки. Дана проблема обмежує доступ користувачів до різних спектрів інформації, чим зменшує різноманітність їхніх поглядів, та інтересів. Це призводить до зростання поляризації і посилення впливу дезінформації, зменшуючи якість інформації. Результатом цього є радикалізація поглядів користувачів, що спричиняє у тому числі і соціально-політичні проблеми.

На сьогоднішній день все більше користувачів та організацій розуміють негативні наслідки подібної фільтрації контенту, що створює новий ринок для соціальних мереж, що не використовують персоналізацію у алгоритмічній фільтрації контенту, з наданням прозорого доступу до вибору фільтрів контенту, чим зменшують проблему інформаційної бульбашки, покращуючи доступ користувачів до різноманітної інформації різного спектру інтересів.

Мета роботи – покращення процесу розповсюдження фотографій за допомогою клієнт-серверного застосунку соціальної мережі.

Об'єкт дослідження – процес розповсюдження фотографій за допомогою соціальних мереж.

Предмет дослідження – клієнт-серверний застосунок для розповсюдження фотографій.

Перший розділ надає загальний огляд соціальних мереж для розповсюдження фотографій, та проблеми інформаційної бульбашки. У цьому розділі наводиться аналіз досліджень алгоритмів фільтрації та їхнього впливу на проблему інформаційної бульбашки та виділено стратегії пом’якшення даної проблеми.

Другий розділ присвячено проектуванню архітектури майбутнього додатку. Також, зроблено моделювання вимог та огляд засобів реалізації.

Третій розділ описує реалізацію клієнт-серверного програмного застосунку. В даному розділі надається діаграма основних класів серверу, мапи сайту авторизованого та неавторизованого користувача, та зображення готового додатку.

Четвертий розділ описує тестування функціоналу серверу застосунку за допомогою програмного рішення Postman. Крім цього, проведено загальне тестування додатку, через проходження тест-кейсів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Сучасний світ характеризується стрімким розвитком в галузі інформаційних технологій, що супроводжується швидким поширенням соціальних мереж. Протягом останнього десятиліття соціальні мережі стали невід'ємною частиною повсякденного життя сучасної людини. Вони відіграють ключову роль у забезпеченні комунікації, формуванні віртуальних спільнот, обміні інформацією та відстежуванні останніх локальних та глобальних події, також надаючи змогу стежити за трендами та залишатися у стрімкому інформаційному потоці сучасного світу.

1.1 Актуальність

Соціальні мережі є одним із найпоширеніших і найпопулярніших цифрових видів діяльності. Кількість користувачів соціальних мереж постійно зростає. Станом на квітень 2024 року, глобальний рівень використання соціальних мереж становив 62,2% [1]. За прогнозами, глобальна кількість користувачів зросте від 4,89 мільярда осіб в 2023 році до шести мільярдів ще до початку 2027 року [2]. Більше того, станом на 2023 рік щоденне використання соціальних мереж інтернет-користувачами в усьому світі становило 151 хвилину на день [3].

Потрібно відзначити, що собливе місце у світі соціальних займає публікація та розповсюдження фотографій. Цей тип інформаційної взаємодії є одним із найзручніших та найпопулярніших, створюючи можливість легкого та швидкого вираження та спілкування. Саме тому зображення зайняли друге місце, після коротких відео, в рейтингу найбільш привабливих типів контенту для маркетингу в соціальних мережах в США за квітень 2022 року, набравши 61% голосів [4].

1.2 Загальний аналіз та характеристика соціальної мережі

Соціальні мережі дозволяють користувачам спілкуватися, обмінюватися інформацією, створювати та споживати контент, взаємодіяти з іншими

користувачами. Спілкування в соціальних мережах може включати в себе публікації, коментарі, обмін повідомленнями. Варто зазначити, що соціальні мережі можуть мати різноманітні функції та спеціалізації, такі як професійні мережі для бізнесу, медійні мережі для спільного споживання контенту або спільноти для обговорення специфічних тем. Зазвичай, соціальна мережа є web-додатком.

Основною цільовою аудиторією соціальних мереж для розповсюдження фотографій є звичайні люди, які хочуть ділитися своїми фотографіями з друзями, родиною та іншими користувачами мережі. Ця аудиторія може бути різноманітною за віком, інтересами, професією та іншими характеристиками. Вони можуть використовувати соціальну мережу для вираження своєї творчості, спілкування з іншими користувачами, взаємодії з контентом та отримання відгуків на свої роботи. Для них важливо мати простий і зручний інтерфейс для завантаження та організації своїх фотографій, а також можливість перегляду та взаємодії з контентом інших користувачів. Також цільовою аудиторією таких додатків можуть виступати професійні фотографи або фотографи-аматори, що перебувають в пошуках місця для показу своїх робіт, щоб отримати відгуки та поради від інших користувачів, а також взяти участь у спільноті фотографів. Вони можуть використовувати мережу для просування свого бренду та реклами своїх послуг, залучення клієнтів.

Одним із основних завдань соціальної мережі є підбір контенту. Для цього вони часто використовуються різноманітні стратегії для підбору контенту та алгоритмічна фільтрація для кожного користувача. Ось деякі з найпоширеніших стратегій:

- персоналізований контент, коли алгоритми аналізують історію перегляду, лайків, коментарів та інших дій окремого користувача, щоб зрозуміти їхні інтереси та вподобання;
- груповий контент, коли користувачів, на базі їх попередніх взаємодій з додатком неявно ділять на групи по інтересам і пропонують контент на базі приналежності до певної групи;

- тренди, коли системи враховують загальні тренди та популярний контент серед всіх користувачів;
- соціальні зв'язки, коли пропонується контент від друзів, фоловерів або груп;

Ці стратегії часто поєднуються для створення персоналізованого досвіду користувачів. Проте потрібно пам’ятати, що однією із головних проблем сучасних соціальних мереж є проблема інформаційної бульбашки, спричинена фільтрувальними алгоритмами підбору контенту. Тому підбір стратегій підбору контенту потребує обережного підходу, оскільки некоректне застосування може погіршити ситуацію з інформаційною бульбашкою.

1.3 Проблема інформаційної бульбашки

Через активну популяризацію соціальних мереж, виникає необхідність у постійному удосконаленні засобів для ефективного та зручного обміну інформацією, у тому числі і фотографіями. Однак, існуючі соціальні мережі не завжди в змозі задовольнити вимоги та потреби своїх користувачів. Одним із таких критичних факторів є проблема "інформаційної бульбашки". Дана проблема обмежує доступ користувачів до інформації різного спектру, адже система показує лише контент відповідний до попередніх взаємодій користувача з соціальною мережею. Це часто призводить до зростання поляризації і посилення впливу дезінформації, утворюючи так звану "інформаційну ехо-камеру".

Це зменшує довіру користувачів і дедалі частіше змушує їх поступово відмовлятись від мереж через подібний тип впливу. Так, за даними опитування по всьому світу у лютому 2019 року, на думку онлайн-користувачів, соціальні мережі мають значний вплив не лише на онлайн-діяльність, але й на поведінку людей та їх життя загалом. Значна частка респондентів заявила, що соціальні мережі покращили їхній доступ до інформації, полегшили спілкування та сприяли свободі вираження думок. Водночас, респонденти також зазначили, що соціальні мережі погіршили їх особисту конфіденційність, посилили поляризацію в політиці та щоденні відволікання [5].

1.4 Алгоритми фільтрації та їхній вплив на проблему інформаційної бульбашки.

Контент і спосіб його подання становлять основу будь-якої соціальної мережі. Це безпосередньо впливає на зацікавленість користувачів у використанні певної платформи та її конкурентоспроможність. Сучасні соціальні мережі використовують стратегії алгоритмічної фільтрації для стимулювання взаємодії користувачів. Використання персоналізації при використанні алгоритмічної фільтрації може призвести до утворення бульбашок фільтрів і підвищення рівня поляризації.

Було виявлено, що використання групових алгоритмів фільтрації призводить до загального підвищення рівня полярності [6], особливо в мережах з екстремістами. Було визначено, що люди, в першу чергу, взаємодіють з людьми зі своєї групи та використовують негативні стереотипи під час взаємодії з людьми із зовнішньої групи. Моделі динаміки думок повинні враховувати це.

Проблема бульбашок фільтрів, що полягає насамперед в зниженні якості публічного обговорення, а не в перешкоді доступу користувачів до бажаної інформації [7]. Поляризовані користувачі та користувачі, що споживають неякісну інформацію, мають викривлене сприйняття реальності. Зважаючи на це, найкращим способом мінімізації проблеми інформаційної бульбашки є відмова від використання персоналізованих алгоритмів фільтрації інформації.

Також, вплив бульбашки інформації на здоров’я, є досить складною та проблемною сферою [8]. Дана технологія створює проблеми як для пацієнтів, так і для лікарів, адже в даній сфері дезінформація може коштувати життя. Розв'язанням цієї проблеми може стати забезпечення можливості нефільтрованого пошуку [8].

На основі аналізу останніх досліджень і публікацій було виявлено тенденцію та чіткий запит користувачів на соціальні мережі з усвідомленим вибором контенту. Враховуючи це, було вирішено підтримати неупередженість у підборі інформації та не використовувати персоналізовані та групові алгоритми

фільтрації підбору контенту, реалізувавши нефільтрований підбір випадкового контенту.

1.5 Стратегії пом’якшення проблеми інформаційної бульбашки

Можна виділити наступні підходи зменшення впливу інформаційної бульбашки[9]:

- Підвищити прозорість роботи алгоритмів.
- Надати користувачам можливості налаштовувати алгоритми під свої потреби та інтереси.
- Розробити алгоритми, які пропонують контент що не базується на попередніх взаємодіях користувача.
- Розробити алгоритми, які забезпечують баланс між популярним і менш популярним, але важливим та якісним контентом.

Зважаючи на запропоновані вище варіанти зменшення впливу інформаційної бульбашки, було вирішено надати користувачам можливість самим обрати спосіб фільтрації їхнього контенту серед наявних та зробити способи фільтрації прозорими. Розробити алгоритм з випадковим підбором інформації для підвищення різноманіття поглядів.

1.6 Аналіз програмного забезпечення соціальної мережі для розповсюдження фотографій

При аналізі програмного забезпечення соціальних мереж для розповсюдження фотографій необхідно врахувати деякі важливі аспекти, а саме:

- функціональність додатку;
- оглянути інтерфейс користувача;
- приватність та безпека;
- наявність фільтрів персоналізованого підбору контенту;
- спосіб підбору контенту;
- прозорість роботи алгоритмів;

– можливості налаштування алгоритмів під свої потреби.

Аналіз цих аспектів допоможе побудувати чітку картину для розуміння роботи наявних програмних рішень, їх недоліків та переваг.

1.6.1 Instagram

Розглядаючи соціальні мережі для розповсюдження фотографій, не можна не згадати про Instagram, що в свій час став іконою фото-центричних мереж. За статистикою Instagram посідає третє місце в рейтингу найпопулярніших соціальних мережі у світі станом на квітень 2024 року за кількістю активних користувачів щомісяця, що становить близько 2 мільярдів користувачів [10].

Instagram [11] – це безкоштовна соціальна мережа, яка спеціалізується на обміні фотографіями і відеозаписами. Користувачі мають можливість завантажувати фотографії зі своєї камери або зберігати вже існуючі фотографії зі свого пристрою. Instagram пропонує широкий вибір фільтрів та інструментів для редагування фотографій, включаючи налаштування яскравості, контрасту, насиченості та тонування. Користувачі можуть також обрізати фотографії або застосовувати спеціальні ефекти. Після редагування фотографії користувач може додати опис та хештеги, щоб зробити свої публікації більш видимими. Фотографії, опубліковані в Instagram, можуть збирати вподобайки, коментарі та репости від інших користувачів (рисунок 1.1). Це створює можливості для спілкування та взаємодії між користувачами.

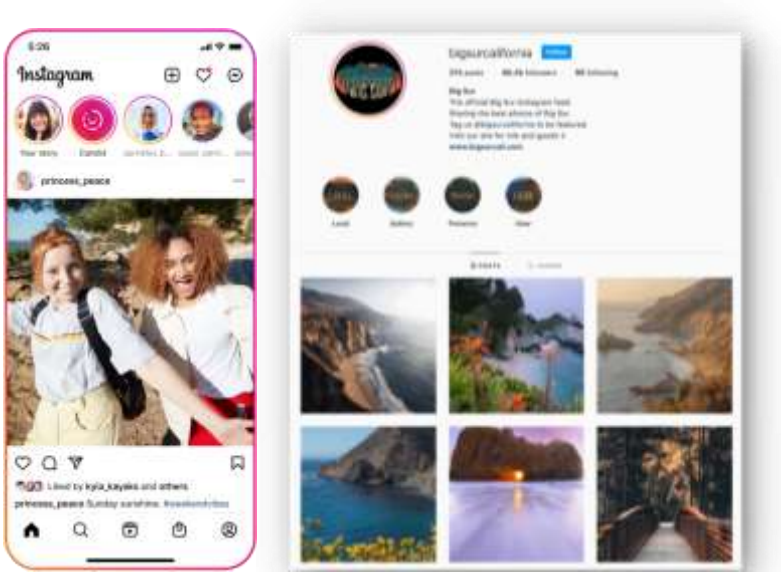


Рис. 1.1 Приклади екранних форм додатку Instagram

Розділ "Explore" в Instagram використовує алгоритми для показу користувачам випадкового контенту на основі їхньої попередньої взаємодії з платформою, вподобань та підписок. Це дозволяє знайти нові акаунти, фото та відео, які можуть бути цікавими користувачу. Функція Reels також використовує алгоритми для показу коротких відео, які можуть бути цікаві користувачу, базуючись на їхніх уподобаннях та поведінці на платформі.

Стрічка новин Instagram використовує безкінечну прокрутку у стрічці новин, дозволяючи користувачам переглядати пости своїх підписок безперервно.

Основними функціями Instagram є [12]:

- публікація фотографій та відео, їх редагування за допомогою вбудованих фільтрів та інструментів обробки;
- stories - фото та відео, які зникають через 24 години після публікації;
- мессенджер;
- search & explore - підбір контенту, на основі інтересів та попередніх взаємодій;
- live - транслювання відео в реальному часі для своїх підписників та спілкуватися з ними через коментарі.

До переваг даної мережі можна віднести:

- зосередженість на візуальному вмісті;
- велику базу користувачів;
- різноманітні функції залучення, такі як вподобайки, коментарі, прямі повідомлення та інтерактивні наклейки в Stories;
- різноманітність форматів взаємодії користувачів зі своєю аудиторією.

До недоліків даної мережі можна віднести:

- сприяння тривожності через представлення ідеалізованої версії життя користувачів, що ще більше підігрівається фільтрами, які орієнтовані на вибір візуально привабливого контенту;
- призводить до частого щоденного відволікання;
- спричиняє проблеми з фокусуванням;

- занепокоєння щодо конфіденційності даних, порушень безпеки та методів збору даних платформи;
- культура порівняння та конкуренції в Instagram, що підживлюється такими показниками, як вподобайки та підписники, може негативно вплинути на самооцінку та психічне благополуччя користувачів;
- через швидке поширення інформації без належної перевірки фактів може стати причиною поширення дезінформації.

1.6.2 VERO

Vero [13] - це соціальна мережа, яка пропонує різноманітні функції та акцентує на приватності та контролі користувачів над своїм контентом. Vero дозволяє користувачам переглядати контент, що публікують їхні друзі та вони самі, у хронологічному порядку. Це допомагає зберігати актуальність та динаміку. Користувачі можуть публікувати різноманітний контент, такий як фотографії, відео, текстові повідомлення, музику тощо на свої сторінки (рисунок 1.2). Vero надає користувачам більший контроль над приватністю свого контенту, дозволяючи обирати, хто може бачити кожен пост - всі або тільки певні категорії друзів (наприклад, близькі друзі, друзі, знайомі або абсолютно приватно) [14].

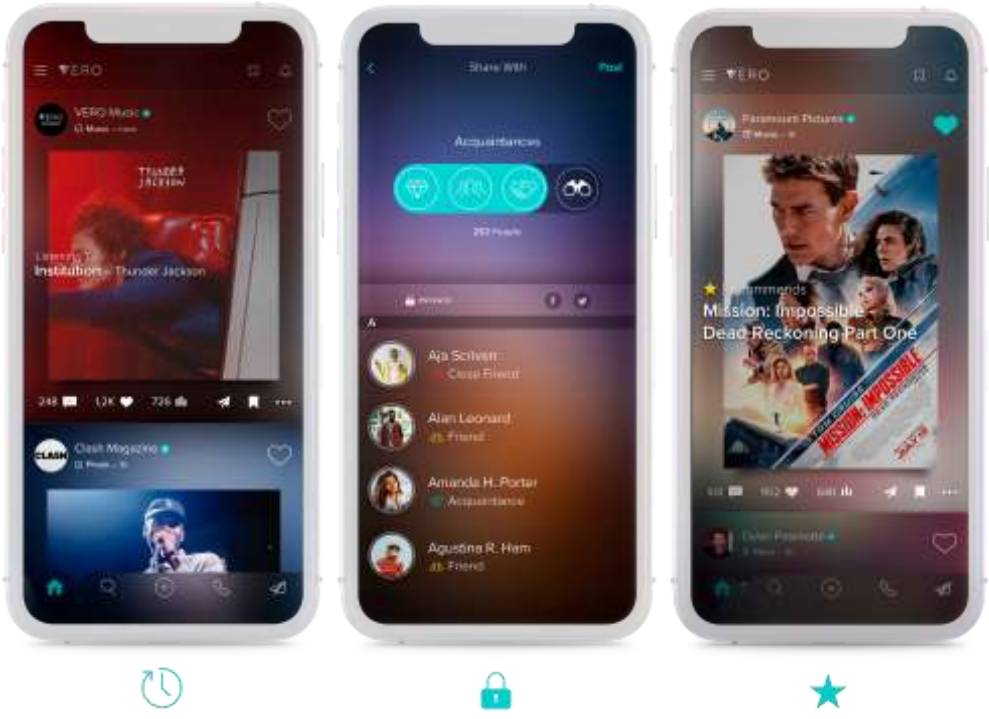


Рис. 1.2 Приклади екранних форм додатку Vero

Vero дозволяє користувачам створювати колекції, де вони можуть зберігати та категоризувати свій контент за темами чи інтересами. Більше того, його функція-моменти дозволяє користувачам збирати фотографії та відео в спеціальні альбоми, якими можна легко ділитися з друзями. Vero надає користувачам можливість ділитися зі своїми друзями рекомендаціями музики, книг і фільмів. Ці функції допомагають створювати особисті інтерактивні спільноти та зберігати контент у більш приватному та контрольованому середовищі порівняно з іншими соціальними мережами. Вміст у стрічці VERO показується в хронологічному порядку, що усуває потребу в алгоритмічному ранжуванні. Це дозволяє користувачам бачити пости у тому порядку, в якому вони були опубліковані, що забезпечує прозорість і передбачуваність контенту. Користувачі можуть використовувати функцію пошуку для знаходження контенту за різними категоріями та тегами. VERO відрізняється від більшості соціальних мереж тим, що не використовує алгоритмічні стрічки для показу контенту. Стрічка новин формується у хронологічному порядку, показуючи пости від людей, на яких користувач підписаний.

VERO може рекомендувати контент на основі категорій, таких як книги, фільми, музика, але це не випадковий контент у традиційному сенсі, оскільки користувачі самі вибирають, які категорії їх цікавлять. VERO яка використовує безкінечну прокрутку, дозволяючи користувачам переглядати пости від своїх підписок безперервно.

Переваги Vero:

- більший контроль над налаштуваннями конфіденційності порівняно з багатьма іншими платформами соціальних мереж. Користувачі можуть вибирати, хто бачитиме їхні публікації, класифікуючи свої зв’язки за різними рівнями інтимності;
- представлення вмісту у хронологічному порядку;
- категоризація контенту. Користувачі можуть впорядковувати свої публікації за категоріями, що полегшує керування вмістом і обмін ним на основі конкретних інтересів або тем;

- відсутність реклами;
- функції рекомендацій (можна рекомендувати друзям книги, фільми, музику та інший вміст, сприяючи взаємодії на основі спільних інтересів).

Недоліки:

- обмежена база користувачів;
- технічні проблеми, включаючи повільне завантаження, помилки та випадкові збої сервера;
- підписка з щорічною оплатою після початкового безкоштовного періоду;
- менш знайома та мало відома платформа.

Загалом, незважаючи на те, що Vero пропонує унікальні функції та зосередженість на конфіденційності, яка приваблює деяких користувачів, вона також стикається з проблемами щодо бази користувачів, технічної стабільності та стратегій монетизації.

1.6.3 Pixelfed

Pixelfed [15] - це децентралізована соціальна мережа для обміну фотографіями (рисунок 1.3), яка ставить перед собою мету надати користувачам контроль над їхнім контентом та приватністю. Одним з ключових аспектів Pixelfed є його децентралізована природа, що надає користувачам більший контроль над своїм контентом та приватністю, оскільки платформа не контролюється централізованим органом. Він працює за федеративною моделлю, що дозволяє користувачам вибирати власні екземпляри (сервери) або навіть розміщувати власні, зменшуючи залежність від центрального органу. Користувачі мають свободу вибирати різні екземпляри з різними правилами спільноти, темами та політикою модерації, що дозволяє їм знайти спільноту, яка відповідає їхнім інтересам і цінностям. Проте, налаштування екземплярів і керування ними, а також розуміння федерації та протоколів, таких як ActivityPub, можуть потребувати певних технічних знань. Також на роботу користувачів все одно можуть впливати надійність і політика інстанцій, до яких вони вирішили

приєднатися. Прикладом подібного є проблеми з модерацією або зміни в адмініструванні, які впливають на взаємодію з користувачем.

Pixelfed не покладається на отримання прибутку від реклами, використовуючи альтернативні джерела фінансування, такі як пожертви або преміум-функції. Також платформа не займається відстеженням даних або цільовою рекламою, що зменшує проблеми конфіденційності. Pixelfed є платформою з відкритим вихідним кодом. Це сприяє прозорості, співпраці спільноти та інноваціям.

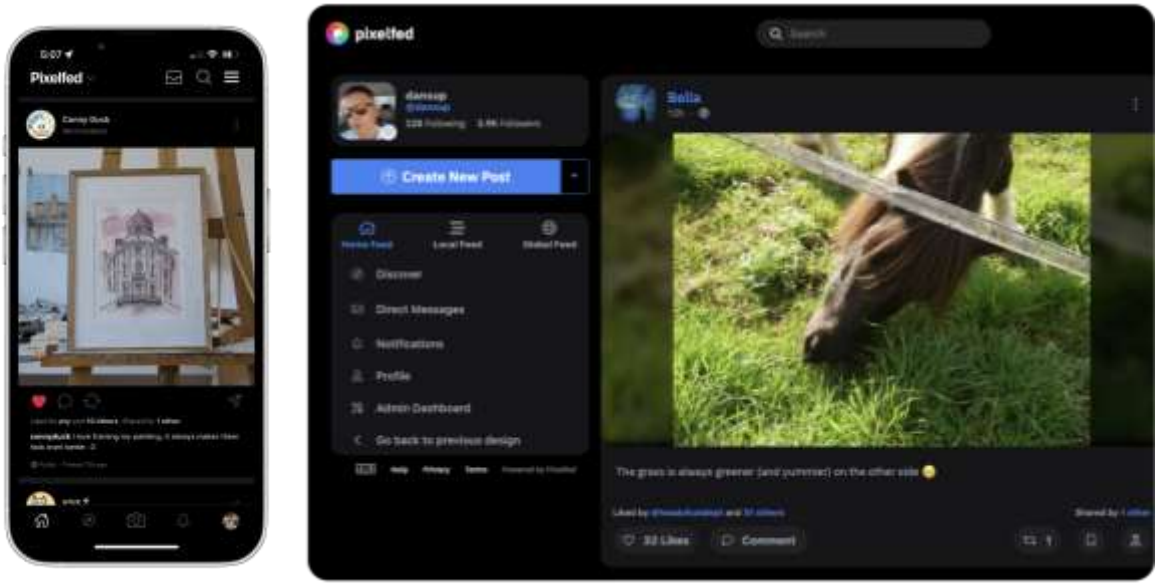


Рис. 1.3 Приклади екранних форм додатку Pixelfed

Користувачі цієї соціальної мережі можуть завантажувати фотографії у профіль, обираючи, хто має доступ до їхніх публікацій, та керуючи, як їхні фотографії та дані використовуються. Платформа надає можливість редагувати свої фотографії за допомогою фільтрів та ефектів, додавати теги до своїх публікацій для підвищення видимості.

