

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Фан-сайт К-рор спільноти»

Виконала студентка групи ІПЗ-20-4
ХАРЧЕНКО Юлія Валеріївна

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ТОЛСТОЙ Ігор Анатолійович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ХАРЧЕНКО Юлія Валеріївна**

Керівник роботи: **ЧИЖМОТРИЯ Олексій Володимирович**

Тема роботи: **«Фан-сайт К-рор спільноти»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74с**

Вихідні дані для роботи: **методи та засоби розробки веб-сайтів, зокрема використання веб-орієнтованих технологій для розробки системи фан-сайту.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	26.02.2024р.	26.02.2024р.
2	Чижмотря О.В.	26.02.2024р.	26.02.2024р.
3	Чижмотря О.В.	26.02.2024р.	26.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-додатку фан-сайту та пояснювальної записки. Пояснювальна записка до випускної роботи містить 76 сторінок, 46 ілюстрацій та 11 таблиць.

Основною метою даної роботи є створення алгоритмів функціонування та реалізація веб-сайту для фан-спільноти К-рор.

Методи, використані під час створення фан-сайту, включаючи в себе програмування на веб-технологіях, роботу з графічними програмами для дизайну та аналіз потреб аудиторії.

Результати розробки поляють у створенні інтерактивної та привабливої платформи для фанатів К-рор, яка надає зручний доступ до інформації та обміну контентом.

КЛЮЧОВІ СЛОВА: NODE.JS, REACT.JS, MONGODB, ВЗАЄМОДІЯ, ГРУПИ, ПОСТИ, ФАНАТИ, ФАН-САЙТ.

					ІПЗ.КР.Б-121-24-ІПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Харченко Ю.В.			Фан-сайт К-рор спільноти			
Керівник		Чижмотря О.В.						
Рецензен.		Толстой І.А.						
Зав. каф.		Вакалюк Т.А.						
						Літ.	Арк.	Аркушів
							3	76
						Житомирська політехніка, група ІПЗ-20-4		

ABSTRACT

The bachelor's thesis consists of a fan site web application and an explanatory note. The explanatory note to the final thesis contains 76 pages, 46 illustrations and 11 tables.

The main goal of this work is to create functioning algorithms and implement a website for the K-pop fan community.

The techniques used in the creation of the fan site included programming on web technologies, working with graphic design programs and analyzing the needs of the audience.

The result of the development was to create an interactive and attractive platform for K-pop fans that provides convenient access to information and content sharing.

KEYWORDS: NODE.JS, REACT.JS, MONGODB, INTERACTION, GROUPS, POSTS, FANS, FAN SITE.

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ФАН-САЙТУ	9
1.1 Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	11
1.3. ВИБІР АРХІТЕКТУРИ ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ФАН-САЙТУ	17
1.4. ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ	20
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ФАН-САЙТУ	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	23
2.2. Розробка бази даних системи	28
2.3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ СИСТЕМИ	32
2.4 РЕАЛІЗАЦІЯ ДОДАТКУ ФАН-САЙТУ	33
Висновки до другого розділу	41
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ФАН-САЙТОМ.....	43
3.1. Порядок встановлення та налаштування параметрів системи	43
3.2. СТРУКТУРА ІНТЕРФЕЙСУ	44
3.3. ТЕСТУВАННЯ РОБОТИ ПРОГРАМНОГО КОМПЛЕКСУ	55
Висновки до третього розділу	58
ВИСНОВКИ.....	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62
ДОДАТКИ.....	65

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CORS – Cross-Origin Resource Sharing

CRUD – ключові дії, які використовуються для керування даними: створення (Create), читання (Read), оновлення (Update) та видалення (Delete)

GIF – Graphics Interchange Format

JFIF – JPEG File Interchange Format

JSON – JavaScript Object Notation

MVC – Model-View-Controller (Модель–вигляд–контролер)

ODM – Object-Document Mapper (Об'єктно-Документний Мапер)

БД – База даних

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

					ПЗ.КР.Б-121-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми полягає в тому, що фан-сайти залишаються актуальними завдяки своєму внеску у створення віртуальних спільнот, розвитку творчості та формуванню інтернет-культури. Вони виконують роль платформи для виявлення та розвитку індивідуальних та колективних інтересів, створюючи простір для взаємодії та самовираження фанатів.

Говорячи про фан-сайти для К-рор товариств, то вони є важливими центрами для спільноти прихильників, де можна обмінюватися інформацією, враженнями та новинами. Вони дозволяють фанам виражати свою любов до музики, артистів та культури К-рор.

Крім того, фан-сайти надають платформу для творчої самовираження, де фанати можуть створювати та ділитися власним контентом, таким як фанарт, фотомонтажі, аналізи пісень і відео, танцювальні кавери та багато іншого. Це сприяє розвитку та поширенню креативного підходу до виявлення підтримки індустрії К-рор.

Фан-сайти також служать як зручний інструмент для споживачів контенту, надаючи їм змогу швидко знаходити актуальну інформацію, графіки виступів, анонси подій та інші важливі аспекти світу К-рор.

Сучасні технології дозволяють фан-сайтам бути інтерактивними та динамічними, пропонуючи різноманітні функції, такі як форуми, чати, стримінгові сервіси та мобільні додатки. Це робить взаємодію фанатів ще більш зручною та захоплюючою, дозволяючи їм завжди залишатися на зв'язку зі своєю улюбленою спільнотою.

Також варто зазначити, що фан-сайти сприяють міжкультурному обміну, адже вони об'єднують людей з різних куточків світу, які мають спільний інтерес до К-рор. Це допомагає розвивати толерантність та взаєморозуміння між представниками різних культур, сприяючи глобалізації музичної індустрії.

Загалом, фан-сайти для К-рор товариств виконують надзвичайно важливу функцію в сучасному цифровому світі, підтримуючи розвиток інтернет-культури,

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

творчості та спільнот фанатів, що робить їх невід'ємною частиною світової музичної екосистеми.

Метою даної роботи є розробка алгоритмів роботи та реалізація веб-сайту для фанатів К-поп спільноти.