Pixelfed має меншу базу користувачів і може не мати такого самого рівня залучення та різноманітності контенту. Користувачі, які звикли до широкого спектру функцій, можуть вважати пропозиції Pixelfed більш обмеженими. Pixelfed показує пости у хронологічному порядку, що усуває потребу в алгоритмічному ранжуванні. Узагальнюючи, Pixelfed має наступні основні функції:

– публікація фотографій;

- приватність та контроль доступу до публікацій;
- редагування фотографій за допомогою фільтрів, ефектів та інших інструментів;
- додавання тегів;
- коментування та лайки;
- підтримує завантаження та обмін відео та інших мультимедійних форматів;
- мобільні додатки для iOS та Android.

Переваги Pixelfed:

- Децентралізація;
- конфіденційність, що проявляється в контролі доступу до контенту і відсутності відстеження персональних даних для цільової реклами;
- свобода вибору в налаштуванні екземплярів (серверів);
- відсутність реклами;
- відкритий вихідний код.

Недоліки Pixelfed:

- обмежена база користувачів;
- налаштування екземплярів і керування ними потребують певних технічних знань;
- більш обмежений функціонал в порівнянні з іншими платформами;
- залежність від політики екземплярів, до яких було вирішено приєднатися;
- складність монетизації, адже монетизація платформи має відбуватися при дотриманні її принципів децентралізації та конфіденційності.

Загалом, Pixelfed пропонує багатообіцяючу альтернативу для користувачів, які шукають більшого контролю над своїм досвідом роботи в соціальних мережах і надають перевагу конфіденційності, але вона може потребувати певного рівня технічних знань для налаштування.

1.6.4 Зведені результати аналізу

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати аналізу характеристик соціальних мереж для розповсюдження фотографій

Показник	Instagram	VERO	Pixelfed	Майбутній додаток
Наявність Web-платформи	+	-	+	+
Хронологічне відображення контенту	-	+	Залежить від налаштувань	Залежить від налаштувань
Відсутність фільтрів персоналізованого підбору контенту	-	+	+	+
Прозорість підбору контенту	-	+	+	+
Легко налаштовується	+	+	-	+
Можливість відображення випадкового контенту	+	-	-	+
Основна цільова аудиторія	Поціновувачі візуального спілкування	Поціновувачі приватності та контролю над своїм контентом	Поціновувачі конфіденційності та децентралізації у соціальних мережах	Поціновувачі можливості усвідомленого вибору контенту

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Моделювання вимог

Проаналізувавши особливості роботи існуючих соціальних мереж, оглянувши стратегії алгоритмічної фільтрації підбору контенту, було сформульовано вимоги до майбутнього програмного продукту:

- можливість реєстрації та авторизація, розлогування;
- можливість роботи з профілем (редагування профілю, його видалення);
- можливість роботи з постами (створення, редагування та видалення);
- можливість перегляду профілів та публікацій інших користувачів;
- можливість взаємодіяти з іншими користувачами через систему підписок(друзів);
- можливість пошуку інших користувачів по імені користувача;
- можливість свідомого вибору фільтрів підбору контенту новинної стрічки (по друзям або випадковий підбір);
- англійська мова локалізації;
- захищеність від міжсайтової підробки запитів (Cross-Site Request Forgery, CSRF);
- зберігання паролів в зашифрованому вигляді.

Для спрощення реалізації було обрано англійську мову локалізації, так як це є основною мовою спілкування в мережі Інтернет.

В майбутньому вибір фільтрів буде розширено та додано можливість їх налаштувань.

Основний функціонал додатку представлено на діаграмі варіантів використання (рисунок 2.1)

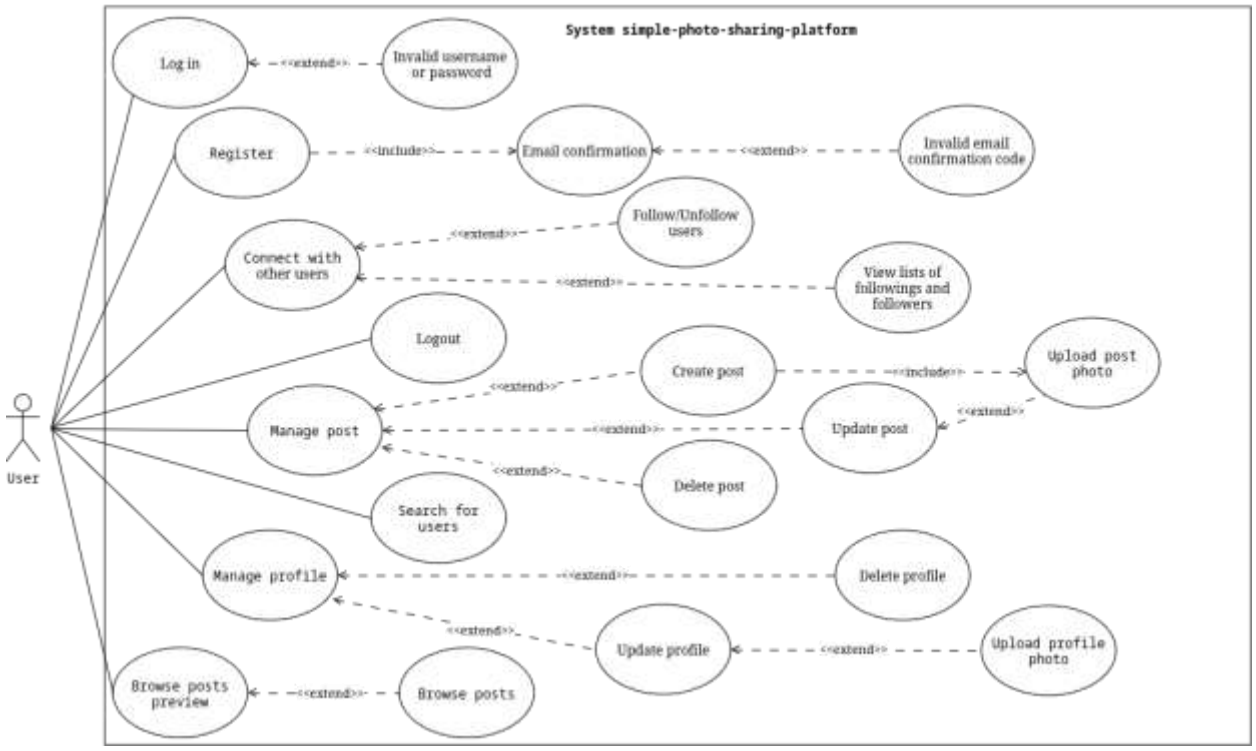


Рис. 2.1 Діаграма варіантів використання

2.2 Огляд засобів реалізації клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій

2.2.1 Серверна частина

Для реалізації серверної частини програмного застосунку було обрано мову програмування Java. Однією з ключових переваг Java є можливість виконувати код на будь-якому пристрої, який має встановлену відповідну віртуальну машину Java (JVM). Це робить Java платформо-незалежною. Java підтримує об'єктно-орієнтований підхід до програмування, що надає можливість організовувати свій код у вигляді класів та об'єктів. Java використовується в різних сферах, включаючи розробку веб-додатків, мобільних додатків (зокрема Android), вбудованих систем.

Java є однією з найпопулярніших мов програмування у світі і знаходить широке застосування в різних сферах програмного забезпечення. Згідно з PYPL, у березні 2024 року Java посідала друге місце за популярністю серед усіх мов програмування світу[17]. Також, за індексом TIOBE, який враховує кількість кваліфікованих інженерів у всьому світі, Java займає четверте місце [18]. Однією з

переваг вибору Java була наявність детальної документації, великої кількості ресурсів та її широке застосування для написання сучасних серверних рішень.

Для спрощення процесу розробки серверної частини був обраний один з найбільш надійних та універсальних серверних фреймворків Java - Spring. Він високо оцінюється при розробці корпоративних додатків Java завдяки своїй надійності, модульності, можливості впровадження залежностей, розширюваності та легкості інтеграції з іншими фреймворками. Spring, як легка та модульна платформа, сприяє модульності застосунку та надає можливість гнучкої конфігурації. Spring використовує принцип інверсії керування, що дозволяє виносити конфігураційні налаштування та управління залежностями з коду додатку, забезпечуючи модульність та надає можливість перевикористання коду.

Однією з найвидатніших переваг Spring є його висока масштабованість, яка досягається завдяки побічним проєктам, що він постачає, роблячи його одним із найкращих фреймворків для розробки бекенд-частини [19]. Так, Spring Boot є розширенням Spring, яке забезпечує швидку конфігурацію та створення автономних, готових до використання додатків, що спрощує розгортання та управління залежностями. Підпроєкт Spring Data надає зручні інструменти для роботи з різними джерелами даних, такими як бази даних та NoSQL-сховища. Модуль під назвою Spring Security призначений для забезпечення безпеки додатків. Він надає можливості аутентифікації, авторизації та управління доступом до ресурсів.

Також, перевагою Spring є його висока, модульність, що дозволяє використовувати тільки ті частини фреймворка, які потрібні для конкретного проєкту, спрощене управління залежностями за допомогою вбудованого контейнера IoC (Inversion of Control), який автоматично керує створенням та впровадженням об'єктів.

Для збереження інформації про користувача було використано базу даних PostgreSQL, а також сервіс зберігання об'єктів Amazon Simple Storage Service (Amazon S3) для збереження фотографій.

PostgreSQL — це потужна та надійна об'єктно-реляційна система керування базами даних з відкритим вихідним кодом. PostgreSQL надає широкий спектр можливостей для зберігання та обробки даних, включаючи підтримку складних структур даних, транзакцій, індексів та різноманітні інструменти для адміністрування та оптимізації баз даних [20]. PostgreSQL легко масштабується для роботи з великими обсягами даних та високими навантаженнями завдяки своїм розширенням та оптимізаціям. Ця база даних широко використовується у багатьох галузях, включаючи веб-розробку та аналітику даних, завдяки своїй надійності, масштабованості та гнучкості.

Amazon Simple Storage Service (Amazon S3) є одним з найоптимальніших варіантів для зберігання великих статичних файлів, таких як зображення. Amazon S3 забезпечує високу доступність та надійність. Він надає широкі можливості для захисту даних, включаючи шифрування, керування доступом на рівні об'єкту, аудит та моніторинг подій. S3 пропонує різні типи зберігання, включаючи стандартне для загального використання, зберігання для частого доступу та архівне для довгострокового зберігання. Цей сервіс відомий своєю вигідною ціновою політикою, оскільки вартість користування ним базується на фактичному використанні ресурсів, що робить його економічно вигідним рішенням для зберігання зображень. Модель ціноутворення Amazon S3, заснована на платі за фактичне використання пам'яті та пропускної здатності, дозволяє платити лише за той обсяг ресурсів, який використовується [21]. Використання вже існуючих та попередньо налаштованих серверів Amazon для зберігання великих файлів, зокрема фотографій, сприяє зменшенню часу для виходу на ринок і полегшить масштабування додатку в майбутньому.

2.2.2 Клієнт частина

Для реалізації клієнтської частини було обрано TypeScript - мову програмування, яка була створена для подолання обмежень, що властиві мові з динамічною типізацією - JavaScript. Незважаючи на те, що JavaScript може бути більш гнучка, відсутність статичної типізації може призвести до численних

помилок. TypeScript допомагає уникнути таких проблем, додавши статичні типи. Статична перевірка типів на етапі розробки дозволяє виявляти потенційні помилки на ранніх стадіях, що сприяє створенню стабільних та надійних додатків[22]. Система типів TypeScript також підвищує читабельність коду та спрощує адаптацію нових розробників в майбутньому.

Для реалізації фронтенд частини додатку було використано бібліотеку з відкритим вихідним кодом React та фреймворк Next.js. React використовується для створення інтерактивних інтерфейсів користувача, в той час як Next.js надає можливість зручного використання серверного рендерингу.

React - це бібліотека JavaScript, яка використовується для розробки користувацьких інтерфейсів веб-додатків. React є провідною технологією для розробки стійких та динамічних інтерфейсів, завдяки багаторазовим компонентам і віртуальній DOM (Document Object Model), які полегшують ефективну обробку даних у реальному часі. Вона дозволяє розробникам побудувати ефективні та динамічні інтерфейси, що реагують на зміни в даних без перезавантаження сторінки.

Однією з головних технічних викликів веб-додатків є обробка великого обсягу користувальницьких даних та одночасне оновлення програми. React вирішує це завдання завдяки вбудованій підтримці віртуального DOM, що полегшує і прискорює процес рендерингу та покращує взаємодію з користувачем. React також використовує віртуальний DOM для оптимізації швидкості оновлення інтерфейсу, пришвидшуючи взаємодію між функціями виклику та відображенням, та має широкий набір інструментів для роботи з компонентами, станами та подіями. Більше того, React надає змогу створювати передові програми без урахування переходів між різними станами, що значно прискорює процес розробки. Ця бібліотека є популярною серед розробників завдяки своїй простоті використання та високій продуктивності. За даними JetBrains за 2023 рік, React є найпопулярнішою бібліотекою JavaScript [23]. Він також посідає друге місце за популярністю в рейтингу веб-фреймворків у всьому світі за 2023 рік [24].

Next.js - це фреймворк React, що використовується для створення багатофункціональних та масштабованих веб-додатків. Він поєднує в собі можливості серверного рендерінгу (SSR) та статичного генерування сайтів (SSG), що робить його ідеальним інструментом для створення динамічних та високопродуктивних веб-сайтів.

Однією з ключових переваг Next.js є його здатність оптимізувати продуктивність веб-сайтів. Завдяки SSR та SSG, він значно зменшує час необхідний для завантаження сторінок та може ефективно обробляти великий обсяг трафіку без втрати продуктивності. Крім того, механізм автоматичного розділення коду дозволяє завантажувати лише необхідний JavaScript код для кожної сторінки, що покращує швидкість завантаження та зменшує обсяги переданих даних. Next.js також надає вбудовану підтримку для компонентів React та маршрутизації, що спрощує процес створення та керування інтерфейсом веб-сайту. Завдяки широкому спектру функцій, таких як автоматична оптимізація зображень, підтримка CSS, попереднє завантаження сторінок та інші, Next.js стає потужним інструментом для розробки сучасних веб-додатків.

Для стилізації користувацького інтерфейсу програми було обрано Tailwind CSS. Tailwind CSS - це сучасний CSS-фреймворк, який дозволяє розробникам швидко створювати стильові компоненти та макети веб-сайтів, використовуючи набір заздалегідь визначених класів. Замість того, щоб писати CSS правила вручну, розробники можуть використовувати класи, які вже визначені у Tailwind CSS, для застосування стилів до елементів.

Однією з основних переваг Tailwind CSS є його атомарний підхід до стилів. Замість того, щоб визначати кожне CSS правило окремо, у Tailwind CSS існує набір класів, кожен з яких представляє конкретну стилістичну властивість (наприклад, розмір, колір, відступ і т. д.). Це робить роботу з CSS більш прозорою та швидкою, оскільки розробники можуть використовувати заздалегідь визначені класи замість того, щоб писати власні стилі. Крім того, Tailwind CSS надає можливість створювати розширені стилі шляхом комбінування класів та використання різноманітних варіантів (наприклад, величини, кольори, відступи),

що дозволяє розробникам швидко та ефективно налаштовувати вигляд своїх веб-сайтів. Tailwind CSS спрощує роботу з CSS, забезпечуючи швидку і ефективну розробку стильового оформлення веб-сайтів.

2.2.3 Середовище розробки

Було обрано Visual Studio Code (VS Code) для середовища розробки. VS Code - це безкоштовний, легкий у використанні текстовий редактор коду, розроблений компанією Microsoft. Однією з головних переваг VS Code є його легкість використання та налаштування. Він має інтуїтивний інтерфейс користувача, який дозволяє швидко розпочати роботу без додаткових складнощів. Крім того, він надає широкий спектр можливостей для редагування коду та роботи з проектами, що включає в себе автодоповнення коду, підтримку різних мов програмування, інтеграцію з системами керування версіями, вбудовану підтримку дебагу та широкий вибір розширень для налаштування функціоналу під потреби кожного розробника [25].

Додатково VS Code має зручне розширення для роботи з Docker, яке дозволяє використовувати контейнер Docker як повноцінне ізольоване середовище розробки.

Docker - це платформа контейнеризації, що дозволяє розробникам створювати і упаковувати програми разом із їх залежностями в самодостатні, ізольовані блоки, які називаються контейнерами. Ці контейнери ізольовані один від одного та від базової хост-системи, що робить їх легкими та портативними для розгортання та забезпечує стабільність та надійність середовища. Контейнери використовують менше ресурсів порівняно з віртуальними машинами, що дозволяє ефективно використовувати обчислювальні потужності сервера. Docker легко масштабується, що дозволяє запускати багато екземплярів контейнерів для обробки великих обсягів трафіку або завдань.

Docker забезпечує простий та послідовний процес створення, розповсюдження та запуску програм, що допомагає оптимізувати процес розробки та зменшує час та зусилля, необхідні для налаштування та керування

складною архітектурою програм. Використання Docker в даній роботі забезпечує ізольованість та незмінність середовища розробки, що гарантує автономність клієнта, сервера та бази даних, ізоляцію налаштувань програми та надійність роботи програмного забезпечення. Швидкість розгортання, відтворюваність середовища та зручність також є важливими факторами, оскільки вони безпосередньо впливають на стабільність релізів серверних інстансів.

2.3 Проектування структури бази даних та хмарного сховища

В проекті буде використовуватись об'єктно-реляційна система керування базами даних PostgreSQL та хмарне сховище Amazon Simple Storage Service (Amazon S3).

Нижче (рисунок 2.2) представлено основні таблиці бази даних, які будуть використовуватися у додатку.

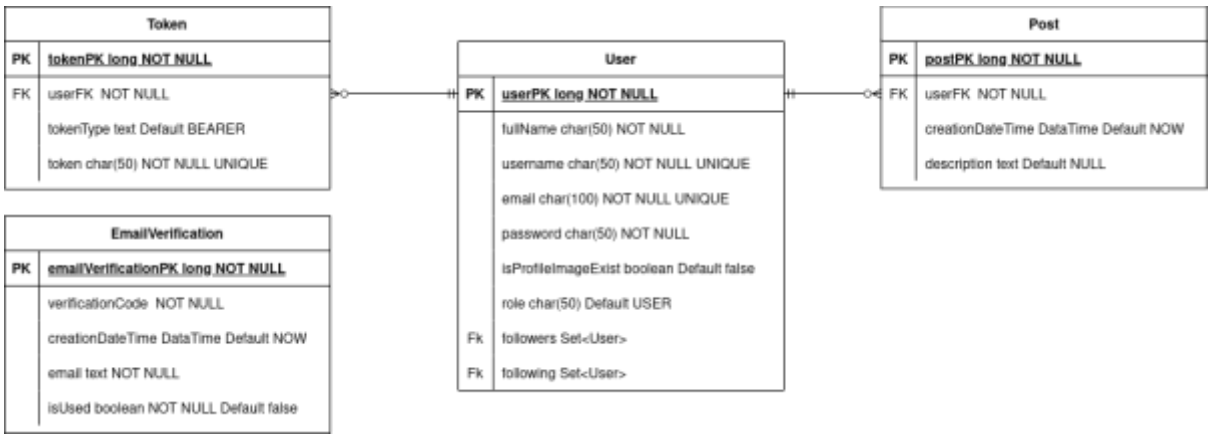


Рис. 2.2 Структура бази данних

Таблиця user зберігатиме основні дані про користувача, посилання на його підписників і на підписки, факт наявності картинки профілю, тощо. Таблиця token - зберігатиме дані про токени користувачів, що генеруються при авторизуванні користувача, для подальших підтверджень авторизації. Таблиця Post також зберігатиме дані про користувача-автора посту, про дату створення посту та не обов'язкове поле для опису. Таблиця EmailVerification використовуватиметься для

зберігання кодів підтвердження електронної пошти. Значення isUsed буде використано для обмеження спроби використання коду лише раз.

Для збереження фото, буде створено 2 відра(buckets) Amazon S3. Перше відро - spsp-post-images зберігатиме картинки постів, які зберігатимуться в файлах з назвою, яка відповідатиме унікальному ідентифікатору поста. Наступне відро - spsp-profile-images - робитиме те ж саме, але вже з картинкою профіля користувача.

2.4 Проектування архітектури

За основу додатку було взято клієнт-серверну архітектуру(рисунок 2.3). Клієнт-серверна архітектура — це модель взаємодії в обчислювальних системах, де функціональні можливості та обробка даних розподіляються між клієнтами та серверами, та можуть працювати між різними комп’ютерами, об’єднаними в мережу.

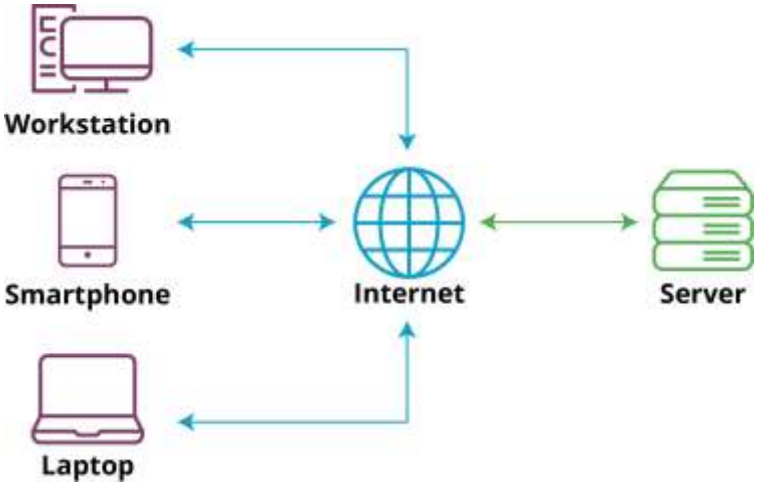


Рис. 2.3 Приклад клієнт-серверної архітектури

Клієнт - це пристрій або програма, яка отримує ресурси або послуги у сервера. Прикладами клієнтів є веб-браузери, мобільні додатки та інші користувацькі інтерфейси. Клієнт відповідає за взаємодію з користувачем, відображення інформації та ініціацію запитів до сервера.

Сервер - Це потужний комп'ютер або група комп'ютерів, які обробляють запити від клієнтів, зберігають дані та надають ресурси або послуги. Сервер

відповідає за обробку логіки бізнес-процесів, управління даними та забезпечення безпеки.

Додаток використовуватиме трирівневу архітектуру (рисунок 2.4). Клієнт-серверна трирівнева архітектура є різновидом клієнт-серверної моделі, яка, окрім основних двух описаних вище рівнів, виділяє ще один додатковий рівень - рівень даних.

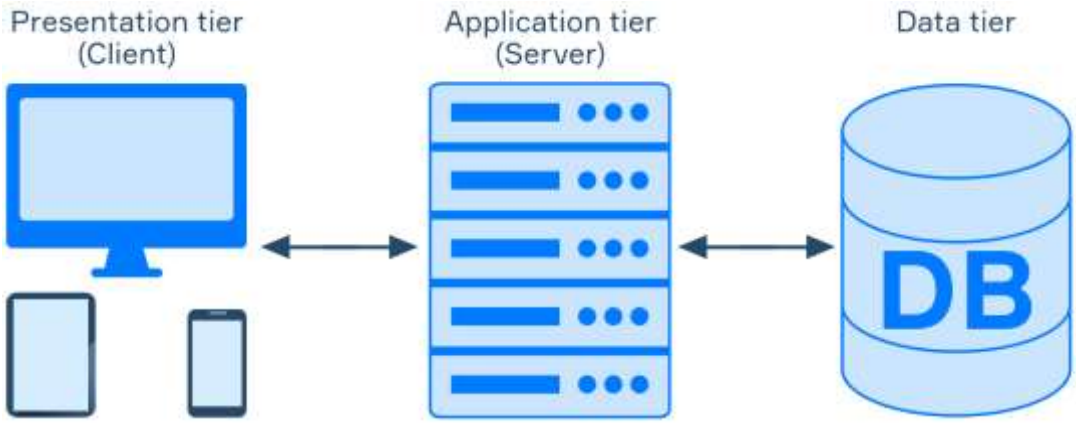


Рис. 2.4 Трирівнева архітектура

Загалом, функціональність системи буде поділена на три основні рівні: клієнтський, логічний та рівень даних. Клієнтський рівень представлений графічним інтерфейсом користувача. Логічний рівень представлено бекендом застосунку. Цей рівень обробляє запити, виконує необхідну бізнес-логіку і взаємодіє з рівнем даних для отримання або збереження інформації. Рівень даних представлено реляційною базою даних PostgreSQL, та хмарним сховищем Amazon S3.

Серверна частина додатку матиме триярусну архітектуру (рисунок 2.5), розділяючи функціонал на наступні рівні:

- Data Access Layer (DAL)- рівень доступу до даних;
- Business Service Layer (BSL) - рівень основної бізнес-логіки ;
- Application Programming Interface (API) - рівень взаємодії з клієнтом.

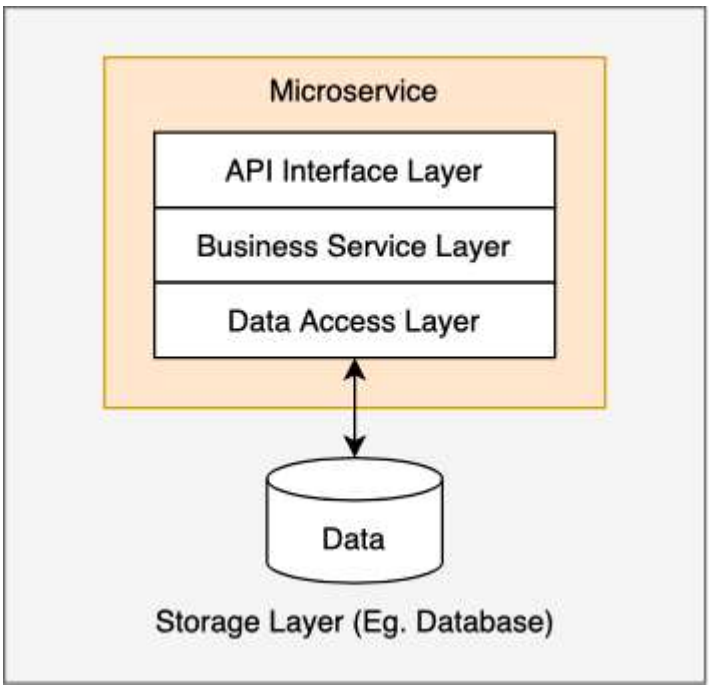


Рис. 2.5 Загальна структура серверу

На рівні DAL будуть реалізовані інструменти для роботи з базою даних та сервіси для роботи з хмарним середовищем. Сюди ж будуть відноситись сутності (entity) та репозиторії для роботи з базами даних.

Сутності (Entities) в Spring Data JPA (Java Persistence API) є основним компонентом для роботи з базами даних, забезпечуючи зручний і ефективний спосіб відображення об'єктів Java на таблиці бази даних.

Рівень бізнес логіки буде реалізовано за рахунок сервісів. Сервіси складатимуть основну частину серверного додатку, оскільки саме тут реалізовуватиметься вся основна логіка та робота з базами даних.

Рівень API визначатиме всі кінцеві точки серверу та відповідатиме за прийняття HTTP запитів клієнта, відправлятиме їх на обробку на рівень бізнес логіки, та формуватиме відповідь клієнту на відповідні запити. Даний рівень буде побудований за рахунок контролерів.

Клієнт частина буде розроблятись як повноцінна функціональна одиниця і взаємодіятиме з серверною частиною додатку через мережу Інтернет. Клієнт частина буде написана за допомогою React, та Next.js.

Буде створено API-маршрутизацію Next.js для додавання токенів авторизації до запитів на сервер, тим самим забезпечуючи можливість реалізувати авторизацію через JWT. Також буде використано middleware для первинної перевірки користувача на наявність JWT, для перенаправлення неавторизованих користувачів на сторінку авторизації за необхідності.

Загалом, клієнт частина додатку складатиметься з web-застосунку та матиме проміжний рівень (middleware), розділяючи функціонал на клієнтський рівень, що взаємодіятиме безпосередньо з користувачем та middleware, що отримуватиме запити з клієнтського рівня, оброблятиме їх, та надсилатиме на сервер самої программи.

Загальну архітектуру додатку було спроектованого у вигляді діаграми (рисунок 2.6).

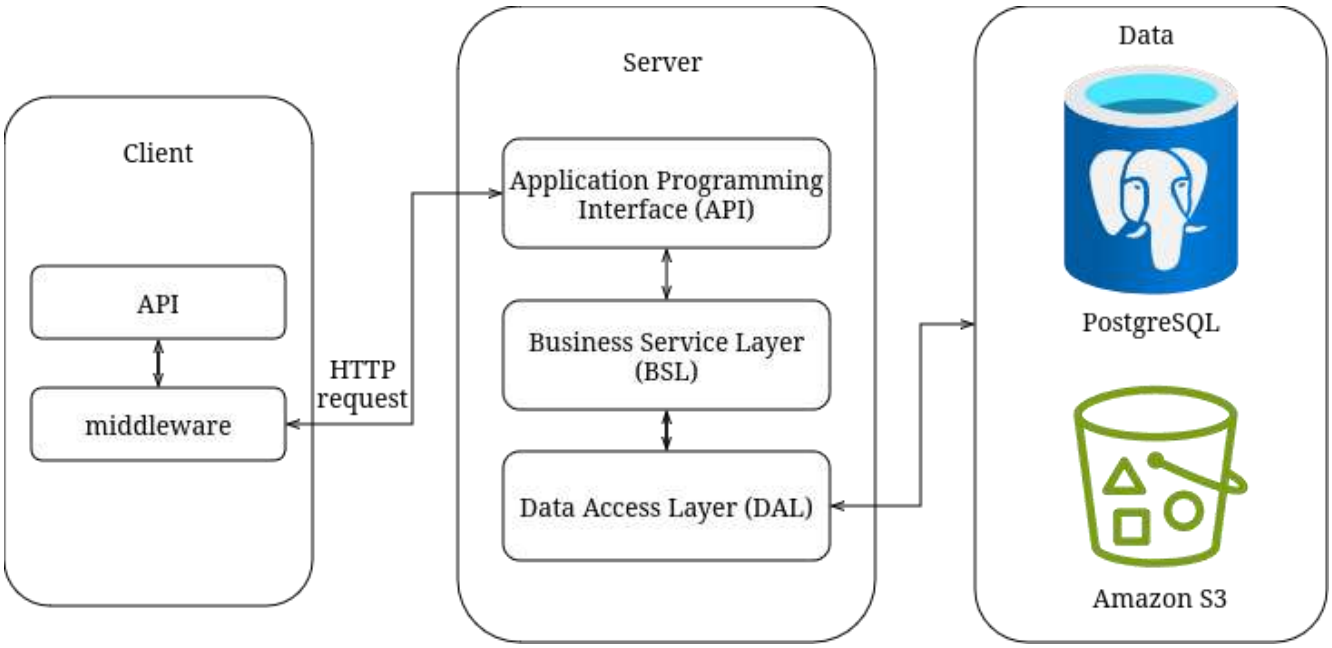


Рис. 2.6 Архітектура додатку

2.5 Проектування інтерфейсу користувача

Інтерфейс фронтенд користувача буде складатись з наступних сторінок:

- сторінка форми реєстрації;
- сторінка форми авторизації;
- сторінка форми редагування профілю з можливістю його видалення;

- сторінка форми створення та редагування посту;
- сторінка новин;
- сторінка профілю;
- сторінка з повідомленням що ресурсу не знайдено;
- публічна основна сторінка;
- сторінка з загальною інформацією про программу;
- сторінка контакті;
- сторінка політики приватності.

Маршрутизація додатку відрізнятиметься в залежності від того чи авторизований користувач чи ні. Неавторизовані користувачі можуть переглядати лише сторінки загального доступу, такі як основна сторінка, сторінка контактів, тому подібне. Вони також матимуть доступ до форм реєстрації та авторизації.

Зважаючи на те, що даний додаток є прототипом, наповнення основної сторінки, сторінки контактів, сторінки даних про програму та сторінки політики приватності буде за рахунок тексту за замовчуванням. Ці ж сторінки, разом із формами реєстрації та авторизації будуть у вільному доступі, усі інші - лише через авторизацію. При спробі перейти на будь яку авторизовану сторінку без відповідних прав, користувача перекине на сторінку авторизації.

Найпростіші прототипи декількох сторінок, спроектованих за допомогою Figma, представлено нижче.

Основна неавторизована сторінка матиме наступний вигляд (рисунок 2.7).

Рис.2.7 Прототип основної сторінки

В стрічці новин (рисунок 2.8) буде наявний спосіб вибрати фільтрацію за бажанням. При прокрутці новин, спрацьовуватиме підгрузка наступних.

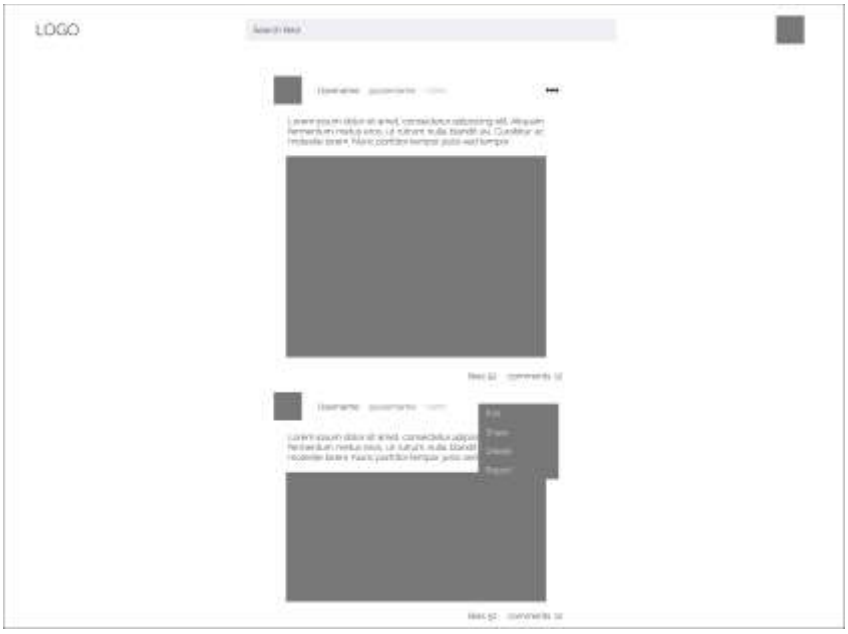


Рис. 2.8 Прототип сторінки новин

В заголовку знаходитимуться логотип програми, що перекидатиме на новинну стрічку, поле для пошуку користувачів, кнопка створення посту, кнопка переходу на свій профіль та кнопка виходу з акаунту. Даний заголовок буде представлений на всіх авторизованих сторінках. На сторінці профілю (рисунок 2.9) відображатиметься кількість постів, підписників, підписок, та, власне самі пости.

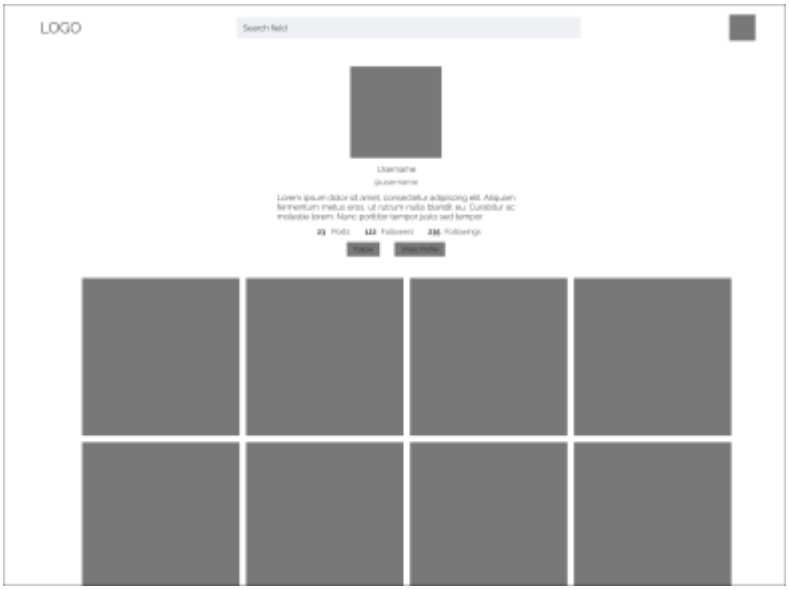


Рис.2.9 Прототип сторінки профілю користувача

Усі інші сторінки матимуть подібні патерни побудови і не потребують детального прототипування.

3 ОПИС РЕАЛІЗАЦІЇ СИСТЕМИ

Клієнт частина Nextjs, Серверна частина Spring та база даних PostgreSQL розроблялись в ізольованих контейнерах Docker як повноцінні функціональні одиниці і взаємодіяли один з одним через порти. Деталі реалізації представлені в коді.

3.1 Серверна частина

Реалізацію серверної частини представлено на на діаграмі класів основної логіки (рисунок 3.1) та діаграмі класів авторизації (рисунок 3.2). Дані діаграми мають спільний клас UserService, і представляють собою беккенд, проте класи довелось розділити на дві окремі діаграми для простоти відображення. Усі зв'язки між класами є зв'язків агрегації.

Класами основної логіки діаграми класів (рисунок 3.1) є сервіси UserService, PostService, S3Service, контроллери UserController, PostController, репозиторії PostRepository, UserRepository та клас S3Buckets.

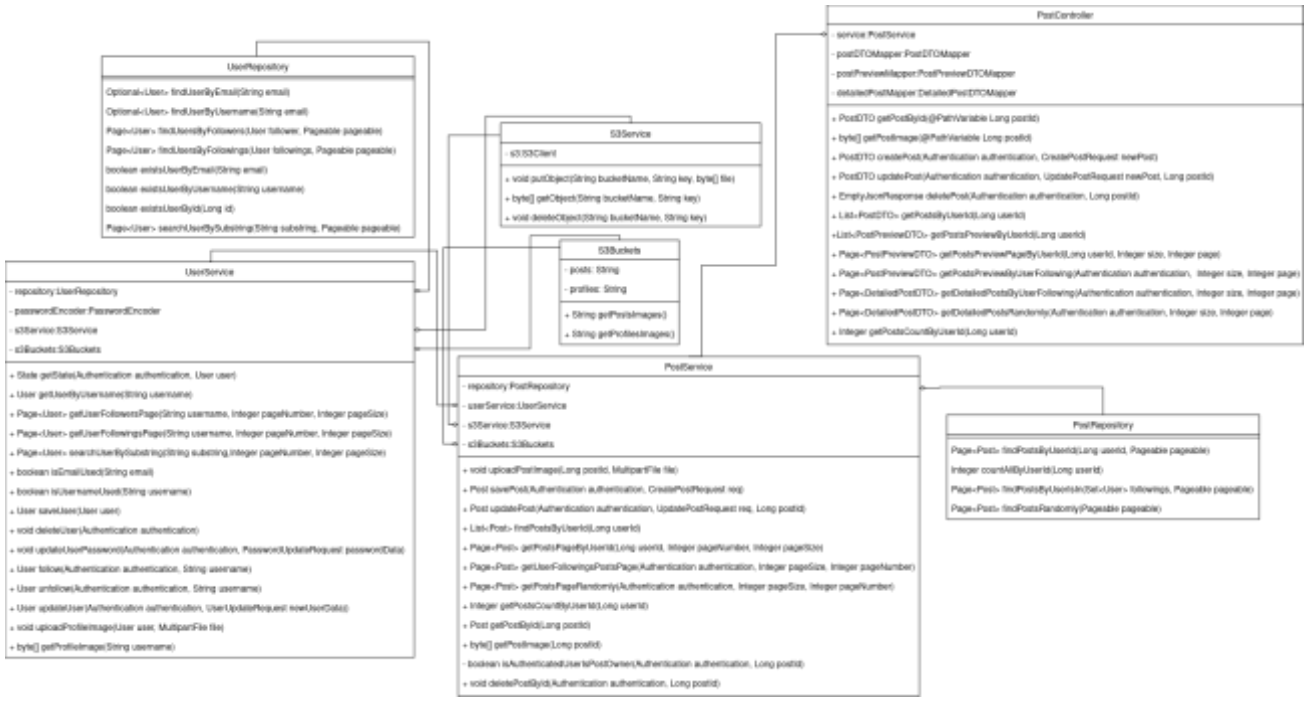


Рис. 3.1 Діаграма класів основної логіки

Клас UserService відповідає за всю основну логіку пов'язану з користувачем, а клас PostService - за основну логіку посту. Клас S3Service використовується сервісами PostService та UserService для отримання, завантаження та видалення фотографій. Клас S3Buckets зберігає посилання на відра Amazon S3. Класи PostRepository та UserRepository є компонентами для роботи з базами даних посту та користувача відповідно. PostController та UserController визначають кінцеві точки серверу. Вони приймають запити від користувача, та обробляють за допомогою сервісів.

Класи що відповідають за авторизацію користувача представлені на діаграмі класів авторизації (рисунок 3.2). Класами авторизації є сервіси AuthenticationService, MailSenderServise, EmailVerificationService та JwtService, контролери AuthenticationController та UserController, репозиторії EmailVerificationRepository, TokenRepository та UserRepository.

Основним класом даної діаграми є сервіс AuthenticationService, який реалізує основну логіку авторизації користувача. Він використовує сервіс EmailVerificationService для генерації та відправлення коду підтвердження електронної пошти. EmailVerificationService використовує MailSenderServise для відправлень повідомлень на електронну пошту і зберігає дані про відправлені коди підтвердження в EmailVerificationRepository. AuthenticationService також використовує JwtService для роботи з токеном. JwtService використовує TokenRepository для доступу до бази даних токену. AuthenticationController приймає запити пов'язані з реєстрацією та авторизацією користувача, та обробляє за допомогою сервісу AuthenticationService.

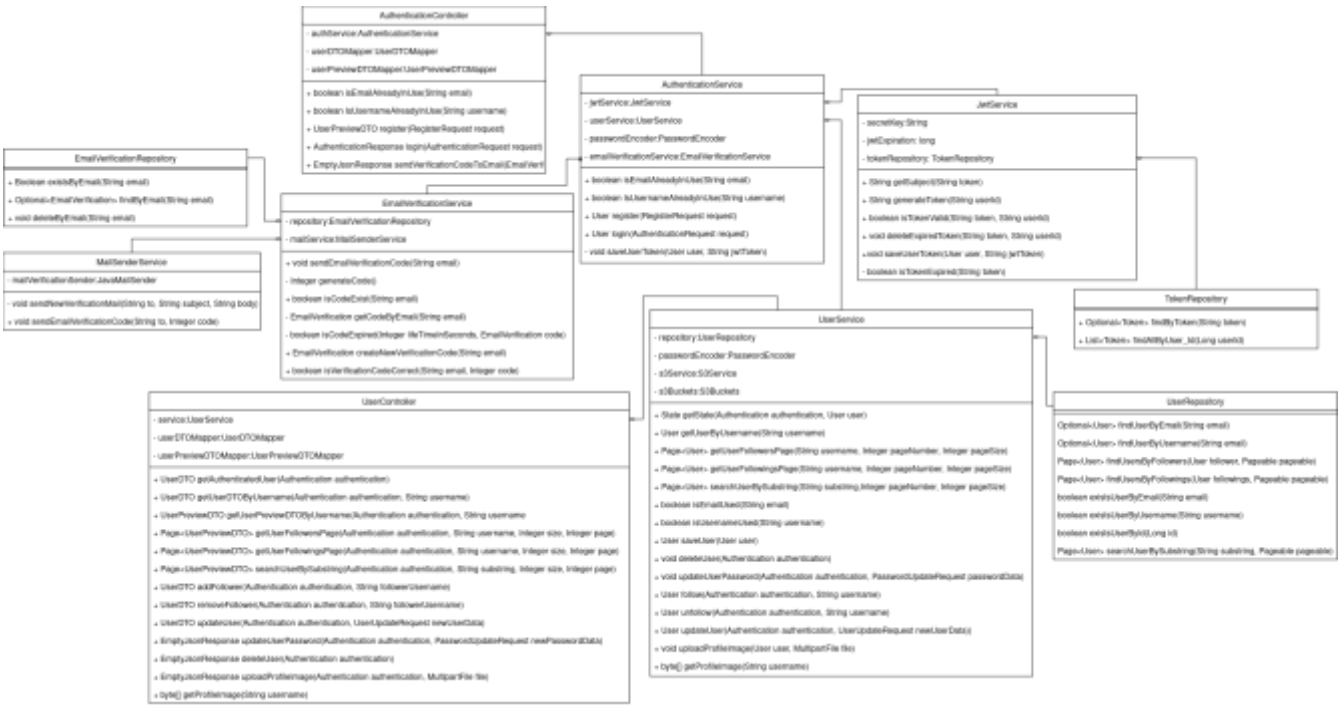


Рис. 3.2 Діаграма класів авторизації

Репозиторії застосунку є інструментами для роботи з базою даних та разом з сервісами для роботи з хмарним середовищем утворюють рівень рівень доступу до даних DAL додатку.

Сервіси застосунку представляють собою рівень бізнес логіки BSL серверу та на пряму взаємодіють між собою, використовуючи репозиторії для доступу до даних.

Рівень API визначає кінцеві точки серверу та реалізований за рахунок контролерів. Контролери в свою чергу залежать від сервісів. Відповідь на запит контроллера, відповідно до архітектурного стилю REST повертає дані в форматі JSON, а всі дані про пост та користувача конвертуються за шаблонами DTO, перед відправкою з контроллера. Так, наприклад, контроллер користувача повертає на фронтенд лише основні дані через UserDTO, уникаючи відправки таких вразливих полів як пароль користувача, його токени, пости, і тому подібне. Це не лише покращує безпеку, а ще й допомагає уникнути пересилки великої кількості непотрібної інформації.

3.2 Клієнт частина

Клієнт частина представляє собою повноцінний Web-додаток, та складається з клієнт частини та проміжного програмного забезпечення(middleware). Структуру клієнтського рівня, що взаємодіє безпосередньо з користувачем за допомогою web-сторінок представлено мапою сайту неавторизованого користувача (рисунок 3.3) та мапою сайту авторизованого користувача (рисунок 3.4).