Сформульована мета визначає **наступні завдання**:

- аналіз теоретичних та практичних вимог щодо розробки та впровадження фан-сайту;
- визначення структури та загальної архітектури системи;
- оцінка існуючих веб-сайтів з обраної тематики та визначення їх особливостей;
- обґрунтування і вибір інструментів для реалізації системи;
- розробка структурних компонентів та алгоритмів функціонування системи;
- реалізація системи фан-сайту.

Об'єктом дослідження є методи та засоби розробки веб-сайтів у визначених предметних областях.

Предметом дослідження є використання веб-орієнтованих технології для розробки системи фан-сайту.

У роботі було використано **методи** об'єктно-орієнтованого проектування, веб-програмування та дизайну.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ФАН-САЙТУ

1.1. Постановка задачі

Фан-сайт для будь-якої К-роп спільноти - це певна онлайн-платформа, створена з метою об'єднання шанувальників К-роп музики та артистів. Суть проекту полягає в створенні сприятливого середовища для спілкування, обміну інформацією та вираження своєї любові до К-роп культури [1].

Говорячи про такі сайти, то зазвичай там є наступне:

- Інформаційний ресурс для збору та публікації актуальної інформації про різних К-роп артистів, їхні новини, релізи, концерти та інші події.
- Форуми та спільноти - це місця для обговорення та обміну думками між фанатами. Розділи можуть бути присвячені окремим групам чи артистам, а також загальним темам.
- Мультимедійний контент для збору та публікації фотографій, відео, аудіозаписів, фан-арту та іншого контенту.
- Графік концертів та подій, тобто актуалізований графік майбутніх концертів, зустрічей з артистами, фан-зустрічей та інших подій, що можуть зацікавити шанувальників.
- Конкурси та певні активності. Відбувається організація різноманітних конкурсів, голосувань та інших активностей для спільноти з метою залучення і взаємодії фанатів.
- Особисті профілі, щоб користувачі мали можливість створювати власні профілі, додавати свої відгуки, створювати списки улюблених артистів та пісень, а також взаємодіяти з іншими учасниками спільноти [2].

Щодо технічних завдань для реалізації сайту, то вони наступні:

- ◆ Вибір технологій:
 - визначення стеку технологій для розробки веб-сайту;
 - вибір фреймворку для швидкої розробки та підтримки.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

- ◆ Розробка бази даних:
 - проектування та розробка структури бази даних для зберігання інформації про групи, виконавців, пісні, альбоми, користувачів і так далі;
 - вибір та налаштування системи управління базами даних.
 - ◆ Користувацький інтерфейс:
 - розробка користувацького інтерфейсу для сайту з використанням HTML, CSS та JavaScript.
 - реалізація адаптивного дизайну для оптимального відображення на різних пристроях та екранах.
 - ◆ Функціонал сайту:
 - створення та впровадження функцій та можливостей веб-сайту з метою забезпечення його функціональності та ефективності.
 - ◆ Управління контентом:
 - розробка панелі керування для адміністраторів сайту для додавання, редагування та видалення контенту, такого як інформація про групи К-поп, пісні, альбоми тощо.
 - ◆ Мультимедійний контент:
 - інтеграція системи завантаження та відображення фотографій та відео;
 - реалізація відтворення аудіо та відео прямо на сайті.
 - ◆ Тестування та відладка:
 - проведення тестування всіх функцій та компонентів сайту для виявлення та виправлення помилок.
 - ◆ Безпека:
 - використання захисту паролів;
 - налаштування резервного копіювання та захисту від втрати даних.
- Підсумовуючи все вище наведене, переді мною постають такі задачі:
- пошук та аналіз аналогів;
 - пошук та збір необхідних даних для створення та оформлення сайту;
 - створення БД та таблиць БД для зберігання даних;

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

- реалізація ролей для користувачів;
- налаштування адміністрування сайту;
- обрання програми та мови програмування для створення;
- визначення структури сайту та кількість модулів;
- тестування сайту.

1.2. Аналіз аналогів програмного продукту

Аналіз аналогів є важливим етапом в розробці будь-якого продукту або сервісу, також він може бути корисний при управлінні і вдосконаленні вже існуючих проектів. Говорячи про аналіз у цілому, то він може бути корисним для розуміння того, що є в цій галузі досліджуваного продукту та які можливості є для покращення програмного продукту. Важливим є те, що це допомагає виявити нові ідеї, функціональні можливості або підходи, які можна впровадити в продукт або сервіс для покращення його конкурентоспроможності та відповідності потребам користувачів.

1. Weverse [3]

Це офіційна платформа для спілкування фанів з різних країн зі своїми улюбленими зірками К-поп та розвагами. Створена компанією Big Hit Entertainment, яка є компанією багатьох відомих гуртів. Weverse надає можливість фанатам з усього світу спілкуватися, ділитися контентом та отримувати ексклюзивну інформацію про своїх улюблених артистів.

Серед основних функцій:

- Фанати можуть приєднатися до спільнот своїх улюблених артистів, де вони можуть обговорювати музику, відео, фотографії та інші аспекти життя та кар'єри своїх ідолів.
- Weverse надає віджети для спілкування та перегляду контенту безпосередньо на головному екрані смартфона користувача.
- Фанати можуть отримувати ексклюзивний контент, такий як фотографії, відеоролики та повідомлення від своїх улюблених артистів.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

- Weverse також має офіційний магазин, де фанати можуть придбати товари з атрибутикою своїх улюблених артистів.

- Компанія Big Hit Entertainment регулярно публікує оновлення, повідомлення та іншу офіційну інформацію про своїх артистів через платформу Weverse.

Переваги:

- Фанати мають можливість спілкуватися зі своїми улюбленими артистами безпосередньо через коментарі та повідомлення.

- Weverse надає фанатам доступ до ексклюзивного контенту, такого як фотографії, відео та повідомлення від артистів.

- Фанати з усього світу можуть приєднатися до спільнот своїх улюблених артистів, створюючи глобальну спільноту шанувальників.