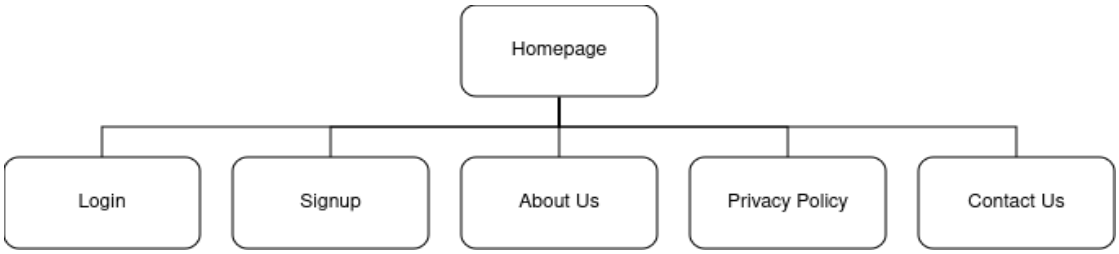


Рис. 3.3 Мапа сайту неавторизованого користувача

Сторінки представлені на мапі сайту неавторизованого користувача (рисунок 3.4) є публічними. Сторінки політики приватності, контактів та сторінки інформації про додаток доступні авторизованому користувачу також. Проте, при вході на домашню сторінку, чи сторінки реєстрації та авторизації, авторизованого користувача буде перенаправлено на сторінку новин як на основну сторінку авторизованого користувача. При спробі отримати приватний контент неавторизованим користувачем, користувача буде перенаправлено на сторінку авторизації.

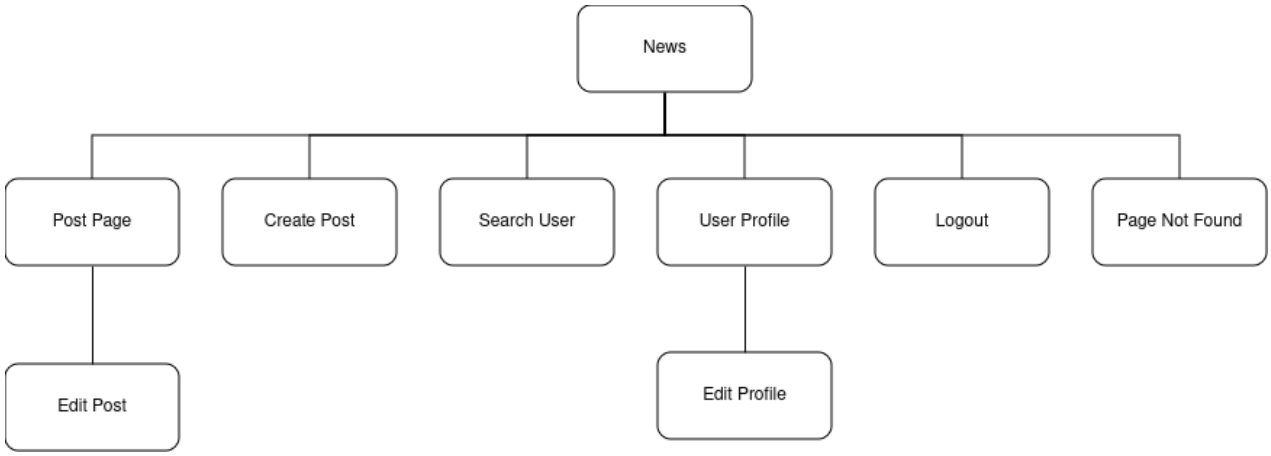


Рис. 3.4 Мапа сайту авторизованого користувача

Усі сторінки представлені на мапі сайту авторизованого користувача (рисунок 3.4) є приватними. Клієнт частина отримує усі запити, відправлені на домен, і, за необхідності перенаправляє користувача на сторінку з написом, що даний ресурс відсутній.

Проміжний рівень (middleware) web-застосунку отримує запити з клієнтського рівня, обробляє їх, та надсилає на сервер программи. Він слугує проміжком між web-застосунком та сервером. Структура проміжного рівня (рисунок 3.5) складається з файлу логування, та двох файлів які приймають усі запити, які починаються з /api/auth або /api/unauth. При авторизації користувача проміжний рівень перехоплює відповідь сервера щоб зберегти JWT(JSON Web Token) в httponly cookies, і передає далі лише дані про користувача. Після цього, щоразу, коли фронтенду потрібно буде зробити авторизований запит, він звертатиметься до проміжного рівня за посиланням /api/auth, той в свою чергу діставатиме токен з cookies і додаватиме його до запиту, пропускаючи запит далі. Для неавторизованих запитів використовуватиметься посилання /api/unauth.

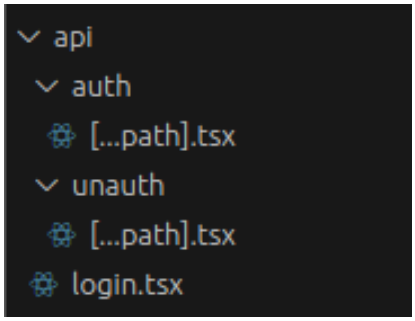


Рис. 3.5 Структура проміжного рівня

Особливу увагу було приділено авторизації користувача через те, що соціальні мережі доволі часто стають жертвами хакерів та махінацій в цілому. На діаграмі послідовності простої авторизації (рисунок 3.6) зображено один із найпростіших способів реалізації авторизації через збереження індивідуального номеру сесії. Проте, така модель абсолютно беззахисна перед міжсайтовою підробкою запитів CSRF (Cross-Site Request Forgery).

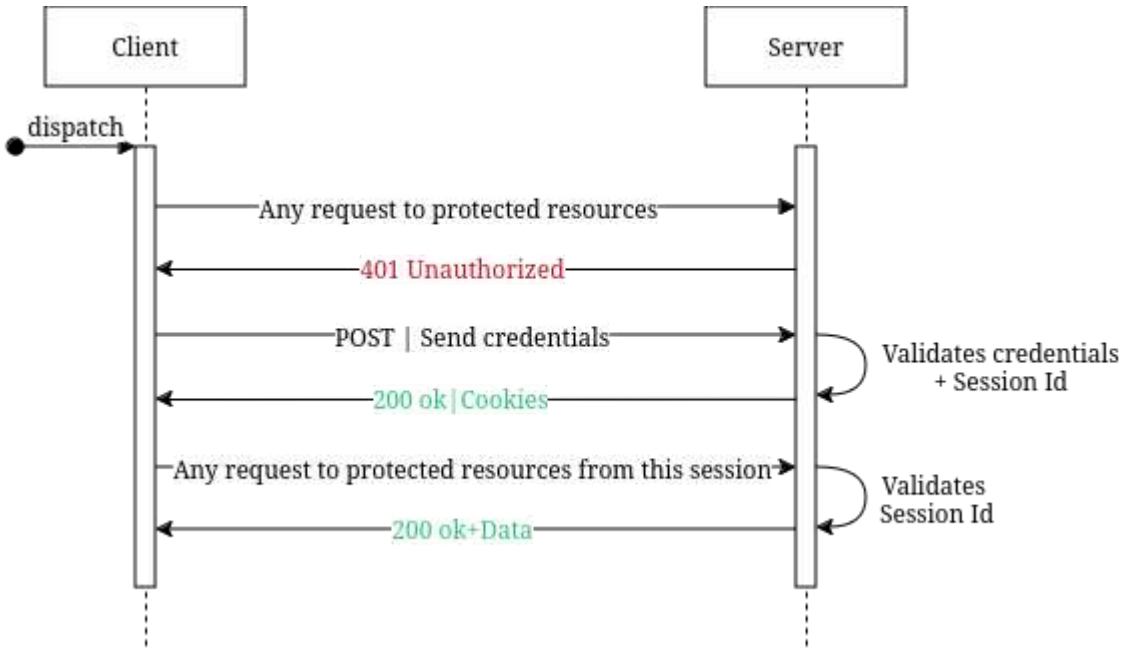


Рис. 3.6 Діаграма послідовності простої авторизації

Міжсайтова підробка запитів - це типова атака на веб-додатки, також відома як атака одним клацанням миші, яка використовується для виконання несанкціонованих дій під ім'я автентифікованого користувача. Щоб завадити подібного роду атакам, власне, й було реалізовано авторизацію через JWT. Деталі реалізації зображено на діаграмі послідовності авторизації з використанням JWT (рисунок 3.7).

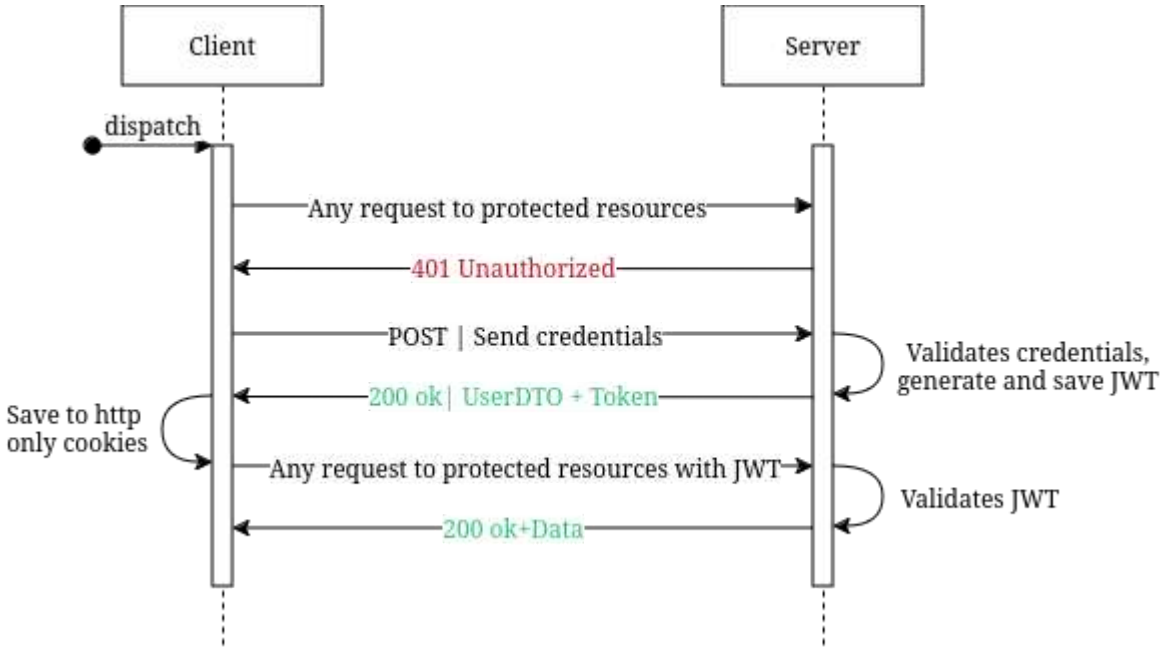


Рис. 3.7 Діаграма послідовності авторизації з використанням JWT

3.3 Інтерфейс користувача

Інтерфейс користувача серверної частини представлений REST API .

Загалом, його можна поділити на відкриті кінцеві точки та на кінцеві точки, які потребують автентифікації. Основні відкриті кінцеві точки представлені в таблиці 3.1.

Таблиця 3.1

Основні відкриті кінцеві точки

Метод	Посилання	Опис
POST	/auth/register	Зареєструвати користувача
POST	/auth/login	Авторизувати користувача
POST	/auth/logout	Вийти з аккаунту користувача.
POST	/auth/sendVerificationCode	Надіслати код для підтвердження електронної пошти.

Для закритих кінцевих точок у заголовок запиту потрібно додати дійсний токен. Токен можна отримати при автентифікації користувача. Закриті кінцеві точка використовують інформацію, пов’язану з користувачем, чий токен надається разом із запитом. Основні закриті кінцеві точки відображено в таблиці 3.2 .

Таблиця 3.2

Основні автентифіковані кінцеві точки

Метод	Посилання	Опис
GET	/post/{postId}	Отримати пост за ідентифікатором
POST	/post/create	Створити допис

Продовження таблиці 3.2

Основні автентифіковані кінцеві точки

Метод	Посилання	Опис
PUT	/post/update	Оновити допис
DELETE	/post/{postId}	Видалити допис
Метод	Посилання	Опис
GET	/user/{username}	Отримати DTO користувача за іменем користувача
GET	/user/{username}/followers/page	Отримати сторінку підписників
GET	/user/{username}/followings/page	Отримати сторінку підписок
GET	/user/search/page	Шукати користувача
PUT	/user/{followerUsername}/follow	Додати підписника
PUT	/user/{followerUsername}/unfollow	Видалити підписника
PUT	/user/update	Оновити користувача
PUT	/user/password/update	Оновити пароль користувача
DELETE	/user	Видалити користувача

Деталі реалізації запитів можна знайти в коді контролерів сервера (додаток Б). Взаємодія з інтерфейсом користувача серверної частини відбувалась за допомогою додатку для тестування та розробки API - Postman (рисунок 3.8). З запиту авторизації через Postman (рисунок 3.9) видно, що авторизація пройшла успішно, і сервер повернув дані про користувача та токен.

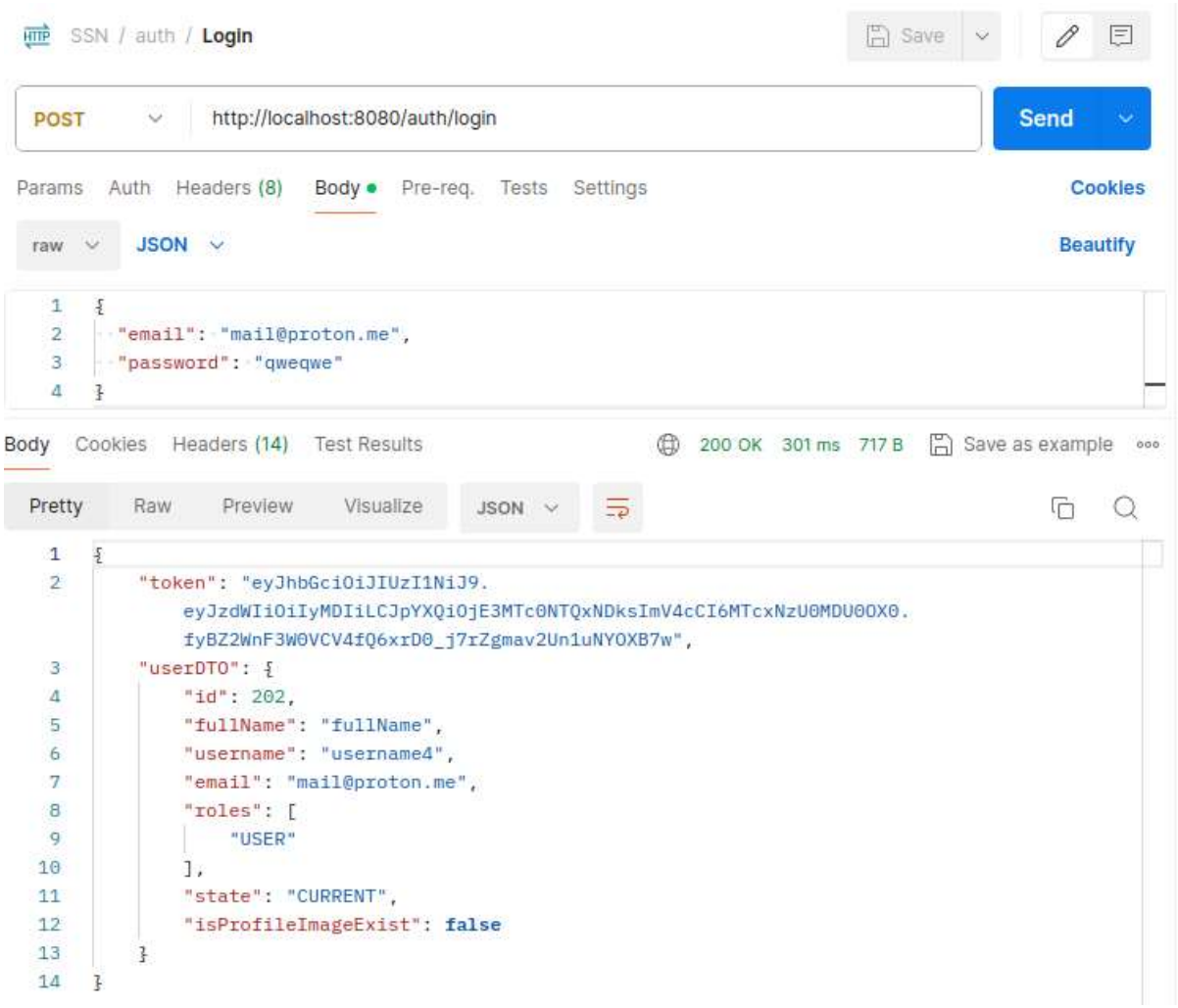


Рис. 3.8 Запит авторизації через Postman

Інтерфейс користувача фронтенду соціальної мережі Simple Photo Sharing Platform (SPSP) представлений Web-сайтом.

Зайшовши вперше на домен додатку як неавторизований користувач, відображатиметься публічна основна сторінка (рисунок 3.9) із загальною інформацією. Як і було вирішено при проектуванні інтерфейсу користувача, наповнення основної сторінки на даному етапі, як і сторінок контактів, даних про програму та політики приватності буде за рахунок тексту за замовчуванням.

В основному меню неавторизованого сайту (рис. 3.3.2) є кнопки-посилання на сторінки реєстрації та авторизації, та на основну сторінку, через натиснення на логотип. Дане меню є ідентичним для всіх сторінок публічного доступу неавторизованого користувача. Основна сторінка також має кнопку-посилання Get started, яка перенаправляє користувача на форму авторизації.

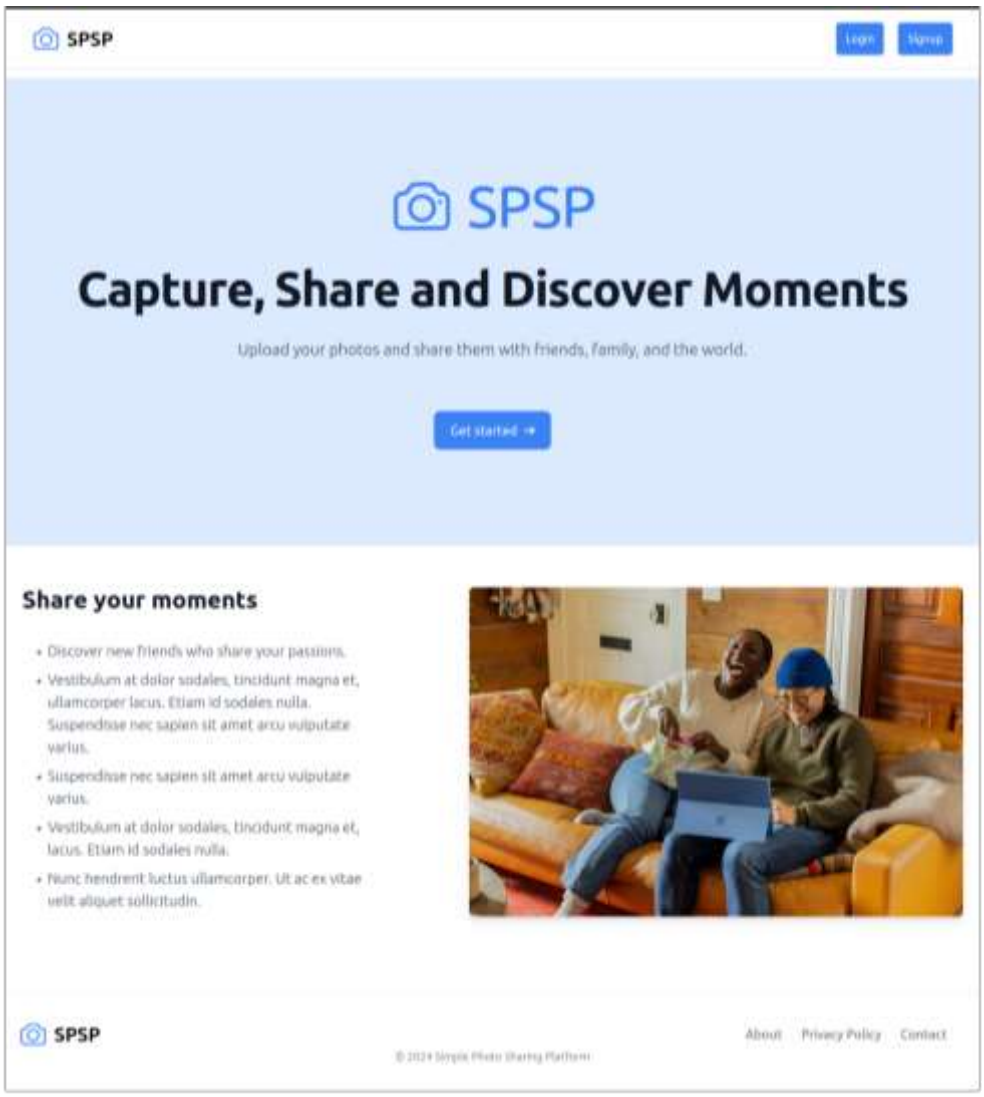


Рис. 3.9 Основна сторінка

Футер — блок в нижній частині сторінки, містить логотип, посилання на інші сторінки публічного доступу та знак авторського права з назвою додатку. Тут розміщені посилання на сторінки з загальною інформацією про программу, політики приватності та контактів. Даний футер є незмінним і використовується як для публічних, так і приватних сторінок.

На сторінці реєстрації (рисунок 3.10) необхідно заповнити обов’язкові поля імені користувача, електронної пошти, паролю, підтвердження паролю, та поля повного імені користувача. Вибір фотографії профілю не є обов'язковим. В разі реєстрації без фотографії профілю, буде згенеровано стандартне зображення профілю, що буде забарвлене випадковим кольором та міститиме перші дві літери імені користувача. Поля мають обмеження, і в разі їх порушень відобразять

помилку. Ім’я користувача та електронна пошта мають бути унікальними. Поки усі вимоги не будуть виконані, кнопка реєстрації не стане активною.

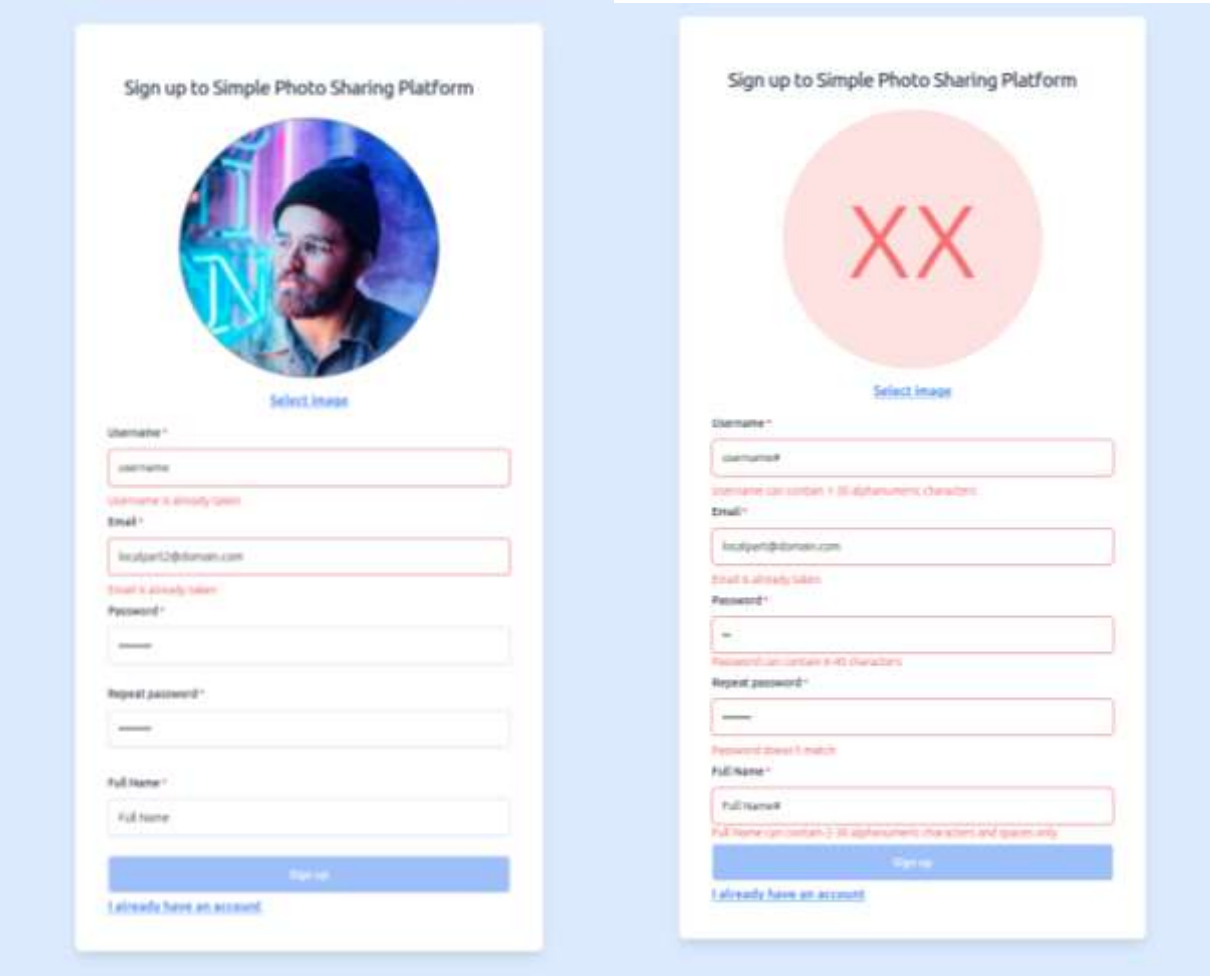


Рис. 3.10 Сторінка форми реєстрації

Після коректного заповнення форми реєстрації, активується кнопка реєстрації. При натисненні на неї активується діалогове вікно підтвердження електронної пошти (рисунок 3.11) і надішлеться код підтвердження на вказану пошту. За необхідності, через 30 секунд, буде можливість переслати код підтвердження.

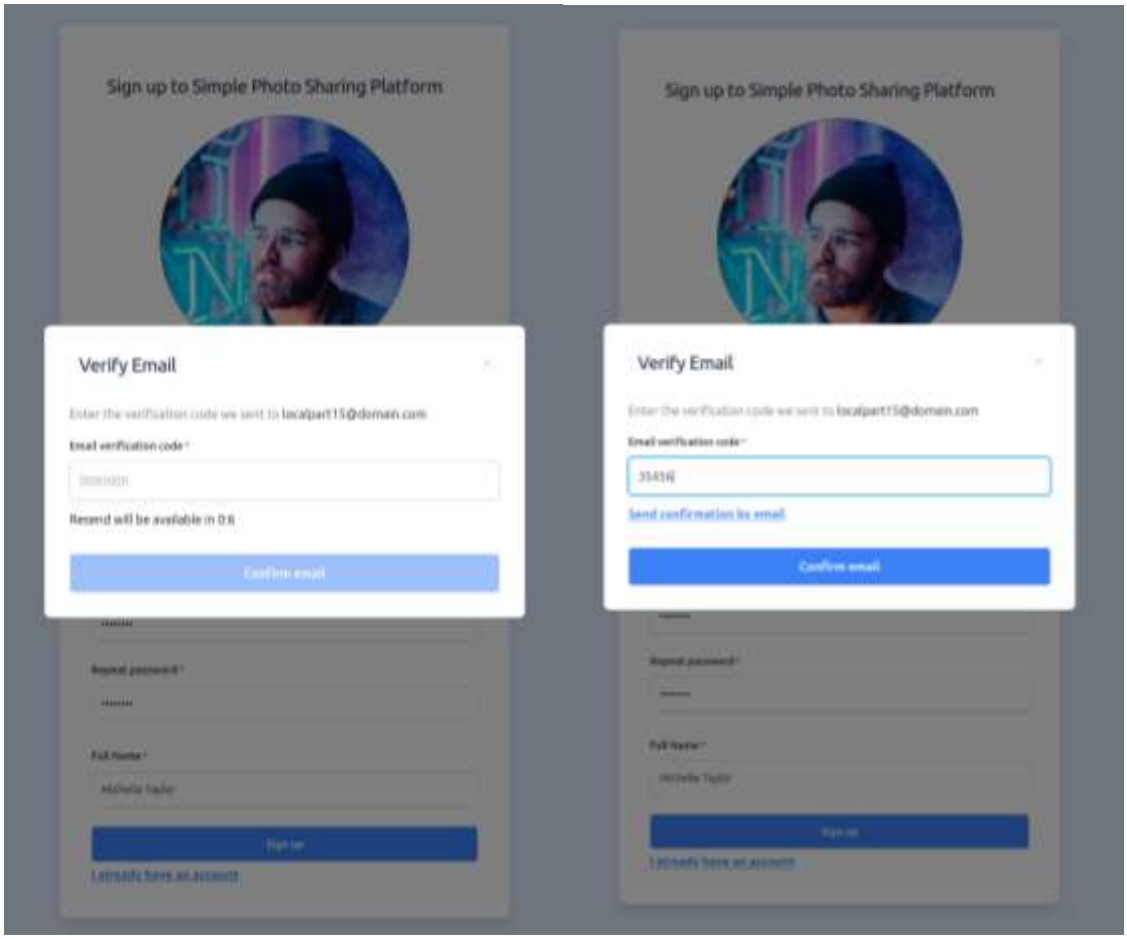


Рис. 3.11 Діалогове вікно підтвердження електронної пошти

Після підтвердження електронної пошти та успішної реєстрації, програма перенаправить користувача на сторінку авторизації (рисунок 3.12). При вводі коректних даних авторизації відбудеться перехід на сторінку новин (рисунок 3.13).

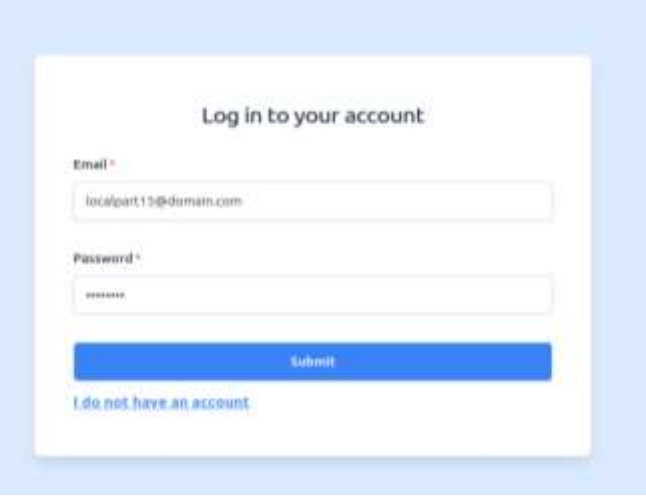


Рис. 3.13 Сторінка форми авторизації

Сторінка новин (рисунок 3.14) має можливість свідомого вибору фільтрації контенту за друзями та випадковим відбором. В майбутньому вибір фільтрів буде розширено та додано можливість їх налаштувань.

Сторінка новин є першою сторінкою приватного доступу. На цьому етапі можна побачити зміни основного меню. Наявне поле для пошуку користувачів, кнопка-посилання на сторінку створення посту та фото профілю користувача. При натисканні на фото профілю користувача відкривається випадаюче меню. Тут можна знайти посилання на свій профіль, посилання на його редагування та розлогування. Головне меню матиме такий вигляд на всіх сторінках авторизованого користувача.

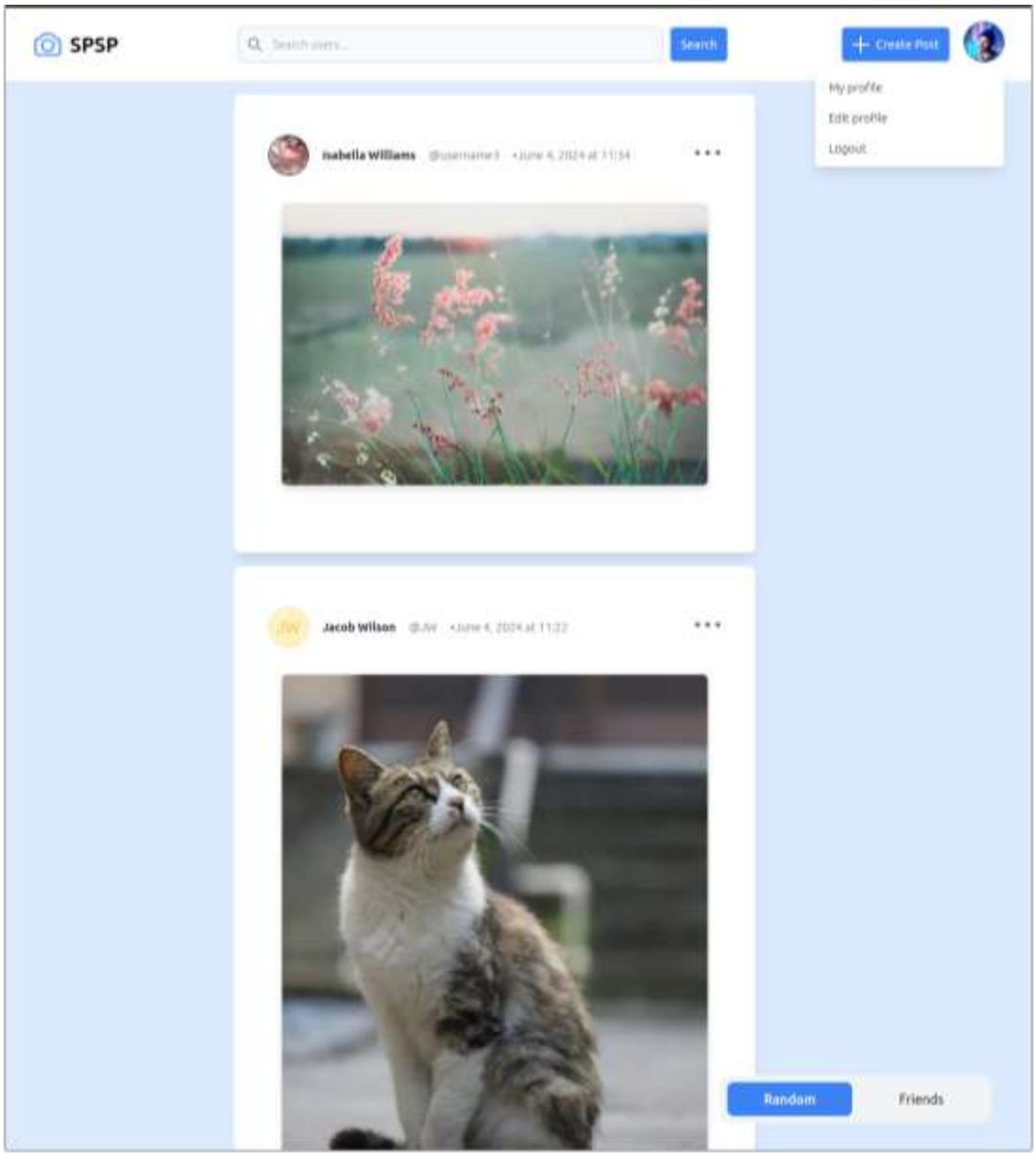
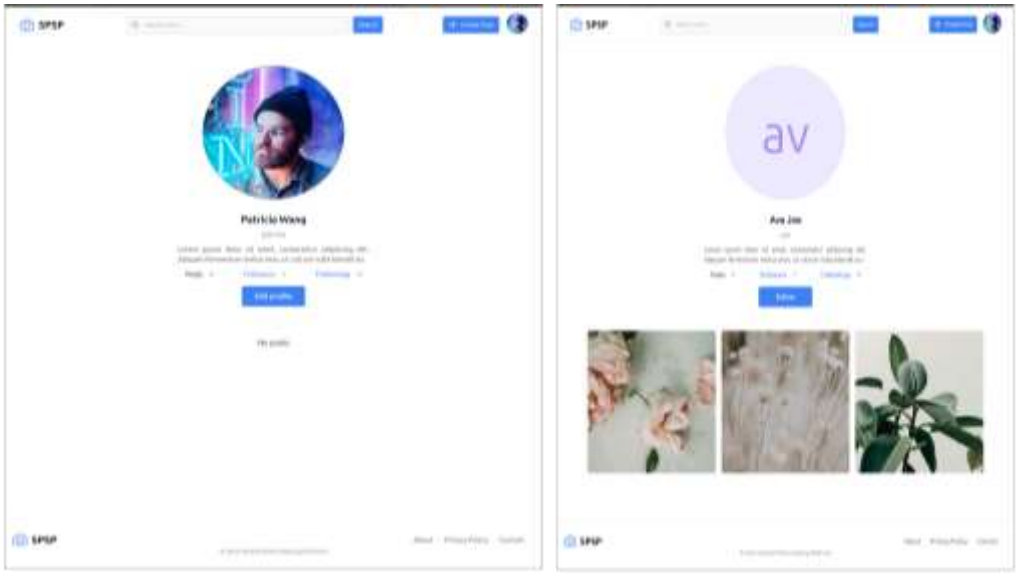


Рис. 3.14 Новинна стрічка з випадковими постами

Власний профіль користувача та профілі інших користувачів мають однакову структуру але відрізняються наявними можливостями (рисунок 3.15). В свому профілі є можливість редагування, а в профілях інших є можливість піписатися або відписатися.



Рисю 3.15 Профілі користувача

Форма редагування профілю (рисунок 3.16) матиме ті ж перевірки полів, що й форма реєстрації. При спробі редагування фото чи паролю, відкриється діалогове вікно редагування. Для зміни паролю, потрібно також вказати старий пароль. При неправильно введених даних, пароль не буде змінено. При спробі видалення профілю, відкриється діалогове вікно для підтвердження.

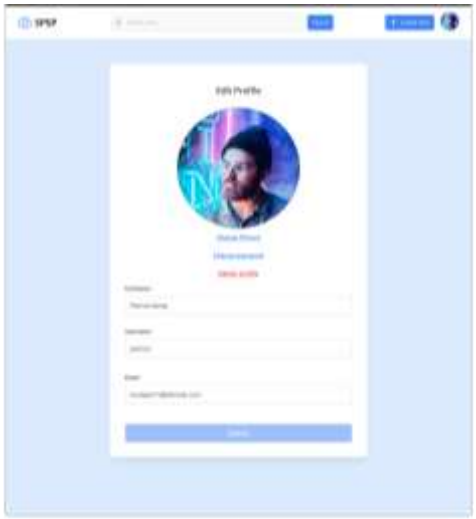


Рис. 3.16 Редагування профілю користувача

Використавши поле пошуку користувача в основному меню, відкриється сторінка пошуку(рисунок 3.17) із списком профілів. Натиснувши на профіль, можна на нього перейти.

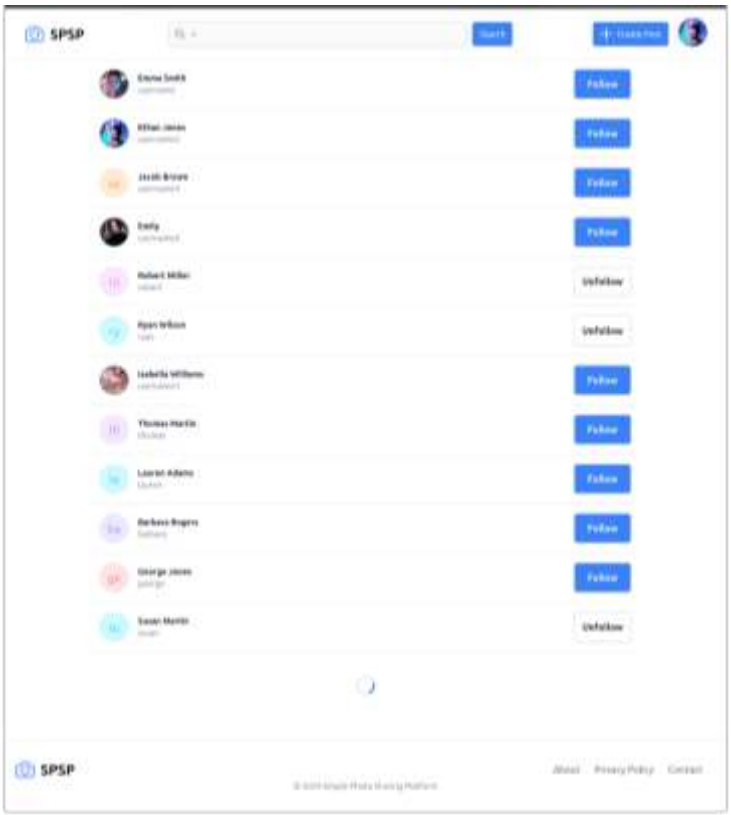


Рис. 3.17 Пошук користувача

В профілі користувача можна переглянути списки своїх підписників та підписок (рисунок 3.18). З даних списків можна перейти на сторінки цих користувачів натиснувши на відповідний елемент списку.

Натиснувши на пост в профілі, користувача буде перенаправлено на сторінку посту (рисунок 3.19). При натисненні на меню в правому верхньому кутку буде представлено випадаюче меню. Якщо користувач є автором посту, йому буде запропоновано редагування або видалення посту. Якщо ж ні, буде запропоновано відписатись чи підписатись на автора посту, або перейти на основну сторінку посту.

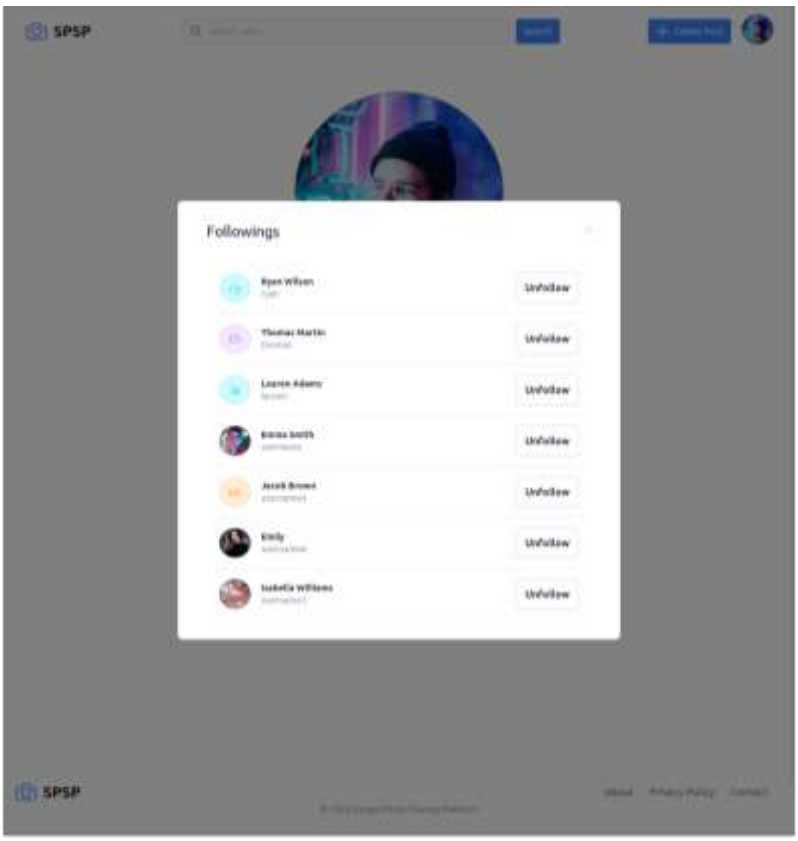


Рис. 3.18 Перегляд списку підписок

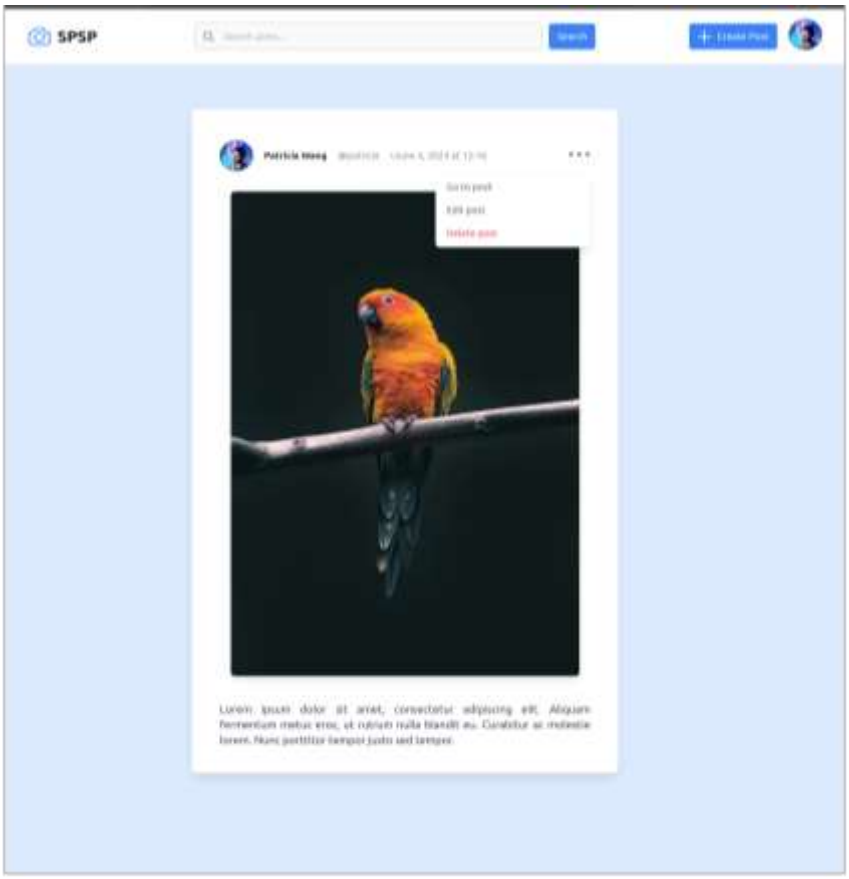


Рис. 3.19 Сторінка посту

При переході за неправильним посиланням, відобразиться сторінка незнайденого ресурсу(рисунок 3.20).

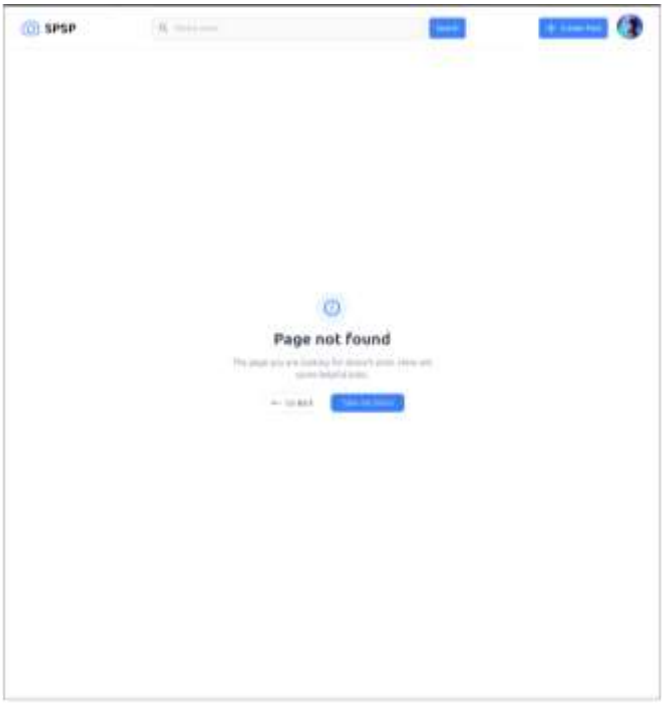


Рис. 3.20 Сторінка незнайденого ресурсу

4 ТЕСТУВАННЯ ПРОГРАМИ

Важливим етапом розробки, який забезпечує високу якість, надійність та безпеку програмного забезпечення є його тестування. Використання різних типів тестування допомагає виявити дефекти на різних стадіях розробки та забезпечити успішне впровадження продукту. Саме тому, вибір стратегій тестування повинен відбуватись відповідально.