- Weverse є офіційною платформою Big Hit Entertainment, тому інформація та оновлення, які надходять через цю платформу, є довіреною та достовірною.

- Платформа має офіційний магазин, де фанати можуть придбати товари з атрибутикою своїх улюблених артистів.

- Має сайт та можна скачати, як мобільний додаток.

Недоліки:

- Деякий контент на Weverse може бути доступний тільки після реєстрації, що обмежує доступ для тих, хто не бажає створювати акаунт.

- У деяких фанатів може виникнути проблема з обробкою великої кількості повідомлень та коментарів через активність на платформі.

- З популярністю Weverse може виникнути проблема з перевантаженням серверів та повільною роботою платформи в періоди активних дій.

- Для деяких дій потрібно оформлювати спонсорську підписку.

- Немає можливості проводити голосування та опитування.

Загальний вигляд сайту можна побачити на рисунках нижче (рис. 1.1) та (рис. 1.2).

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

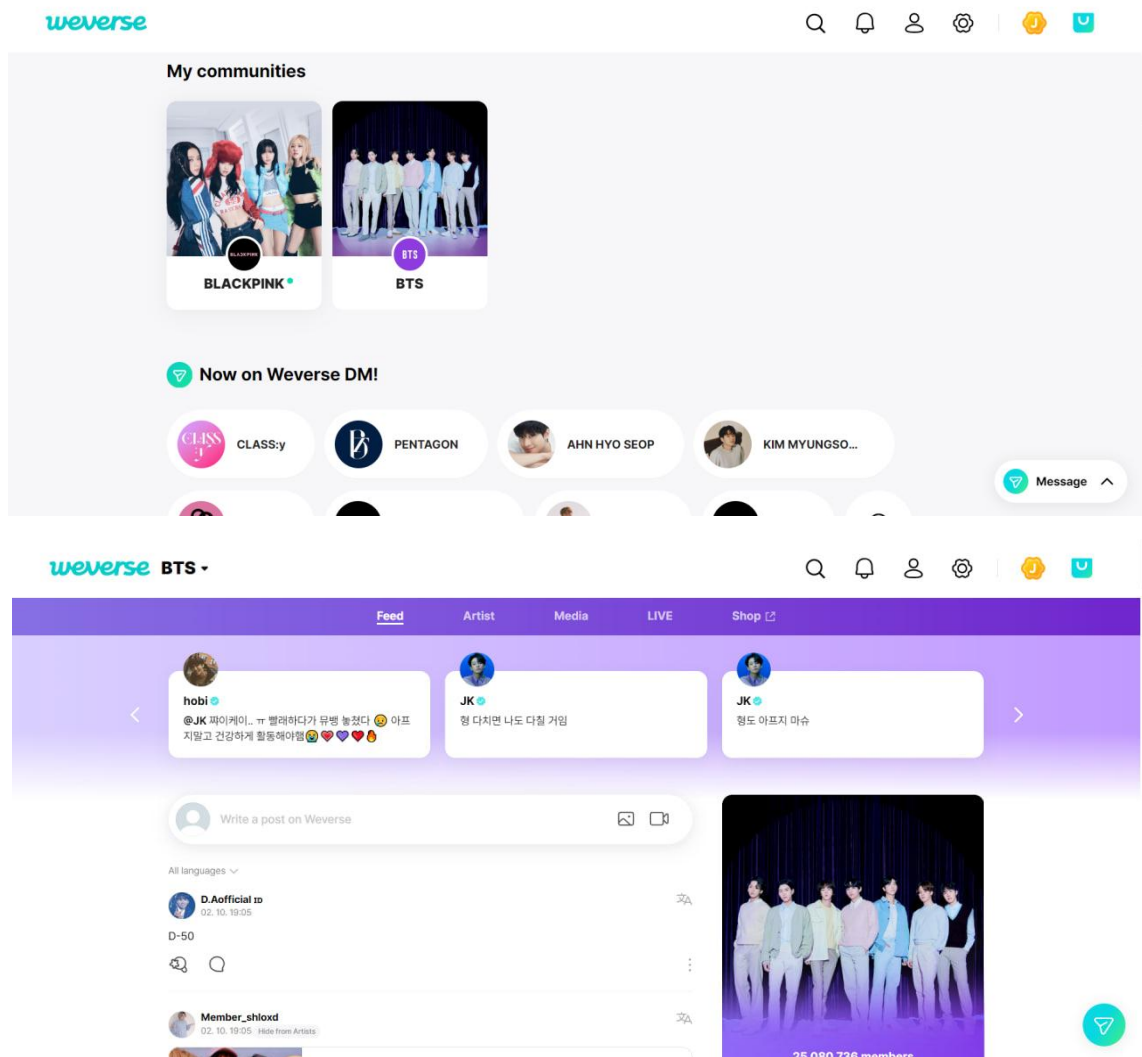


Рис. 1.1 - 1.2 - Вигляд сайту Weverse

2. FANS [4]

Це також офіційний сервіс, створений компанією JYP Entertainment, однією з провідних розважальних компаній Південної Кореї, для спілкування фанів з деякими К-поп зірками та публікації матеріалу.

Основні функції:

- Можливість надсилати повідомлення улюбленим виконавцям і отримувати лайки та коментарі.
- Можливість слідкувати за щоденним розкладом улюблених виконавців.
- Можливість перегляду дат днів народження улюблених артистів, ювілеїв, випусків нових альбомів, концертів і виступів на телебаченні.

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

- Можливість брати участь у різноманітних заходах, пов'язаних із різними виконавцями.

- Можливість голосувати та брати участь в опитуваннях.

Переваги:

- Фанам надається можливість зв'язку з їх улюбленими артистами безпосередньо через коментарі та повідомлення.

- Платформа FANS надає фанатам доступ до ексклюзивного контенту, такого як фотографії, відео та повідомлення від артистів.

- Прихильники з усього світу можуть приєднатися до спільнот своїх улюблених артистів, утворюючи глобальну спільноту шанувальників.

- FANS є офіційним сервісом JYP Entertainment, що гарантує надійність та достовірність інформації та оновлень, які надходять через нього.

Недоліки:

- Доступ до контенту доступний лише після реєстрації, що обмежує можливість перегляду для тих, хто не бажає створювати обліковий запис.

- Деякі фанати можуть стикатися з труднощами у керуванні великим потоком повідомлень та коментарів через інтенсивну активність на платформі.

- Існує можливість перевантаження серверів та повільної роботи платформи в періоди активності, що може призвести до тимчасових перешкод у доступі.

- Для багатьох дій потрібно оформлювати спонсорську підписку.

- Платформа немає офіційний магазин з атрибутикою артистів.

- Можна скачати, як мобільний додаток, але немає сайту, тобто не розрахований для користувачів комп'ютерів та ноутбуків.

Загальний вигляд додатку можна побачити на рисунках нижче (рис. 1.3), (рис. 1.4) та (рис. 1.5).

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

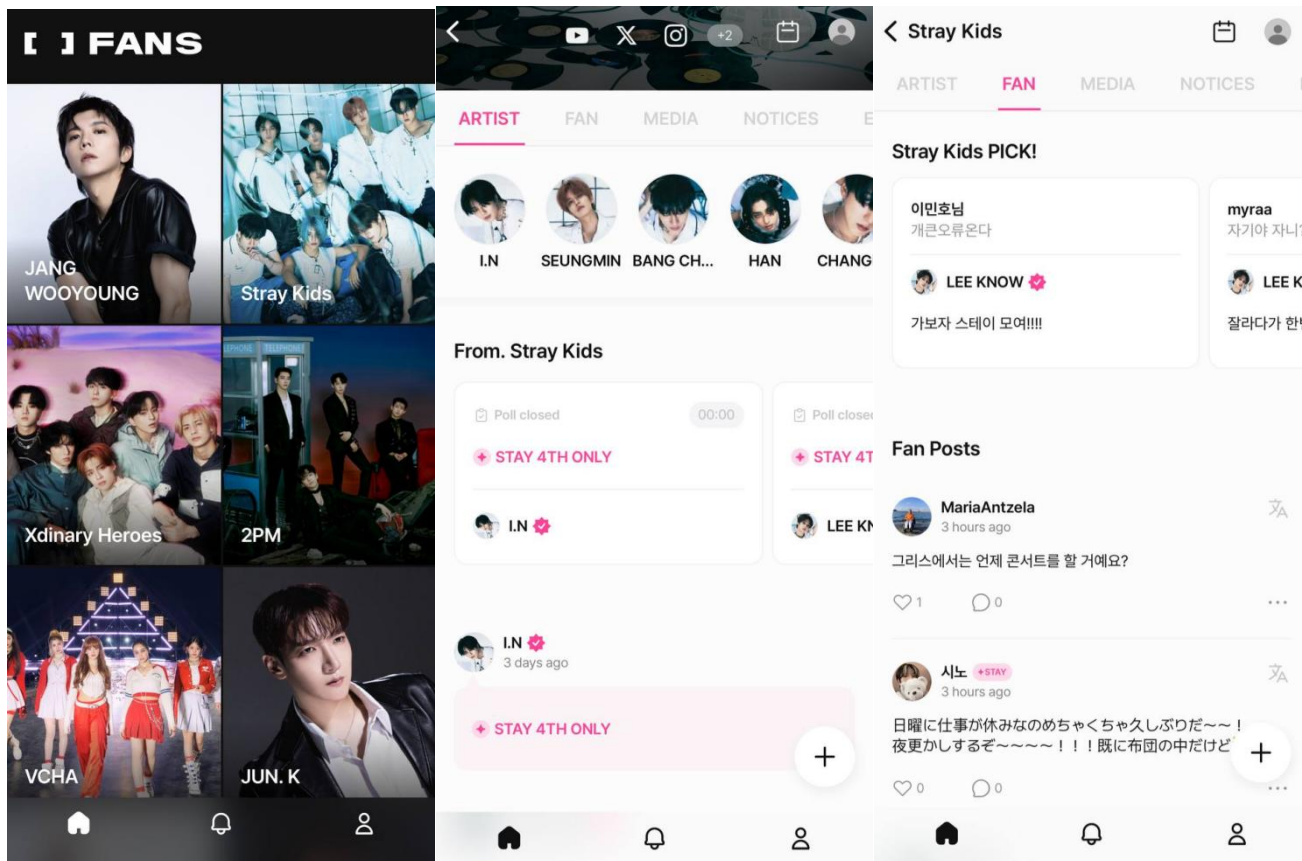


Рис. 1.3 - 1.5 - Вигляд додатку FANS

3. PODOAL [5]

Це неофіційний фан-додаток для спілкування між фанатами та публікації матеріалу.

Основні функції:

- Можливість спілкуватись з різними фанатами та слідкувати за ними.
- Можливість слідкувати за актуальними новинами.
- Можливість публікації різного контенту.
- Можливість брати участь у різних голосуваннях.

Переваги:

- Можливість зв'язку з іншими фанами улюблених артистів через коментарі та повідомлення.
- Є доступ до різного контенту: фотографії, відео та інше.
- Можливість отримувати актуальну інформацію.

Недоліки:

- Для перегляду контенту необхідно зареєструватися, що забороняє доступ для тих, хто не хоче створювати обліковий запис.
- Існує можливість перевантаження серверів та повільної роботи платформи під час активних періодів, що може призвести до тимчасових перешкод у доступі.
- Деякі функції доступні тільки за умови сплати спонсорської підписки.
- На платформі відсутній офіційний магазин із товаром артистів.
- Ця платформа не оптимізована для використання на комп'ютерах та ноутбуках.

Загальний вигляд додатку можна побачити на рисунках нижче (рис. 1.6) та (рис. 1.7).