Функціональне тестування — це процес перевірки функціональності програмного забезпечення, щоб переконатися, що воно працює відповідно до специфікацій і вимог.

Функціональні вимоги, що будуть протестовані:

- реєстрація користувача;
- авторизація користувача;
- редагування профілю;
- видалення профілю;
- створення посту;
- редагування посту;
- видалення посту;

Для тестування даних вимог було розроблено тест кейси. Тест кейс – це документ що представляє собою інструкції по перевірці певного функціоналу, який описує виконання тестової ситуації та очікуваний результат.

Таблиця 4.1

Тест кейс позитивної реєстрації

Тест кейс F-01. Позитивна реєстрації користувача		
Передумови: Зайти на основну сторінку додатку. Перейти за посиланням для реєстрації (/auth/signup)		
Кроки	Введені Дані	Очікуваний результат
Введення даних в поля.	username, Password, Password, Full Name, localpart@domain.com	Після заповнення всіх полів, кнопка реєстрації стане активною
Натиснути кнопку реєстрації		З'явиться діалогове вікно підтвердження електронної пошти. На електронну пошту прийде числовий код підтвердження.
Ввести код підтвердження з електронної пошти.	Код підтвердження з електронної пошти	Кнопка підтвердження стане активною.
Натиснути на кнопку підтвердження.		Реєстрація пройшла успішно. Відбулось перенаправлення на сторінку авторизації

Таблиця 4.2

Тест кейс негативної реєстрації

Тест кейс F-02. Негативна реєстрації користувача		
Передумови: Виконати тест-кейс 01. Перейти за посиланням для реєстрації (/auth/signup).		
Кроки	Введені Дані	Очікуваний результат
Введення даних в поля.	username, Password, Password, Full Name, localpart2@domain.com	Після заповнення, поле Username показує повідомлення що даний username вже існує, і просить обрати інший. Кнопка реєстрації не активна.

Таблиця 4.3

Тест кейс позитивної автентифікації

Тест кейс F-03. Позитивна автентифікація користувача		
Передумови: Зайти на основну сторінку додатку. Виконати тест-кейс 01. Перейти за посиланням для автентифікації (/auth/login)		
Кроки	Введені Дані	Очікуваний результат
Введення даних в поля.	localpart@domain.com, Password,	Після заповнення всіх полів, кнопка автентифікації стане активною.
Натиснути кнопку автентифікації		Автентифікація пройшла успішно. Відбулось перенаправлення на сторінку новин (/news)

Таблиця 4.4

Тест кейс негативної автентифікації

Тест кейс F-04. Негативна автентифікація користувача		
Передумови: Зайти на основну сторінку додатку. Виконати тест-кейс 01. Перейти за посиланням для автентифікації (/auth/login)		
Кроки	Введені Дані	Очікуваний результат
Введення даних в поля.	localpart2@domain.com, Password,	Після заповнення всіх полів, кнопка автентифікації стане активною.
Натиснути кнопку автентифікації.		Автентифікація неуспішна. З'явилось відповідне сповіщення.

Таблиця 4.5

Позитивний тест кейс створення посту

Тест кейс F-05. Позитивне створення посту		
Передумови: Авторизувати користувача. Зайти на основну сторінку додатку (/).		
Кроки	Введені Дані	Очікуваний результат
В правому верхньому куті головного меню натиснути на кнопку створення посту.		Відкриється сторінка з формою створення посту (post/create)
Натиснути кнопку завантаження фотографії	Вибрати картинку формату .jpg або .jpeg	Картинка відобразилась в полі для вибору. Кнопка створення посту стане активною.
Натиснути кнопку створення посту		Пост створено. Перенаправлення на сторінку посту.

Таблиця 4.6

Негативний тест кейс створення посту

Тест кейс F-06. Позитивне створення посту		
Передумови: Авторизувати користувача. Зайти на основну сторінку додатку.		
Кроки	Введені Дані	Очікуваний результат
В правому верхньому куті головного меню натиснути на кнопку створення посту.		Відкриється сторінка з формою створення посту (post/create)
Натиснути кнопку завантаження фотографії	Вибрати картинку не формату .jpg або .jpeg	Висвітиться помилка формату

Таблиця 4.7

Тест кейс видалення посту

Тест кейс F-07. Видалення посту		
Передумови: Авторизувати користувача. Створити пост. Зайти на сторінку посту.		
Кроки	Введені Дані	Очікуваний результат
В правому верхньому куті посту натиснути на кнопку меню. Вибрати пункт Видалити		Пост успішно видалено

Таблиця 4.8

Тест кейс редагування посту

Тест кейс F-08. Редагування посту		
Передумови: Авторизувати користувача. Створити пост. Зайти на сторінку посту.		
Кроки	Введені Дані	Очікуваний результат
В правому верхньому куті посту натиснути на кнопку меню. Вибрати пункт Редагувати		Перехід на сторінку редагування посту.
Додати опис посту.	Lorem ipsum dolor sit amet.	Кнопка редагування стане активною.
Натиснути кнопку редагування		Опис посту успішно змінено.

Таблиця 4.9

Тест кейс видалення профілю

Тест кейс F-08. Видалення профілю		
Передумови: Авторизувати користувача. Зайти на сторінку редагування (/account/edit).		
Кроки	Введені Дані	Очікуваний результат
Під фотографією профілю користувача натиснути кнопку видалення профілю.		З'явиться діалогове вікно-попередження, яке запросить підтвердження на видалення.
Підтвердити видалення		Профіль успішно видалено.

Таблиця 4.10

Тест кейс редагування профілю

Тест кейс F-08. Редагування профілю		
Передумови: Авторизувати користувача. Зайти на сторінку редагування(/account/edit).		
Кроки	Введені Дані	Очікуваний результат
Під фотографією профілю користувача натиснути кнопку завантаження фотографії		З'явиться діалогове вікно вибору фотографії.
Вибрати картинку	Вибрати картинку формату .jpg	Кнопка редагування стане активною
Натиснути кнопку редагування		Картинку профілю успішно змінено.

Сформовано тестові набори даних із використанням класів еквівалентності та кордонних умов для тестування основних полів форми реєстрації користувача, що представлені нижче (Таблиці 4.1 - 4.10).

Клас еквівалентності - це множина вхідних значень або станів системи, які обробляються однаково або мають однаковий очікуваний результат. Валідні класи еквівалентності (ОК-тести) - це класи, які містять допустимі або очікувані вхідні дані. Невалідні класи еквівалентності (NOK-тести) - це класи, які містять недопустимі або неочікувані вхідні дані.

Кордонні умови — це специфічні тестові випадки, які перевіряють поведінку програмного забезпечення на межах допустимих вхідних значень. Помилки часто виникають саме на межах діапазонів, тому перевірка цих умов є ефективною.

Таблиця 4.11

Класи коректних даних поля Username (ОК-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 1 до 30 символів	UsernameUsernameUsernameUserna	30 символів
		u	1 символ
		Username	8 символів
2	Латиниця великим та малим регістром [A;Z] [a; z]	USERNAME	Латиниця великим регістром
		username	Латиниця малим регістром
		UsErNaMe	Латиниця великим та малим регістром
3	Числа	12345	Лише числа
		Username12	Числа та латиниця

Таблиця 4.12

Класи некоректних даних поля Username (NOK-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 1 до 30 символів		0 символів
		UsernameUsernameUsernameUsernam	31 символ
2	Не дозволені спеціальні символи	#U*sern@m.e~	Спеціальні символи

Таблиця 4.13

Класи коректних даних поля Password (ОК-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 6 до 40 символів	PasswordPasswordPasswordPasswordPassword	30 символів
		Password	8 символів
		passwo	6 символів
2	Дозволена латиниця, числа та спеціальні символи.	P@s8Wo&d*	Спеціальні символи + латиниця + числа

Таблиця 4.14

Класи некоректних даних поля Password (NOK-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 6 до 40 символів	Passw	5 символів
		PasswordPasswordPasswordPasswordPasswordP	41 символ

Таблиця 4.15

Класи коректних даних поля Full Name (ОК-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 2 до 30 символів	Full Name Full Name Full Name	30 символів
		FN	2 символи
2	Дозволена латиниця великим та малим регістром [A;Z] [a; z] + пропуск	Full Name	Латиниця великим і малим регістром +пропуск

Таблиця 4.16

Класи некоректних даних поля Full Name (NOK-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 2 до 30 символів	F	1 символ
		Full Name Full Name Full NameF	31 символ
2	Не дозволені спеціальні символи	F*u!! N@me*	Спеціальні символи
3	Не дозволені числа	12345	Числа

Таблиця 4.17

Класи некоректних даних поля Full Name (NOK-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 5 до 50 символів	l@de	4 символів
		localpartlocalpartlocalpartloc@domaindomain.comcomm	51 символ
2	Дозволені спеціальні символи . -_	local*part@do#m!ain.com	Використання спеціальних символів, що не дозволені
3	Обов’язковий символ @ між локальною та доменною частинами	localpartdomain.com	Відсутність символу @
		@localpartdomain.com	Символ @ перед локальною частиною

Таблиця 4.18

Класи коректних даних поля Email (ОК-тести)

№	Опис класу	Приклади даних	Коментарі до даних
1	Від 5 до 50 символів	l@d.c	5 символів
		localpart@domain.com	20 символів
		localpartlocalpartlocalpartloc@domaindomain.comcom	50 символ
2	Дозволена латиниця великим та малим регістром [A;Z] [a; z]	LocaLparT@DomaIn.Com	Латиниця великим та малим регістром
3	Дозволені спеціальні символи . -_	local-part@dom_ain.com	Спеціальні символи
4	Обов’язковий символ @ між локальною та доменною частинами	localpart@domain.com	Символ @ між локальною та доменною частинами
5	Дозволені числа	l0calpart2@domai2n.com	Числа

Тестування серверної частини відбувалось з використанням Postman. Postman — це популярна платформа для розробки API, що дозволяє розробникам ефективно створювати, тестувати та змінювати API.

Під час написання API серверу, був створений список запитів для тестування за допомогою Postman (рисунок 4.1).

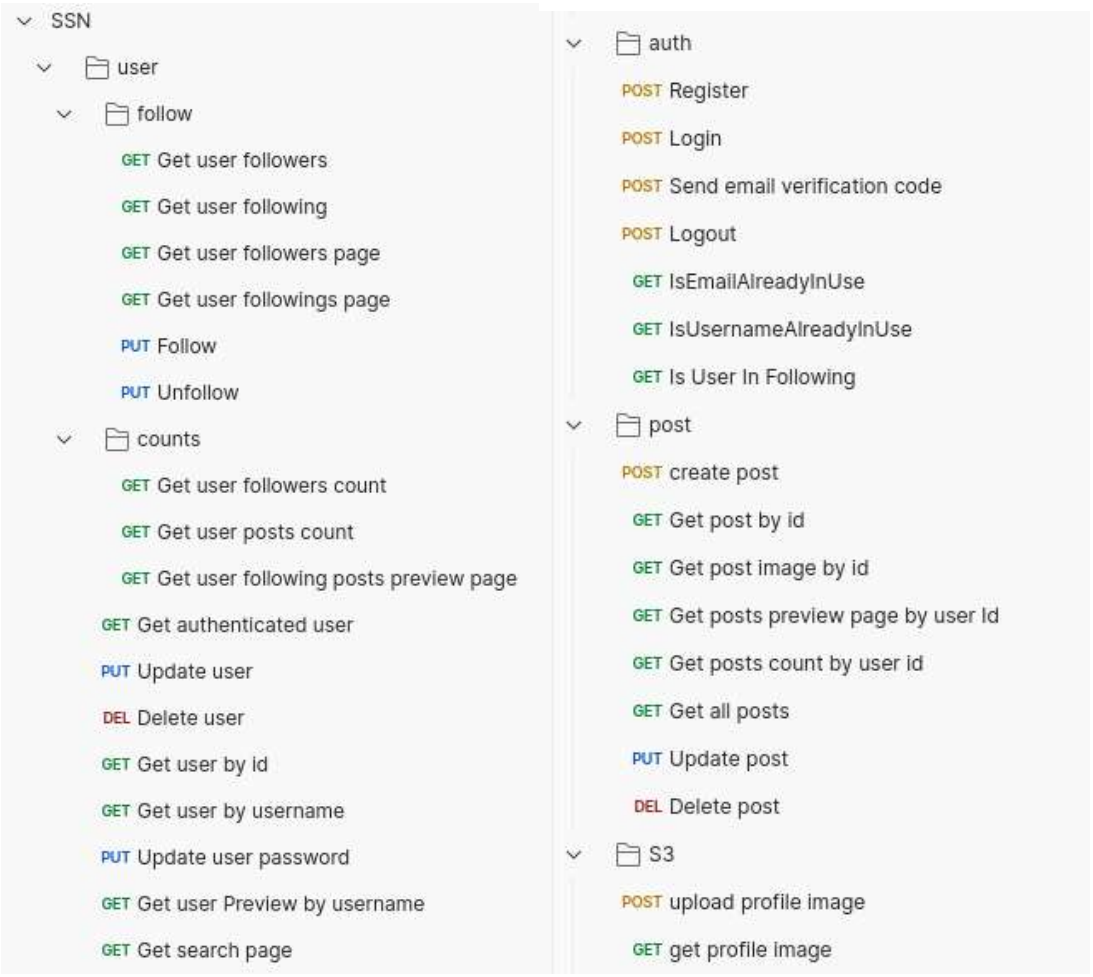


Рис. 4.1 Список запитів для тестування API серверу

Програмний застосунок протестовано на відповідність вимогам за допомогою основних тест кейсів (таблиці 4.1 - 4.10). Було проведено тестування форми реєстрації та форми редагування профілю користувача із використанням сформованих тестових наборів даних на базі класів еквівалентності та кордонних умов(таблиці 4.11 - 4.18). Додаток SPSP успішно пройшов тестування.

ВИСНОВКИ

Під час виконання даної роботи було проведено аналіз літературних джерел на тему роботи соціальних мереж, оглянуто стратегії алгоритмічної фільтрації підбору контенту, визначено та описано вплив алгоритмів фільтрації на проблему інформаційної бульбашки, оглянуто запропоновані стратегії її пом’якшення.

Було здійснено огляд та аналіз існуючих соціальних мереж, що використовуються для розповсюдження фотографій. Визначено ключові проблеми існуючих додатків, які пов’язані з використанням персоналізованих та групових алгоритмів фільтрації підбору контенту, що призводить до утворення бульбашок фільтрів, і як, наслідок - підвищення рівня поляризації та дезінформації в суспільстві. Наявність великої кількості функціоналу що перегружає інтерфейс користувача та ускладнює використання та складність налаштування додатку також є важливими недоліками існуючих програм.

Проведено огляд IT-засобів для розробки програмного забезпечення. В результаті було обрано серверний фреймворк Java - Spring для спрощення процесу написання бекенду та React фреймворк - Nextjs для створення динамічних веб-сайтів.

Було сформовано вимоги до клієнт-серверного застосунку соціальної мережі та спроектовано архітектуру.

Спроектовано прототипи основних сторінок веб додатку засобами Figma.

Розроблено клієнт-серверний застосунок соціальної мережі для розповсюдження фотографій.

Описано реалізацію розробленого серверу системи діаграмами класів. Було представлено мапи сайту для кращого розуміння маршрутизації веб додатку та додано основні екранні форми.

Результатом роботи є додаток соціальної мережі з можливістю прозорого вибору способу фільтрації контенту. Також, було проведено тестування програмного застосунку, для підтвердження його відповідності вимогам.

Тестування проходило за допомогою тест кейсів та тестових наборів даних на базі класів еквівалентності та кордонних умов.

Апробація результатів дослідження:

1. Андрійчук І.Р., Шевченко С.М. Розробка клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків SPRING ТА NEXTJS. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м.Київ. К.: ДУІКТ, 2024. С. 20-21.
2. Андрійчук І.Р., Негоденко О.В., Шевченко С.М. Мінімізація негативних впливів від інформаційної бульбашки при розробці клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків spring та nextjs. Зв’язок, 2024. в.4.

ПЕРЕЛІК ПОСИЛАНЬ

1. Statista Research Department, Apr 29, 2024. Global social network penetration rate as of April 2024, by region. URL: https://www.statista.com/statistics/269615/social-network-penetration-by-region/
2. Stacy Jo Dixon, Aug 29, 2023. Number of social media users worldwide from 2017 to 2027. URL: https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/
3. Stacy Jo Dixon, Apr 10, 2024. Daily time spent on social networking by internet users worldwide from 2012 to 2024. URL: https://www.statista.com/statistics/433871/daily-social-media-usage-worldwide/
4. Valentina Dencheva, May 26, 2023. Types of most engaging content for social media marketing according to consumers in the United States as of April 2022. URL: https://www.statista.com/statistics/1273876/content-social-media-marketing-usa/
5. Stacy Jo Dixon, Apr 28, 2022. Share of internet users worldwide who believe that social media platforms have had an impact on selected aspects of daily life as of February 2019. URL: https://www.statista.com/statistics/1015131/impact-of-social-media-on-daily-life-worldwide/
6. Jean Springsteen, William Yeoh, Dino Christenson. Algorithmic Filtering, Out-Group Stereotype, and Polarization on Social Media. URL: https://yeoh-lab.wustl.edu/assets/pdf/aamas-SpringsteenYC24.pdf
7. Engin Bozdag, Jeroen van den Hoven. Breaking the filter bubble: democracy and design. *Ethics and Information Technology*. 2015. Vol. 17, no. 4. P. 249–265. URL: https://link.springer.com/article/10.1007/s10676-015-9380-y
8. Holone H. The filter bubble and its effect on online personal health information. *Croatian Medical Journal*. 2016. Vol. 57, no. 3. P. 298–301. URL: https://doi.org/10.3325/cmj.2016.57.298 .
9. Shayan A. Tabrizi, Azadeh Shakery. Perspective-based search: a new paradigm for bursting the information bubble. FACETS. 5 August 2019. URL: https://www.facetsjournal.com/doi/full/10.1139/facets-2019-0002

10. Stacy Jo Dixon, Apr 29, 2024. Most popular social networks worldwide as of April 2024, ranked by number of monthly active users. URL: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/
11. Instagram. URL: https://www.instagram.com/
12. About Instagram. URL: https://help.instagram.com/424737657584573
13. VERO. URL: https://vero.co/
14. Data Protection Guide for Candidates. URL: https://veroscreening.com/wp-content/uploads/2016/12/Data-Protection-Guide-for-Candidates.pdf
15. Pixelfed. URL: https://pixelfed.org/
16. Flickr. URL: https://www.flickr.com/
17. PYPL PopularitY of Programming Language. URL: https://pypl.github.io/PYPL.html
18. TIOBE Index for May 2024. URL: https://www.tiobe.com/tiobe-index/
19. Spring Framework. URL: https://spring.io/projects/spring-framework
20. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/
21. Amazon S3. URL: https://aws.amazon.com/s3/
22. TypeScript for the New Programmer. URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html
23. JavaScript frameworks and libraries. URL: https://www.jetbrains.com/lp/devecosystem-2023/javascript/#js_frameworks
24. Lionel Sujay Vailshery, Jan 19, 2024. Most used web frameworks among developers worldwide, as of 2023. URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/
25. Why did we build Visual Studio Code? URL: https://code.visualstudio.com/docs/editor/whyvscode

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків Spring та Nextjs

Виконала студентка 4 курсу
групи ПД-43
Андрійчук Іванна Ростиславівна
Керівник роботи
К.п.н., доц., доцент кафедри ІПЗ Шевченко Світлана Миколаївна

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – покращення процесу розповсюдження фотографій за допомогою клієнт-серверного застосунку соціальної мережі.
- **Об'єкт дослідження** – процес розповсюдження фотографій за допомогою соціальних мереж.
- **Предмет дослідження** – клієнт-серверний застосунок для розповсюдження фотографій.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз літературних джерел з проблеми алгоритмічної фільтрації підбору контенту у соціальних мережах, визначити та описати вплив алгоритмів фільтрації на проблему інформаційної бульбашки, оглянути запропоновані варіанти її розв'язання.
2. Здійснити огляд та аналіз існуючих соціальних мереж, що використовуються для розповсюдження фотографій, визначити їх переваги та недоліки.
3. Провести огляд засобів для розробки програмного забезпечення клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій.
4. Сформувати вимоги до розробки клієнт-серверного застосунку.
5. Спроектувати архітектуру клієнт-серверного застосунку соціальної мережі.
6. Розробити клієнт-серверний застосунок соціальної мережі для розповсюдження фотографій.
7. Провести тестування клієнт-серверного застосунку соціальної мережі.

АНАЛІЗ АНАЛОГІВ

Показник	Instagram	VERO	Pixelfed	SPSP
Наявність веб-платформи:	+	-	+	+
Хронологічне відображення контенту	-	+	Залежить від налаштувань	Залежить від налаштувань
Відсутність фільтрів персоналізованого підбору контенту	-	+	+	+
Прозорість підбору контенту	-	+	+	+
Легкість налаштування	+	+	-	+
Можливість відображення випадкового контенту	+	-	-	+
Основна цільова аудиторія	Поціновувачі візуального спілкування	Поціновувачі приватності та контролю над своїм контентом	Поціновувачі конфіденційності та децентралізації у соціальних мережах	Поціновувачі можливості усвідомленого вибору контенту

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

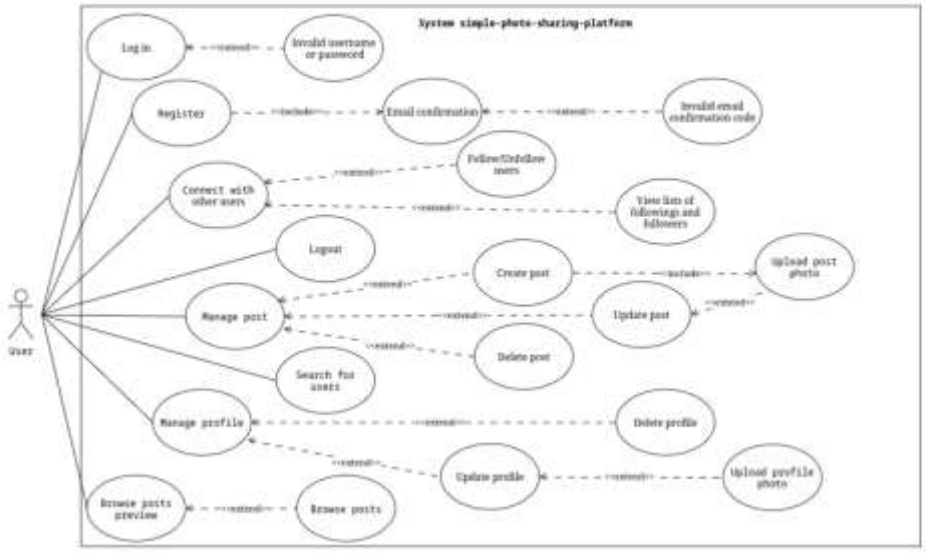
1. Реєстрація та авторизація, розлогування.
2. Можливість роботи з профілем (редагування профілю, його видалення).
3. Можливість роботи з постами (створення, редагування та видалення).
4. Можливість перегляду профілів та публікацій інших користувачів.
5. Можливість взаємодіяти з іншими користувачами.
6. Можливість пошуку інших користувачів.
7. Можливість свідомого вибору фільтрів підбору контенту новинної стрічки.