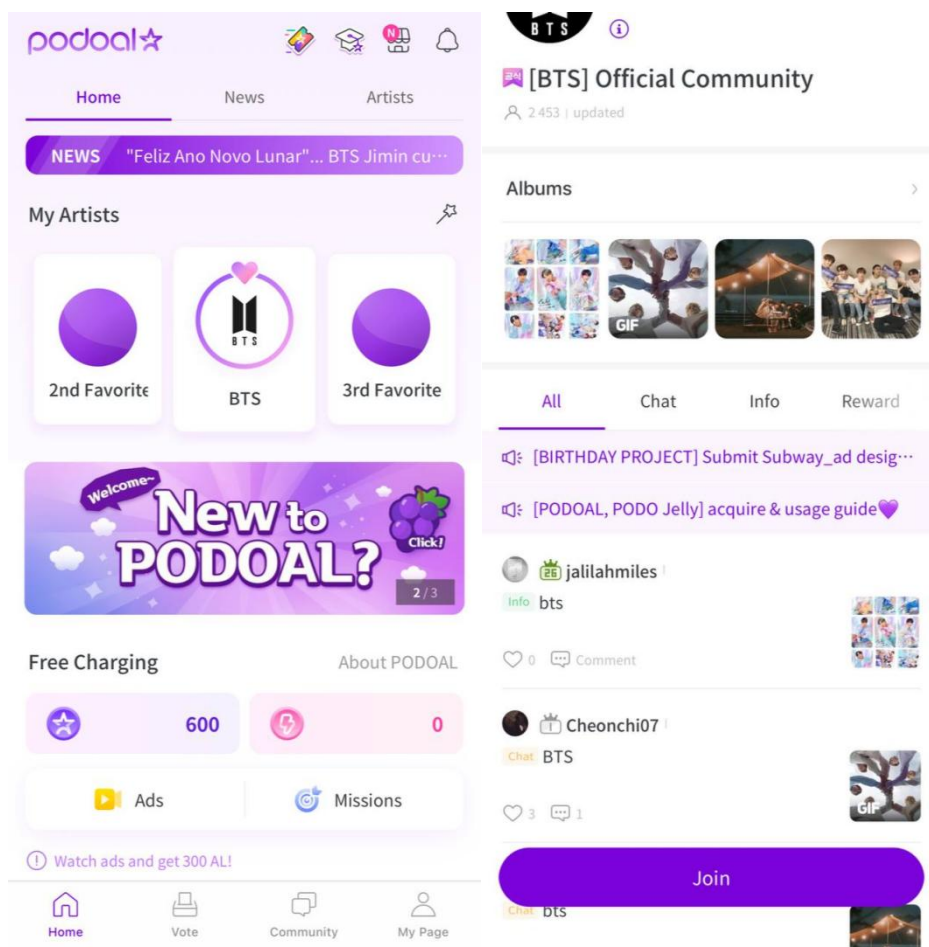


Рис. 1.6 - 1.7 - Вигляд додатку PODOAL

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

Порівняння аналогів

Назва	Weverse	FANS	PODOAL	Мій
Можливість спілкуватись	+	+	+	+
Публікація контенту	+	+	+	+
Актуальна інформація	+	+	+	+
Можливість опитувати/голосувати	-	+	+	-
Офіційний магазин	+	-	-	-
Доступ на всіх пристроях	+	-	-	+
Доступ для незареєстрованих	+-	-	-	+

1.3. Вибір архітектури веб-орієнтованої системи фан-сайту

Вибір архітектури веб-орієнтованої системи фан-сайту має значний вплив на всі аспекти розробки, експлуатації та майбутнього розвитку проекту.

Вірно обрана архітектура може полегшити роботу розробників і забезпечити оптимальну продуктивність. Деякі архітектурні підходи можуть підвищити швидкість відгуку сервера, знизити час завантаження сторінок або зменшити використання ресурсів сервера.

Крім того, правильна архітектура дозволяє системі масштабуватися, коли зростає обсяг даних або трафіку. Наприклад, мікросервісна архітектура може дозволити гнучко масштабувати окремі компоненти системи незалежно один від одного.

Також, деякі архітектурні рішення можуть покращити безпеку сайту, наприклад, розділення бізнес-логіки на мікросервіси може зменшити ризик витоку даних.

Більше того, гнучка архітектура дозволить легко вносити зміни в систему та додавати нові функціональні можливості без значних перешкод або рефакторингу.

Варто зазначити, що обрана архітектура також може вплинути на легкість підтримки системи в майбутньому та можливості її розвитку. Деякі архітектурні підходи можуть ускладнити додавання нових функцій або внесення змін у систему.

При цьому, не ефективно обрана архітектура може призвести до збільшення витрат на розробку, експлуатацію та підтримку системи в подальшому.

Вибір архітектури для веб-орієнтованої системи фан-сайту залежить від різних факторів, таких як масштаб проекту, вимоги до продуктивності, доступність технічних ресурсів і власність розробки [6].

Розглянемо деякі найбільш поширені архітектурні підходи:

- Монолітна архітектура - це коли всі компоненти програми розглядаються як єдине ціле, яке розгортається і працює разом. Це простий підхід, особливо для невеликих проектів або в ранніх стадіях розвитку, але може бути складним для масштабування і підтримки на великому масштабі [7].
- Клієнт-серверна архітектура - це розділення програми на клієнтську і серверну частини, де клієнтська частина відповідає за інтерфейс користувача, а серверна - за обробку даних.
- Мікросервісна архітектура - це розбиття програми на невеликі і незалежні сервіси, які взаємодіють між собою через API. Це дозволяє гнучко масштабувати та розвивати окремі частини системи [8].
- Шарова архітектура - розбиття програми на рівні (наприклад, презентаційний, бізнес-логіки, доступ до даних), кожен з яких відповідає за певні функціональні аспекти.

Найбільш використовуваними є монолітна архітектура та мікросервісна архітектура. Різниця між ними полягає в підходах до розробки, впровадження та масштабування програмного забезпечення.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

Ключовою відмінністю між ними є те, що у монолітній архітектурі усі компоненти застосунку розташовані в межах одного проекту. Весь функціонал, такий як бізнес-логіка, презентаційний шар, доступ до даних, інтегровані в одну програму, а у мікросервісній архітектурі весь функціонал розбивається на невеликі, самостійні та незалежні компоненти - сервіси (рис. 1.8). Кожен сервіс виконує певні обов'язки і взаємодіє з іншими через мережу, часто за допомогою API. Крім того, монолітна архітектура масштабується в цілому, тобто всі компоненти розгортуються та масштабуються разом, а мікросервісна архітектура дозволяє масштабувати окремі сервіси незалежно один від одного, що дозволяє оптимізувати ресурси та реагувати на потреби навантаження конкретних сервісів [9].

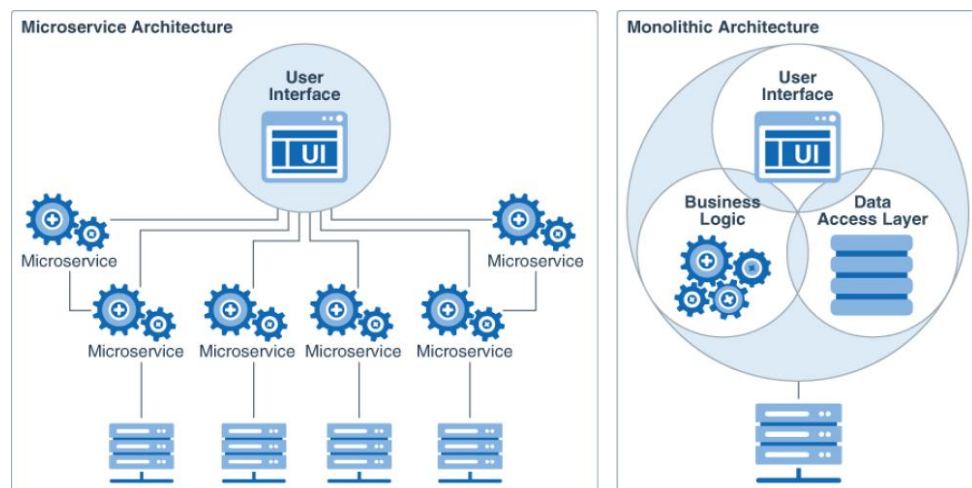


Рис. 1.8 - Різниця між мікросервісною та монолітною архітектурами

Зважаючи на все вище сказане, **серверну частину** я вирішила реалізувати з використанням монолітної архітектури. Це пов'язано з тим, що вона зазвичай простіша у розробці, оскільки всі компоненти знаходяться в одному проекті. Це дуже зручно для проекту, де немає необхідності у складних архітектурних рішеннях. Оскільки весь код знаходиться в одному місці, відлагодження буде простішим, ніж у мікросервісній архітектурі. Також тестування може бути легшим і ефективнішим, через те що не потрібно налаштовувати взаємодію між різними сервісами. Дуже важливою перевагою цієї архітектури є те, що розробка нового функціоналу або внесення змін буде швидшим, бо немає необхідності визначати взаємодію між різними сервісами.

Що стосується **архітектури веб-додатку**, вона є клієнт-серверною. Клієнтська частина, реалізована за допомогою React, відповідає за відображення інтерфейсу користувача, взаємодію з ним, та, в свою чергу, надсилає запити на сервер для отримання даних, тоді як серверна частина обробляє їх, використовуючи БД для збереження даних. Це дозволяє краще розділити обов'язки між різними компонентами системи, полегшує підтримку та розвиток.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При виборі інструментальних засобів для розробки сайту було обрано для клієнтської частини - React.JS, а для серверної - Node.JS. Вони є досить популярними технологіями у веб-розробці. Вибір цих технологій обґрунтований їхньою продуктивністю, масштабованістю, широким вибором інструментів та активною спільнотою розробників. Для реалізації бази даних було обрано MongoDB. Вона є відмінним вибором для багатьох проектів завдяки гнучкості, масштабованості, підтримці геопросторових запитів, інтеграції з сучасними технологіями та доступності як відкрите джерело.

Було обрано React, бо, по-перше, він базується на компонентній архітектурі, що дозволяє розбити користувацький інтерфейс на невеликі незалежні компоненти. Це полегшує управління складними інтерфейсами, поліпшує перевикористання коду та сприяє зручності розвитку та підтримки додатків. По-друге, React використовує віртуальний DOM для ефективного оновлення сторінок. Це дозволяє забезпечити швидку та плавну реакцію на зміни в даних, що покращує користувацький досвід. По-третє, React має велику та активну спільноту розробників, а також багато розширень, бібліотек та інструментів, що допомагають прискорити розробку та покращити функціональність додатків [10].

Node.js є гарним рішенням, бо це потужний інструмент для розробки серверних додатків, який відзначається швидкодією, асинхронним

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

програмуванням, легкістю вивчення та широкою підтримкою спільноти. Це популярна технологія для розробки серверних додатків на JavaScript [11].

MongoDB є перевіреним та надійним рішенням для зберігання та управління даними, завдяки своїй гнучкості та масштабованості. Його гнучка структура документів дозволяє зберігати дані у вигляді JSON-подібних об'єктів, що спрощує розробку та підтримку додатків. MongoDB також відомий своєю здатністю працювати з великими обсягами даних та високими навантаженнями завдяки горизонтальному масштабуванню. Крім того, він має вбудовану підтримку геопросторових запитів, що робить його ідеальним вибором для додатків, які потребують обробки географічних даних [12].

Мінімальні системні вимоги для серверу:

- 16 Гб пам'яті;
- 100 Гб вільного дискового простору;
- процесор з тактовою частотою не нижче 1GHz;
- доступ до Інтернету;
- операційна система Windows або Unix.

Висновки до першого розділу

У першому підрозділі з'ясували, що таке фан-сайт і які елементи на ньому можуть бути. Крім того, визначили технічні завдання для розробки сайту і основні задачі на дипломну роботу.

У другому підрозділі визначили для чого потрібно робити аналіз схожих продуктів. Також порівняли між собою три аналоги, з'ясували їх основний функціонал, переваги та недоліки кожного.