Нефункціональні вимоги:

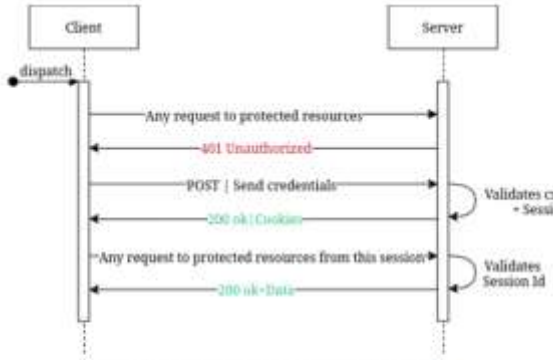
1. Англійська мова локалізації.
2. Взаємодія з іншими користувачами через систему підписок
3. Пошук користувачів по імені користувача.
4. Вибір фільтру підбору контенту новинної стрічки по друзям або випадковий підбір.
5. Захищеність від міжсайтової підробки запитів (Cross-Site Request Forgery, CSRF).
6. Зберігання паролів в зашифрованому вигляді.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

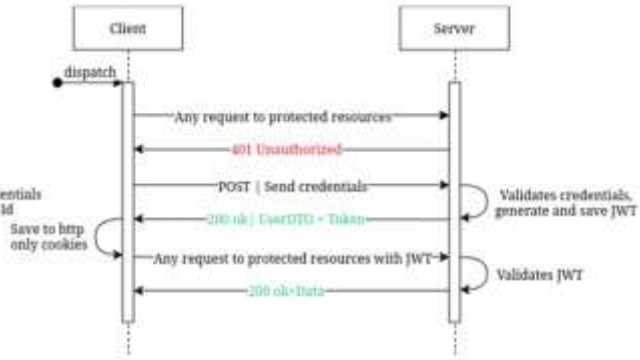
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



ДІАГРАМИ ПОСЛІДОВНОСТІ АВТОРИЗАЦІЇ

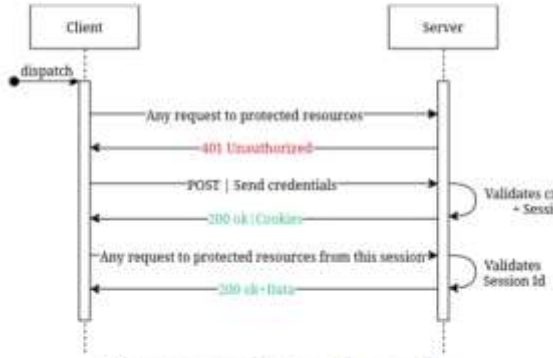


Діаграма послідовності простої авторизації

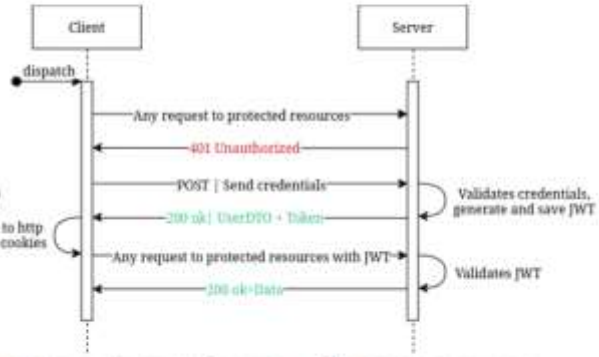


Діаграма послідовності авторизації з використанням JWT

ДІАГРАМИ ПОСЛІДОВНОСТІ АВТОРИЗАЦІЇ

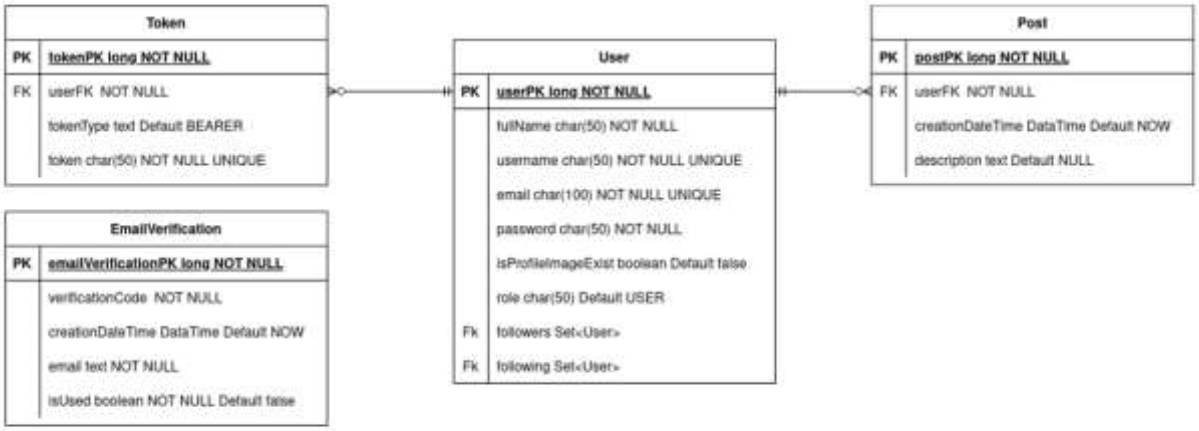


Діаграма послідовності простої авторизації

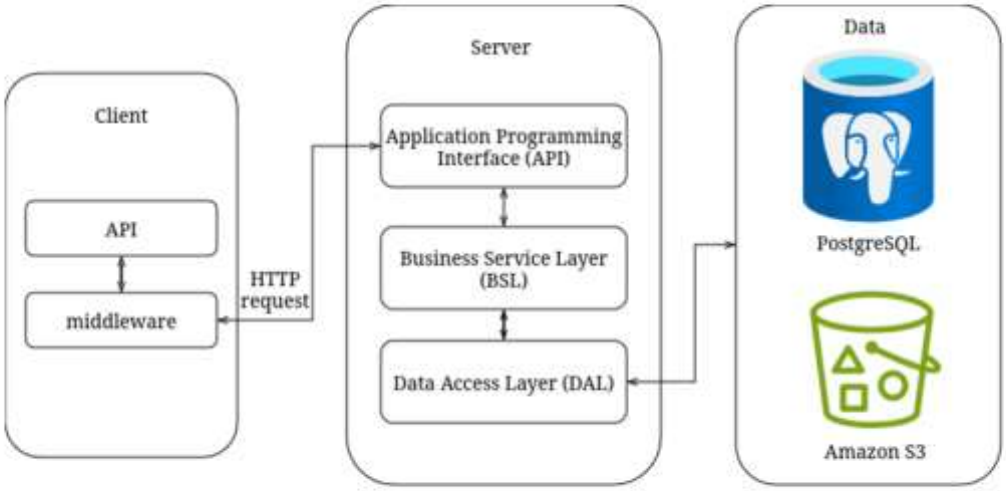


Діаграма послідовності авторизації з використанням JWT

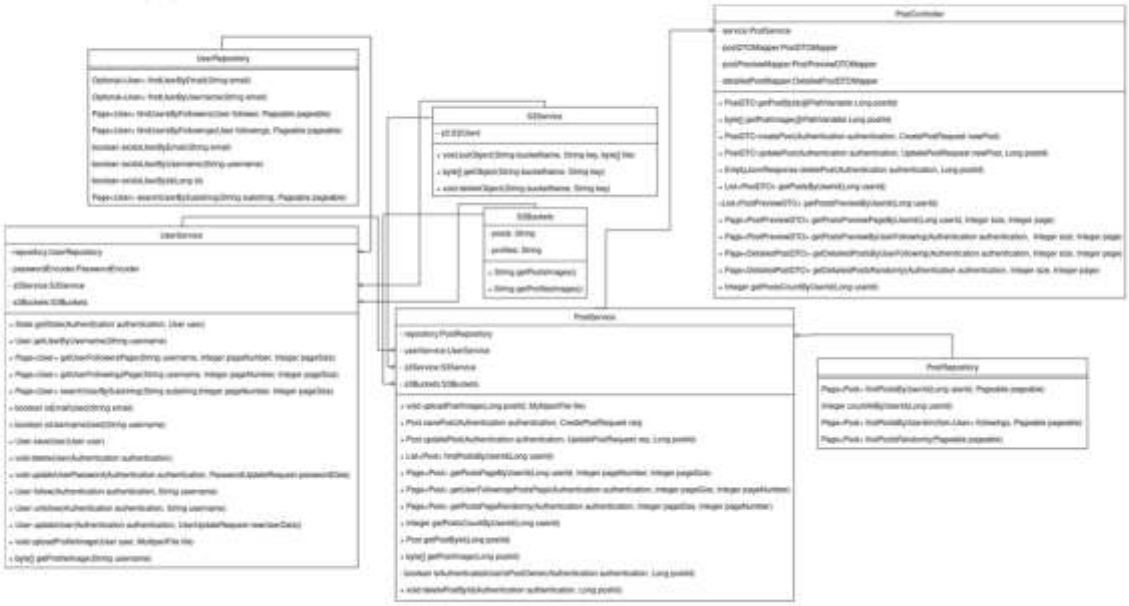
СТРУКТУРА БАЗИ ДАННИХ



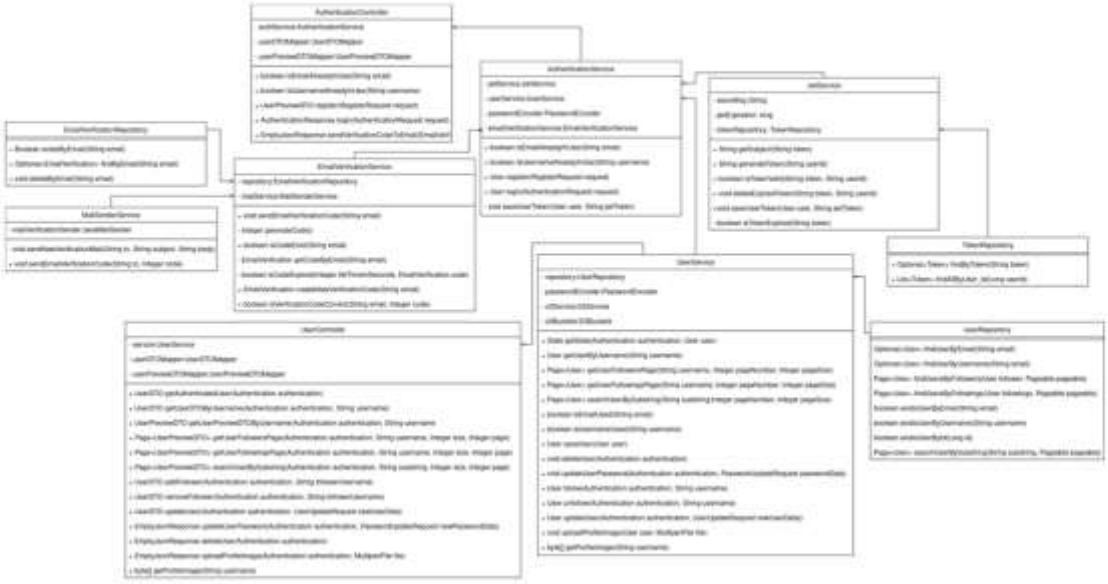
АРХІТЕКТУРА ДОДАТКУ



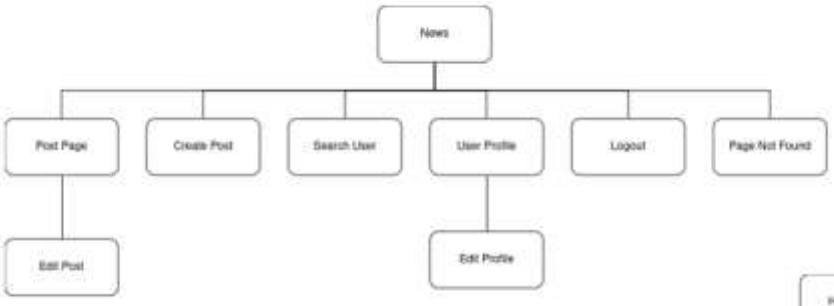
ДІАГРАМА КЛАСІВ ОСНОВНОЇ ЛОГІКИ



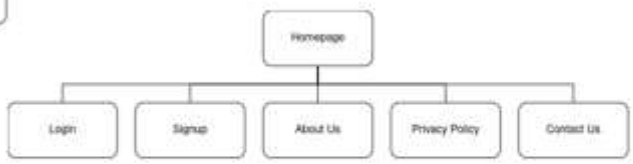
ДІАГРАМА КЛАСІВ АВТОРИЗАЦІЇ



МАПИ ВЕБ-ЗАСТОСУНКУ



Мапа сайту авторизованого користувача

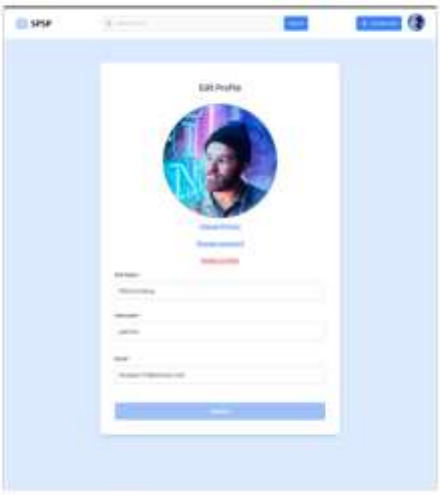


Мапа сайту неавторизованого користувача

ЕКРАННІ ФОРМИ

Реєстрація користувача

Запит для підтвердження електронної пошти



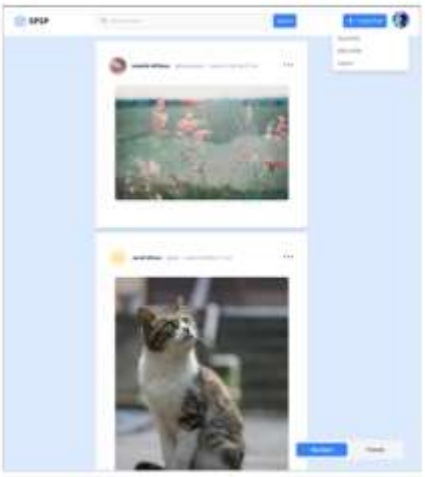
Редагування профілю користувача

ЕКРАННІ ФОРМИ

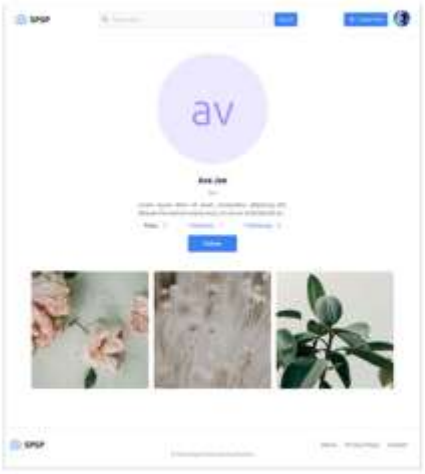
Сторінка посту

Перегляд списку підписок

ЕКРАННІ ФОРМИ



Новинна стрічка з випадковими постами



Профіль користувача

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Андрійчук І.Р., Шевченко С.М. Розробка клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків SPRING ТА NEXTJS. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м.Київ. К.: ДУІКТ, 2024. С. 20-21.
2. Андрійчук І.Р., Негоденко О.В., Шевченко С.М. Мінімізація негативних впливів від інформаційної бульбашки при розробці клієнт-серверного застосунку соціальної мережі для розповсюдження фотографій з використанням фреймворків spring та nextjs. Зв'язок, 2024. в.4.

ВИСНОВКИ

1. Проведено аналіз літературних джерел з проблеми алгоритмічної фільтрації підбору контенту у соціальних мережах.
2. Визначено ключові проблеми існуючих додатків, які пов'язані з використанням прихованих персоналізованих та групових алгоритмів фільтрації підбору контенту, що призводить до утворення бульбашок фільтрів, і як, наслідок - підвищення рівня поляризації та дезінформації в суспільстві.
3. Здійснено аналіз можливостей соціальних мереж для розповсюдження фотографій, визначено їх переваги та недоліки.
4. Сформовано вимоги до клієнт-серверного застосунку соціальної мережі та спроектовано архітектуру.
5. Проаналізовано засоби розробки клієнт-серверного застосунку та обгрунтовано їх вибір.
6. Розроблено клієнт-серверний застосунок, спрямований на розповсюдження фотографій з можливістю прозорого вибору способу фільтрації контенту, з метою мінімізації ефекту інформаційної бульбашки.
7. Проведено тестування програмного застосунку, для підтвердження його відповідності вимогам, за допомогою тест кейсів та тестових наборів даних на базі класів еквівалентності та кордонних умов.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

User.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(
        name = "_user",
        uniqueConstraints = {
        @UniqueConstraint(
                name = "user_email_unique",
                columnNames = "email"
        ),
        }
)
public class User implements UserDetails {

        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private Long id;

        @Column(nullable = false)
        private String fullName;

        @Column(nullable = false)
        private String username;

        @Column(nullable = false)
        private String email;

        @Column(nullable = false)
        private String password;

        @Builder.Default
        private Boolean isProfileImageExist = false;

        @Builder.Default
        @Enumerated(EnumType.STRING)
        private Role role = Role.USER;

        @OneToMany(mappedBy = "user", cascade =
CascadeType.ALL)
        private List<Token> tokens;

        @OneToMany(mappedBy = "user", cascade =
CascadeType.ALL)
        private Set<Post> posts;

        @ManyToMany
        @JoinTable(name = "following",
        joinColumns = @JoinColumn(name = "followingId"),
        inverseJoinColumns = @JoinColumn(name = "followerId")
        )
        private Set<User> followers;

        @ManyToMany
        @JoinTable(name = "following",
        joinColumns = @JoinColumn(name = "followerId"),
        inverseJoinColumns = @JoinColumn(name =
"followingId")
        )
        private Set<User> followings;

        @Override
        public Collection<? extends GrantedAuthority>
getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role.name()));
        }
        @Override
        public boolean isAccountNonExpired() {
        return true;
        }

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

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

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

        @Override
        public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id)
        && Objects.equals(fullName, user.fullName)
        && Objects.equals(username, user.username)
        && Objects.equals(email, user.email)
        && Objects.equals(password, user.password)
        && Objects.equals(isProfileImageExist,
user.isProfileImageExist);
        }

        @Override
        public int hashCode() {
        return Objects.hash(id, fullName, username, email,
password, isProfileImageExist);
        }

        @Override
        public String toString() {
        return "User{" +
        "id=" + id +
        ", fullName='" + fullName + '\'' +
        ", username=" + username + '\'' +
        ", email='" + email + '\'' +
        ", password='" + password + '\'' +
        ", isProfileImageExist='" + isProfileImageExist + '\'' +
        '}';
        }
}
```

State.java

```
package com.xamarsia.simplephotosharingplatform.user;
public enum State {
        CURRENT,
        FOLLOWED,
        UNFOLLOWED;
}
```

Role.java

```
package com.xamarsia.simplephotosharingplatform.user;
public enum Role {
```

```
	USER,
	ADMIN;

	public String getRole() {
	return "ROLE_" + name();
	}
}

UserDTO.java
public record UserDTO(
	@NotBlank(message = "Id is required.")
	Long id,

	@NotBlank(message = "Full name is required.")
	String fullName,

	@NotBlank(message = "Username is required.")
	String username,

	@NotBlank(message = "Email is required.")
	String email,

	@NotBlank(message = "Roles is required.")
	List<String> roles,

	@NotBlank(message = "State is required.")
	@Enumerated(EnumType.STRING)
	State state,

	Boolean isProfileImageExist
) {
}

UserDTOMapper.java
@Service
public class UserDTOMapper {
	private final UserService service;

	public UserDTOMapper(UserService service) {
	this.service = service;
	}

	public UserDTO apply(User user, State userState) {
	return new UserDTO(
	user.getId(),
	user.getFullName(),
	user.getUsername(),
	user.getEmail(),
	user.getAuthorities()
		.stream()
		.map(GrantedAuthority::getAuthority)
		.collect(Collectors.toList()),
	userState,
	user.getIsProfileImageExist()
	);
	}

	public UserDTO apply(Authentication authentication, User
user) {
	return new UserDTO(
	user.getId(),
	user.getFullName(),
	user.getUsername(),
	user.getEmail(),
	user.getAuthorities()
		.stream()
		.map(GrantedAuthority::getAuthority)
		.collect(Collectors.toList()),
	service.getState(authentication, user),
	user.getIsProfileImageExist()
	);
	}
}
UserRepository.java
public interface UserRepository extends JpaRepository<User, Long> {
	Optional<User> findUserByEmail(String email);

	Optional<User> findUserByUsername(String email);

	Page<User> findUsersByFollowers(User follower, Pageable
pageable);

	Page<User> findUsersByFollowings(User followings,
Pageable pageable);

	boolean existsUserByEmail(String email);

	boolean existsUserByUsername(String username);

	boolean existsUserById(Long id);

	@Query(value = "Select * from _user u WHERE
(u.username LIKE CONCAT('%',:substring,'%') OR u.full_name LIKE
CONCAT('%',:substring,'%'))", countQuery = "SELECT count(*)
FROM _user", nativeQuery = true)
	Page<User> searchUserBySubstring(@Param("substring")
String substring, Pageable pageable);
}

UserService.java
@Service
public class UserService {
	private final UserRepository repository;
	private final PasswordEncoder passwordEncoder;
	private final S3Service s3Service;
	private final S3Buckets s3Buckets;

	public UserService(UserRepository repository,
PasswordEncoder passwordEncoder, S3Service s3Service, S3Buckets
s3Buckets) {
	this.repository = repository;
	this.passwordEncoder = passwordEncoder;
	this.s3Service = s3Service;
	this.s3Buckets = s3Buckets;
	}

	@Transactional(readOnly = true)
	public User getByEmail(String email) {
	return repository.findUserByEmail(email).orElseThrow(() -
> new
ResourceNotFoundException(String.format("[GetUserByEmail]: User
not found with email '%s'.", email)));
	}

	@Transactional(readOnly = true)
	public State getState(Authentication authentication, String
username) {
	User currentUser = getAuthenticatedUser(authentication);
	if (Objects.equals(currentUser.getUsername(), username)) {
	return State.CURRENT;
	}
	if (isUserInFollowing(currentUser, username)) {
	return State.FOLLOWED;
	}
	return State.UNFOLLOWED;
	}
```

```
	@Transactional(readOnly = true)
	public State getState(Authentication authentication, User
user) {
	User currentUser = getAuthenticatedUser(authentication);
	if (Objects.equals(currentUser.getUsername(),
user.getUsername())) {
	return State.CURRENT;
	}
	if (isUserInFollowing(currentUser, user.getUsername())) {
	return State.FOLLOWED;
	}
	return State.UNFOLLOWED;
	}

	@Transactional(readOnly = true)
	public User getUserByUsername(String username) {
	return
repository.findUserByUsername(username).orElseThrow(() -> new
ResourceNotFoundException(String.format("[GetUserByUsername]:
User not found with username '%s'.", username)));
	}

	@Transactional(readOnly = true)
	public User getById(Long customerId) {
	return selectUserById(customerId).orElseThrow(() -> new
ResourceNotFoundException(String.format("[GetUserByID]: User not
found with id '%s'.", customerId)));
	}

	public User getAuthenticatedUser(Authentication
authentication) {
	if (authentication instanceof
AnonymousAuthenticationToken) {
	throw new
UnauthorizedAccessException("[GetAuthenticatedUser]: User not
authenticated.");
	}

	String username = authentication.getName();
	return findUserByUsername(username).orElseThrow(() ->
new
ResourceNotFoundException(String.format("[GetUserByUsername]:
User not found with username '%s'.", username)));
	}

	@Transactional(readOnly = true)
	public Page<User> getUserFollowersPage(String username,
Integer pageNumber, Integer pageSize) {
	Pageable pageable = PageRequest.of(pageNumber,
pageSize);
	User user = getUserByUsername(username);
	return repository.findUsersByFollowings(user, pageable);
	}

	@Transactional(readOnly = true)
	public Page<User> getUserFollowingsPage(String
username, Integer pageNumber, Integer pageSize) {
	Pageable pageable = PageRequest.of(pageNumber,
pageSize);
	User user = getUserByUsername(username);
	return repository.findUsersByFollowers(user, pageable);
	}

	@Transactional(readOnly = true)
	public Page<User> searchUserBySubstring(String
substring,Integer pageNumber, Integer pageSize) {
	Pageable pageable = PageRequest.of(pageNumber,
pageSize);
	return repository.searchUserBySubstring(substring,
pageable);
	}

	public Optional<User> selectUserById(Long id) {
	return repository.findById(id);
	}

	public User saveUser(User user) {
	return repository.save(user);
	}

	public void deleteUser(Authentication authentication) {
	User user = getAuthenticatedUser(authentication);
	if (user.getIsProfileImageExist()) {
	s3Service.deleteObject(s3Buckets.getProfilesImages(),
user.getId().toString());
	}
	repository.deleteById(user.getId());
	}

	public User updateUserPassword(Authentication
authentication, PasswordUpdateRequest passwordData) {
	User user = getAuthenticatedUser(authentication);
	String oldPassword = passwordData.getOldPassword();
	String newPassword = passwordData.getNewPassword();

	if (!passwordEncoder.matches(oldPassword,
user.getPassword())) {
	throw new
BadCredentialsException("[UpdateUserPassword]: Wrong
confirmation password.");
	}
	user.setPassword(passwordEncoder.encode(newPassword));
	return saveUser(user);
	}

	public User follow(Authentication authentication, String
username) {
	User user = getAuthenticatedUser(authentication);

	User follower =
repository.findUserByUsername(username).orElseThrow(() -> new
ResourceNotFoundException(String.format("[Follow]: Follower not
found with username '%s'.", username)));
	if (Objects.equals(user.getUsername(),
follower.getUsername())) {
	throw new
IllegalArgumentException(String.format("[Follow]: Invalid parameter.
User and his follower can't have the same username '%s'.", username));
	}
	user.getFollowings().add(follower);
	saveUser(follower);
	return saveUser(user);
	}

	public User unfollow(Authentication authentication, String
username) {
	User user = getAuthenticatedUser(authentication);

	User follower =
repository.findUserByUsername(username).orElseThrow(() -> new
ResourceNotFoundException(String.format("[Unfollow]: Follower not
found with username '%s'.", username)));
	if (Objects.equals(user.getUsername(),
follower.getUsername())) {
	throw new
IllegalArgumentException(String.format("[Unfollow]: Invalid
```

```
parameter. User and his follower can't have the same username '%s'.",
username));
        }
        user.getFollowings().remove(follower);
        saveUser(follower);
        return saveUser(user);
        }

        public Boolean isUserInFollowing(Authentication
authentication, String username) {
        User user = getAuthenticatedUser(authentication);
        return isUserInFollowing(user, username);
        }

        public Boolean isUserInFollowing(User currentUser, String
username) {
        return
currentUser.getFollowings().stream().map(User::getUsername).anyMat
ch(username::equals);
        }

        public User updateUser(Authentication authentication,
UserUpdateRequest newUserData) {
        User user = getAuthenticatedUser(authentication);

        String newUsername = newUserData.getUsername();
        String newEmail = newUserData.getEmail();

        if (!Objects.equals(user.getEmail(), newEmail) &&
repository.existsUserByEmail(newEmail)) {
        throw new
IllegalArgumentException(String.format("[UpdateUser]: User with
email '%s' already exist.", newEmail));
        }

        if (!Objects.equals(user.getUsername(), newUsername) &&
repository.existsUserByUsername(newUsername)) {
        throw new
IllegalArgumentException(String.format("[UpdateUser]: User with
username '%s' already exist.", newUsername));
        }

        user.setFullName(newUserData.getFullName());
        user.setUsername(newUserData.getUsername());
        user.setEmail(newUserData.getEmail());

        return saveUser(user);
        }

        private Optional<User> findUserByUsername(String
username) {
        return repository.findUserByUsername(username);
        }

        public Optional<User> selectUserByEmail(String email) {
        return repository.findUserByEmail(email);
        }

        public void uploadProfileImage(User user, MultipartFile
file) {
        if (file.isEmpty()) {
        throw new
IllegalArgumentException("[UploadProfileImage]: File is empty.
Cannot save an empty file");
        }
        String extension =
Objects.requireNonNull(file.getOriginalFilename()).split("\\.")[1];
        if (!(Objects.equals(extension, "jpg") ||
Objects.equals(extension, "jpeg "))) {
        throw new
IllegalArgumentException(String.format("[UploadProfileImage]:
Wrong file extension '%s' found. "
        + "Only .jpg and .jpeg files are allowed.", extension));
        }
        try {
        s3Service.putObject(s3Buckets.getProfilesImages(),
user.getId().toString(), file.getBytes());
        } catch (IOException e) {
        throw new
ApplicationException(ApplicationError.AWS_S3_ERROR,
"[UploadProfileImage]: " + e.getMessage());
        }

        user.setIsProfileImageExist(true);
        saveUser(user);
        }

        @Transactional(readOnly = true)
        public byte[] getProfileImage(String username) {
        User user = getUserByUsername(username);
        String key = user.getIsProfileImageExist() ?
user.getId().toString() : "default";
        return s3Service.getObject(s3Buckets.getProfilesImages(),
key);
        }
}
```

UserController.java

```
@RestController
@RequestMapping("/user")
@RequiredArgsConstructor
public class UserController {
        private final UserService service;
        private final UserDTOMapper userDTOMapper;
        private final UserPreviewDTOMapper
userPreviewDTOMapper;

        @GetMapping
        public ResponseEntity<UserDTO>
getAuthenticatedUser(Authentication authentication) {
        UserDTO userDTO =
userDTOMapper.apply(authentication,
service.getAuthenticatedUser(authentication));
        return ResponseEntity.ok().body(userDTO);
        }

        @GetMapping("/{username}")
        public ResponseEntity<UserDTO>
getUserDTOByUsername(Authentication authentication,
        @PathVariable String username) {
        User user = service.getUserByUsername(username);
        UserDTO userDTO =
userDTOMapper.apply(authentication, user);
        return ResponseEntity.ok().body(userDTO);
        }

        @GetMapping("/preview/{username}")
        public ResponseEntity<UserPreviewDTO>
getUserPreviewDTOByUsername(Authentication authentication,
        @PathVariable String username) {
        User user = service.getUserByUsername(username);
        UserPreviewDTO userPreviewDTO =
userPreviewDTOMapper.apply(authentication, user);
        return ResponseEntity.ok().body(userPreviewDTO);
        }


        @GetMapping("/{username}/followers/page")
```

```
	public Page<UserPreviewDTO>
getUserFollowersPage(Authentication authentication,
	@PathVariable String username,
	@RequestParam Integer size,
	@RequestParam Integer page) {
	Page<User> followersPage =
service.getUserFollowersPage(username, page, size);
	List<UserPreviewDTO> followingsPreviewDTO =
followersPage.stream()
	.map(follower ->
userPreviewDTOMapper.apply(authentication,
follower)).collect(Collectors.toList());
	return new PageImpl<>(followingsPreviewDTO,
followersPage.getPageable(), followersPage.getTotalElements());
	}

	@GetMapping("/{username}/followings/page")
	public Page<UserPreviewDTO>
getUserFollowingsPage(Authentication authentication,
	@PathVariable String username,
	@RequestParam Integer size,
	@RequestParam Integer page) {
	Page<User> followingsPage =
service.getUserFollowingsPage(username, page, size);
	List<UserPreviewDTO> followingsPreviewDTO =
followingsPage.stream()
	.map(following ->
userPreviewDTOMapper.apply(authentication,
following)).collect(Collectors.toList());
	return new PageImpl<>(followingsPreviewDTO,
followingsPage.getPageable(), followingsPage.getTotalElements());
	}

	@GetMapping("/search/page")
	public Page<UserPreviewDTO>
searchUserBySubstring(Authentication authentication,
	@RequestParam String substring,
	@RequestParam Integer size,
	@RequestParam Integer page) {
	Page<User> searchedPage =
service.searchUserBySubstring(substring, page, size);

	List<UserPreviewDTO> followingsPreviewDTO =
searchedPage.stream()
	.map(following ->
userPreviewDTOMapper.apply(authentication,
following)).collect(Collectors.toList());
	return new PageImpl<>(followingsPreviewDTO,
searchedPage.getPageable(), searchedPage.getTotalElements());
	}

	@PutMapping("/{followerUsername}/follow")
	public ResponseEntity<?> addFollower(Authentication
authentication,
	@PathVariable String followerUsername) {
	User user = service.follow(authentication,
followerUsername);
	UserDTO userDTO =
userDTOMapper.apply(authentication, user);
	return ResponseEntity.ok().body(userDTO);
	}

	@PutMapping("/{followerUsername}/unfollow")
	public ResponseEntity<?> removeFollower(Authentication
authentication,
	@PathVariable String followerUsername) {
	User user = service.unfollow(authentication,
followerUsername);
	UserDTO userDTO =
userDTOMapper.apply(authentication, user);
	return ResponseEntity.ok().body(userDTO);
	}

	@PutMapping("/update")
	public ResponseEntity<?> updateUser(Authentication
authentication, @RequestBody @Valid UserUpdateRequest
newUserData) {
	User updatedUser = service.updateUser(authentication,
newUserData);
	UserDTO userDTO = userDTOMapper.apply(updatedUser,
State.CURRENT);
	return ResponseEntity.ok().body(userDTO);
	}

	@PutMapping("/password/update")
	public ResponseEntity<?>
updateUserPassword(Authentication authentication,
	@RequestBody @Valid PasswordUpdateRequest
newPasswordData) {
	User updatedUser =
service.updateUserPassword(authentication, newPasswordData);
	UserDTO userDTO = userDTOMapper.apply(updatedUser,
State.CURRENT);
	return ResponseEntity.ok()
	.body(userDTO);
	}

	@DeleteMapping
	public ResponseEntity<?> deleteUser(Authentication
authentication) {
	service.deleteUser(authentication);
	return ResponseEntity.status(HttpStatus.OK).body(new
EmptyJsonResponse());
	}

	@PostMapping(value = "/profile/image", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
	public ResponseEntity<?> uploadProfileImage(
	Authentication authentication,
	@RequestParam("file") MultipartFile file) {
	User user = service.getAuthenticatedUser(authentication);
	service.uploadProfileImage(user, file);
	return ResponseEntity.status(HttpStatus.OK).body(new
EmptyJsonResponse());
	}

	@GetMapping(value = "{username}/profile/image",
produces = MediaType.IMAGE_JPEG_VALUE)
	public byte[] getProfileImage(@PathVariable("username")
String username) {
	return service.getProfileImage(username);
	}
}
```

Token.java

```
@Data
@Builder
@Entity
@NoArgsConstructor
@AllArgsConstructor
public class Token {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	public long id;

	@Column(unique = true)
```

```
	public String token;

	@Enumerated(EnumType.STRING)
	@Builder.Default
	public TokenType tokenType = TokenType.BEARER;

	@ManyToOne(cascade = CascadeType.DETACH)
	@JoinColumn(name = "user_id")
	public User user;
}

TokenRepository.java
public interface TokenRepository extends JpaRepository<Token,
Long> {
	Optional<Token> findByToken(String token);
}

TokenType.java
public enum TokenType {
	BEARER
}

JwtService.java
@Service
public class JwtService {

	@Value("${application.security.jwt.secret-key}")
	private String secretKey;

	@Value("${application.security.jwt.expiration}")
	private long jwtExpiration;

	private final TokenRepository tokenRepository;

	public JwtService(TokenRepository tokenRepository) {
	this.tokenRepository = tokenRepository;
	}

	public String getSubject(String token) {
	return extractAllClaims(token).getSubject();
	}

	public String generateToken(String userId) {
	return generateToken(new HashMap<>(), userId);
	}

	public String generateToken(Map<String, Object>
extraClaims, String userId) {
	return Jwts.builder()
	.setClaims(extraClaims)
	.setSubject(userId)
	.setIssuedAt(new Date(System.currentTimeMillis()))
	.setExpiration(new Date(System.currentTimeMillis() +
jwtExpiration))
	.signWith(getSigningKey(), SignatureAlgorithm.HS256)
	.compact();
	}

	public boolean isTokenValid(String token, String userId) {
	final String userIdFromToken = getSubject(token);
	return (userIdFromToken.equals(userId) &&
!isTokenExpired(token) && isTokenExist(token));
	}

	private boolean isTokenExist(String token) {
	Token storedToken =
tokenRepository.findByToken(token).orElse(null);

	return (storedToken != null);
	}

	private boolean isTokenExpired(String token) {
	Date today = Date.from(Instant.now());
	return
extractAllClaims(token).getExpiration().before(today);
	}

	private Claims extractAllClaims(String token) {
	Claims claims =
Jwts.parserBuilder().setSigningKey(getSigningKey()).build().parseClai
msJws(token).getBody();
	return claims;
	}

	public void deleteExpiredToken(String token, String userId)
{
	if (!isTokenValid(token, userId)) {
	Token storedToken =
tokenRepository.findByToken(token).orElse(null);

	if (storedToken != null) {
	tokenRepository.deleteById(storedToken.id);
	}
	}
	}

	private Key getSigningKey() {
	byte[] keyBytes = Decoders.BASE64.decode(secretKey);
	return Keys.hmacShaKeyFor(keyBytes);
	}

	public void deleteAllUserTokens(Long userId) {
	var validUserTokens =
tokenRepository.findAllByUser_Id(userId);
	if (validUserTokens.isEmpty())
	return;

	validUserTokens.forEach(storedToken -> {
	tokenRepository.deleteById(storedToken.id);
	});
	}

	public void saveUserToken(User user, String jwtToken) {
	var token = Token.builder()
	.user(user)
	.token(jwtToken)
	.tokenType(TokenType.BEARER)
	.build();
	tokenRepository.save(token);
	}
}

S3Buckets.java
@Setter
@Configuration
@ConfigurationProperties(prefix = "aws.s3.buckets")
public class S3Buckets {
	private String posts;
	private String profiles;

	public String getPostsImages() {
	System.out.println("Posts: " + posts);
	return posts;
	}

	public String getProfilesImages() {
	System.out.println("Profiles: " + profiles);
	return profiles;
```

```
	}
}

S3Config.java
@Configuration
public class S3Config {

	@Value("${aws.region}")
	private String awsRegion;

	@Bean
	public S3Client s3Client() {
	S3Client client = S3Client.builder()
	.region(Region.of(awsRegion))
	.build();
	return client;
	}
}

S3Service.java
@Service
public class S3Service {
	private final S3Client s3;

	public S3Service(S3Client s3Client) {
	this.s3 = s3Client;
	}

	public void putObject(String bucketName, String key,
byte[] file) {
	PutObjectRequest objectRequest =
PutObjectRequest.builder()
	.bucket(bucketName)
	.key(key)
	.build();
	try {
	s3.putObject(objectRequest, RequestBody.fromBytes(file));
	} catch (Exception e) {
	throw new AWSException("[S3Service]: putObject error
with key " + key + "\n Error: " + e.getMessage());
	}
	}

	public byte[] getObject(String bucketName, String key) {
	GetObjectRequest getObjectRequest =
GetObjectRequest.builder()
	.bucket(bucketName)
	.key(key)
	.build();
	try {
	ResponseInputStream<GetObjectResponse> res =
s3.getObject(getObjectRequest);
	return res.readAllBytes();
	} catch (Exception e) {
	throw new AWSException("[S3Service]: getObject error
with key " + key + "\n Error: " + e.getMessage());
	}
	}

	public void deleteObject(String bucketName, String key) {
	DeleteObjectRequest deleteObjectRequest =
DeleteObjectRequest.builder()
	.bucket(bucketName)
	.key(key)
	.build();
	try {
	s3.deleteObject(deleteObjectRequest);
	} catch (Exception e) {
	throw new AWSException("[S3Service]: deleteObject error
with key " + key + "\n Error: " + e.getMessage());
	}
	}
}

EmailVerification.java
@Data
@Builder
@Entity
@NoArgsConstructor
@AllArgsConstructor
public class EmailVerification {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	public long id;

	@Column(nullable = false)
	public Integer verificationCode;

	@Column(nullable = false)
	public final LocalDateTime creationDateTime =
LocalDateTime.now();

	@Column(nullable = false)
	public String email;

	@Column(nullable = false)
	@Builder.Default
	public Boolean isUsed = false;
}

Post.java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Post {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	public final LocalDateTime creationDateTime =
LocalDateTime.now();

	private String description;

	@ManyToOne
	@JoinColumn(name = "user_id", nullable = false)
	private User user;
}

EmailVerificationRepository.java
public interface EmailVerificationRepository extends
JpaRepository<EmailVerification, Long> {
	Boolean existsByEmail(String email);

	Optional<EmailVerification> findByEmail(String email);

	void deleteByEmail(String email);
}

EmailVerificationService.java
@Service
@AllArgsConstructor
public class EmailVerificationService {
	private final EmailVerificationRepository repository;
	private final MailSenderService mailService;
```

```
	public void sendEmailVerificationCode(@NotNull
@NotEmpty String email) {
	EmailVerification verificationCode =
createNewVerificationCode(email);
	mailService.sendEmailVerificationCode(email,
verificationCode.verificationCode);
	}

	private Integer generateCode() {
	int length =
ApplicationConstants.Validation.EMAIL_VERIFICATION_CODE_L
ENGTH;
	boolean useLetters = false;
	boolean useNumbers = true;
	String code = RandomStringUtils.random(length,
useLetters, useNumbers);
	return Integer.parseInt(code);
	}

	public boolean isCodeExist(String email) {
	return repository.existsByEmail(email);
	}

	private EmailVerification getCodeByEmail(String email) {
	return repository.findByEmail(email)
	.orElseThrow(() -> new
ResourceNotFoundException(String.format("[GetEmailVerificationCo
deByEmail]: Email verification code not found with email '%s'.",
email)));
	}

	private boolean isCodeExpired(Integer lifeTimeInSeconds,
EmailVerification code) {
	return
ChronoUnit.SECONDS.between(code.creationDateTime,
LocalDateTime.now()) >= lifeTimeInSeconds;
	}

	public EmailVerification createNewVerificationCode(String
email) {
	if (isCodeExist(email)) {
	EmailVerification code = getCodeByEmail(email);
	boolean isCodeExpired =
isCodeExpired(ApplicationConstants.Validation.FAILED_EMAIL_VE
RIFICATION_DELAY_TIME, code);
	if (isCodeExpired) {
	code.setVerificationCode(generateCode());
	code.setIsUsed(false);
	} else {
	throw new
TooManyRequestsException("[CreateNewVerificationCode]: Email
verification code cannot be created for less than 30 seconds.");
	}
	return repository.save(code);
	}
	EmailVerification code =
	EmailVerification.builder()
		.verificationCode(generateCode())
		.email(email)
		.isUsed(false)
		.build();

	return repository.save(code);
	}

	public boolean isVerificationCodeCorrect(String email,
Integer code) {
	EmailVerification savedCode = getCodeByEmail(email);
	if
(isCodeExpired(ApplicationConstants.Validation.EMAIL_VERIFICA
TION_CODE_LIFETIME_IN_SECONDS, savedCode)) {
	throw new
InvalidEmailVerification("[IsVerificationCodeCorrect]: Email
verification code has expired. Please, create a new one.");
	} else if (savedCode.isUsed) {
	throw new
InvalidEmailVerification("[IsVerificationCodeCorrect]: Email
verification code cannot be used twice. Please, wait " +

	(ApplicationConstants.Validation.EMAIL_VERIFICATIO
N_RESEND_TIME / 60) +
	" minutes and create new one.");
	} else if (!Objects.equals(savedCode.getVerificationCode(),
code)) {
	savedCode.setIsUsed(true);
	repository.save(savedCode);
	return false;
	}
	savedCode.setIsUsed(true);
	repository.save(savedCode);
	return true;
	}
}
```

api/auth/[...path].tsx

```
import type { NextApiRequest, NextApiResponse } from 'next'
import httpProxyMiddleware from 'next-http-proxy-middleware'
import Cookies from 'cookies'

export const config = {
 api: {
	bodyParser: false,
	sizeLimit: '10mb',
	externalResolver: true,
 },
};

export default function authorizedRequest(
 req: NextApiRequest,
 res: NextApiResponse) {

 httpProxyMiddleware(req, res, {
	changeOrigin: true,
	target: process.env.API_BASE_URL,
	pathRewrite: [
	{
	patternStr: '/api/auth',
	replaceStr: ''
	}
	],

	onProxyInit: proxy => {
	proxy.on('proxyReq', proxyReq => {
	try {
	let authorization: string | undefined =
req.headers.authorization;
	if (!authorization) {
	const cookies = new Cookies(req, res);
	authorization = cookies.get('authorization');

	if (!authorization) {
	throw new Error("[api/auth]: Authorisation requered!");
	}
	proxyReq.setHeader('authorization', authorization)
	}
	} catch (error: unknown) {
	if (error instanceof Error) {
```

```
            console.error("[api/auth]: An unexpected error happened
occurred during proxyReq: ", error.message);
            }
            }
            })
            }
  })
}

api/unauth/[...path].tsx
export const config = {
  api: {
            bodyParser: false,
            sizeLimit: '10mb',
  },
};

export default function unauthorizedRequest(
  req: NextApiRequest,
  res: NextApiResponse) {

  httpProxyMiddleware(req, res, {
            target: process.env.API_BASE_URL,
            pathRewrite: [
            {
            patternStr: '/api/unauth',
            replaceStr: ''
            }
            ],
  })
}
api/login.tsx
export const config = {
            api: {
            bodyParser: true,
            externalResolver: true,
            },
};

export default function login(
            })            }))}}))}
            req: NextApiRequest,
            res: NextApiResponse) {

            httpProxyMiddleware(req, res, {
            changeOrigin: true,
            target: process.env.API_BASE_URL,
            selfHandleResponse: true,
            pathRewrite: [
            {
            patternStr: '/api',
            replaceStr: '/auth'
            }
            ],
            onProxyInit: proxy => {
            proxy.on('proxyRes', proxyRes => {
            let responseBody = '';

            proxyRes.on('data', (chunk) => {
            responseBody += chunk;
            })

            proxyRes.on('end', () => {
            try {
                        const { token, userDTO } =
JSON.parse(responseBody);
                        const cookies = new Cookies(req, res);
                        if (token) {
                        cookies.set('authorization', `Bearer ${token}`, {
                        httpOnly: true,
                        sameSite: 'lax', // CSRF protection
                        });
                        }
                        return res.status(200).send(userDTO);
            } catch (error: unknown) {
                        if (error instanceof Error) {
                        console.error("[api/login]: An unexpected error
happened occurred: ", error.message);
                        }
            }
```