У третьому підрозділі було зроблено вибір архітектури для веб-системи. Спочатку, було проаналізовано які архітектури є, на основі цього, обрано необхідну для розробки. Таким чином, для розробки серверної частини буде використано монолітну архітектуру, а для архітектури веб-додатку, в цілому, - клієнт-серверну.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

У четвертому підрозділі обрано інструментальні засоби та визначено вимоги до апаратного забезпечення. Отже, для реалізації клієнтської частини буде використовуватись React.JS, для серверної - Node.JS, а для реалізації бази даних - MongoDB.

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ФАН-САЙТУ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Аналіз вимог передбачає визначення та деталізацію функціональних і нефункціональних вимог до програмного продукту, оцінку часових характеристик реакції системи на дії користувачів, специфікацію можливих сценаріїв використання системи, а також розробку узагальненої схеми елементів системи, включаючи їх структуру, взаємодію та методи з відповідним описом. Цей процес забезпечує чітке розуміння потреб користувачів та технічних характеристик, необхідних для створення ефективного і зручного програмного продукту [13].

Функціональні вимоги

Функціональні вимоги визначають конкретні функції, які система повинна виконувати [14]. Для фан-сайту, що надає вебінтерфейс для творчості та взаємодії користувачів, основні вимоги включають:

1. Реєстрація та аутентифікація користувачів:
 - Можливість реєстрації нових користувачів через email.
 - Вхід в систему за допомогою логіна і пароля.
2. Профілі користувачів:
 - Створення і редагування профілю користувача, включаючи додавання аватару та особистої інформації.
3. Пости та обговорення:
 - Створення нових постів і публікація.
 - Коментування постів.
4. Новини та фан-листи:
 - Публікація новин про К-поп.
 - Публікація фан-листів до свого улюбленого артиста або групи.
5. Галерея:
 - Завантаження і перегляд фото матеріалів.
6. Події та календар:

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

- Створення і перегляд подій, таких як концерти та фан-зустрічі.
 - Відслідковування сьогоденних подій.
7. Створення та керування групами по конкретній групі або артисту:
- Створення нових груп, присвячених конкретним К-рор виконавцям або гуртам.
 - Управління групами, включаючи публікацію постів, новин, оголошень, подій та іншого.
 - Можливість перегляду і участі у групах інших користувачів.

Нефункціональні вимоги

Нефункціональні вимоги стосуються характеристик системи, таких як продуктивність, безпека тощо:

1. Продуктивність:

Система повинна забезпечувати час відгуку не більше 2 секунд для основних операцій (перегляд сторінок, завантаження медіа). Витримування одночасного доступу до 1000 користувачів без деградації продуктивності.

2. Безпека:

Захист даних користувачів та хешування паролів для їх захисту. Регулярне оновлення програмного забезпечення для забезпечення безпеки.

3. Зручність використання:

Інтуїтивно зрозумілий інтерфейс з адаптивним дизайном для мобільних пристроїв [15].

Вимоги до часових характеристик реакції системи на дії користувача

Часові характеристики визначають прийнятний час відгуку системи на різні дії користувачів:

1. Реєстрація нового користувача: Час відгуку не більше 3 секунд.
2. Завантаження медіафайлів: Час відгуку для завантаження зображень не більше 5 секунд.
3. Перегляд постів, новин, фан-листів: Час відгуку не більше 2 секунд.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

Специфікація варіантів використання системи

Специфікація варіантів використання системи розкриває стандартні сценарії взаємодії користувачів із системою, щоб усвідомити, як саме користувачі можуть використовувати її для досягнення своїх мет. Ця специфікація тісно пов'язана з діаграмою варіантів використання (діаграмою прецедентів), яка графічно відображає ці сценарії. Діаграма варіантів використання слугує засобом візуалізації того, як користувачі можуть використовувати систему для своїх цілей, описаних у специфікації варіантів використання. Тому, для кращого уявлення того, як система буде використовуватися була створена діаграма (рис. 2.1).

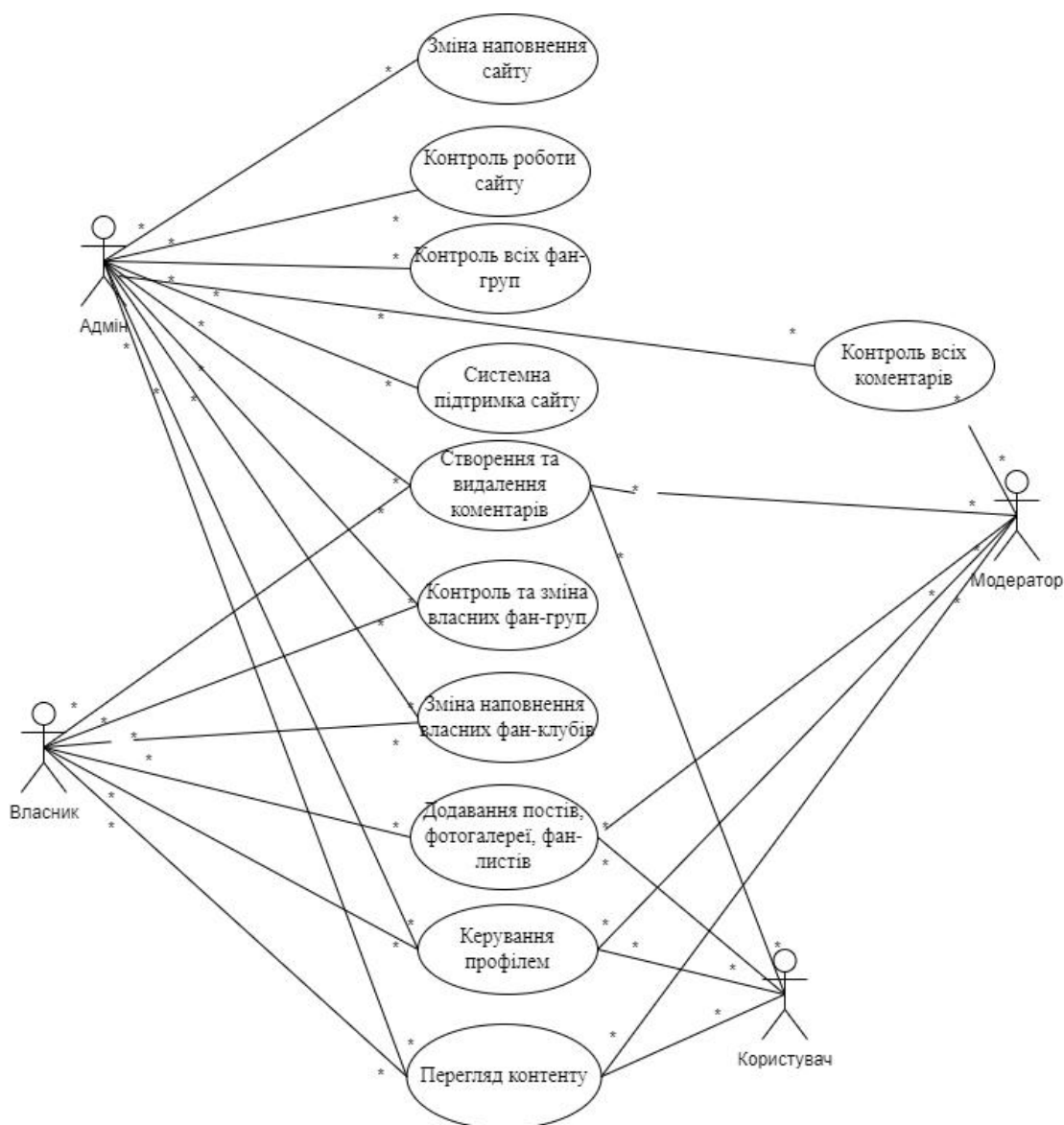


Рис. 2.1 - Діаграма варіантів використання

Змн.	Арк.	№ докум.	Підпис	Дата

ПЗ.КР.Б-121-24-ПЗ

Арк.

25

Нижче наведено основні сценарії, які будуть ілюструвати типові випадки використання фан-сайту (табл. 2.1).

Таблиця 2.1

Типові випадки використання

Актори	Найменування	Формулювання
Адмін	Зміна наповнення сайту	Цей варіант використання дозволяє адміністратору змінювати контент сайту, додавати нові блоки інформації.
Адмін	Контроль роботи сайту	Цей варіант використання дозволяє адміністратору контролювати дії, які відбуваються на сайті.
Адмін	Контроль всіх фан-групи	Цей варіант використання дозволяє адміністратору контролювати всі фан-групи на сайті та вносити зміни.
Адмін	Системна підтримка сайту	Цей варіант використання дозволяє адміністратору системно обслуговувати сайт та проводити технічні роботи.
Модератор, Адмін	Контроль всіх коментарів	Цей варіант використання дозволяє контролювати коментарі на сайті та групах.
Всі актори	Створення та видалення коментарів	Цей варіант використання дозволяє створювати і видаляти коментарі.
Власник, Адмін	Контроль та зміна власних фан-груп	Цей варіант використання дозволяє створювати та змінювати дані власних фан-груп.
Власник, Адмін	Зміна наповнення власних фан-груп	Цей варіант використання дозволяє розвивати створені ним фан-групи, за допомогою наповнення контентом (постами, подіями та іншим).
Всі актори	Додавання постів, фотогалереї, фан-листів	Цей варіант використання дозволяє додавати та керувати своїми постами, фото та фан-листами
Всі актори	Керування профілем	Цей варіант використання дозволяє створювати профіль, редагувати його.
Всі актори	Перегляд контенту	Цей варіант використання переглядати весь контент на сайті

Таким чином, діаграма варіантів використання становить важливу складову в процесі проектування системи. Вона допомагає краще розуміти потреби користувачів. Аналізуючи різні сценарії взаємодії між користувачами та системою, ця діаграма дозволяє ідентифікувати ключові функціональні можливості, які повинна мати система. Вона спрямовує увагу на реальні потреби та очікування користувачів, що дозволяє уникнути непорозумінь та забезпечити відповідність функціональності системи вимогам користувачів [16].

Об'єктно-орієнтована структура системи

Щоб зобразити структуру системи, використаю діаграму класів. Діаграма класів - це статична структурна діаграма, що використовується в об'єктно-орієнтованому програмуванні та моделюванні систем для візуального представлення класів, їх атрибутів, методів та взаємозв'язків між ними [17].

Проаналізувавши всі вимоги та потреби, можна виділити наступні класи, діаграма яких знаходиться в додатку А.

З діаграми можна виділити наступне:

- User - клас користувача, в якому зберігаються поля з інформацією про користувача, наприклад нікнейм, дата народження, роль та інші. Також у класі є деякі CRUD-методи для роботи з класом.

- Post, News, Gallery, ContactUs - класи постів, новин, галереї та зв'язку для всього сайті. У полях кожного з них зберігається потрібна для того класу інформація, наприклад ім'я, кількість переглядів та інші. У кожному класі є CRUD-методи для комфортної роботи та методи, які допомагають здійснювати пошук та фільтрацію.

- Group - клас групи, в якому є поля з назвою, посиланням на картинку та найголовніше, там зберігається ідентифікатор користувача, який створив групу. У класі є CRUD-методи для роботи з ним.

- GroupPost, GroupNews, GroupGallery - класи постів, новин, галереї для конкретної групи. Схоже з класами, які відповідають за весь сайт, але головною відмінністю є те, що ту є ідентифікатор групи до якої вони відносяться.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		27

- GroupInfo, GroupFanlist, Event - класи інформації, фан-листів та подій для конкретної групи. Кожен з класів містить свої особливі поля, наприклад GroupInfo містить поля з інформацією про групу, GroupFanlist - поля з назвою і текстом, а Event - назву, дату і певні деталі. Кожен з них має CRUD-методи та деякі допоміжні методи.

- Comment, CommentPostGroup - класи коментарів, які відрізняються тим, що другий клас відноситься тільки до коментарів тих постів, що є саме в групах. Мають схожі атрибути, наприклад текст і інформацію про того, хто створив, також в CommentPostGroup є додатково дані про групу в якій було створено. Мають схожі CRUD-методи.

2.2. Розробка бази даних системи

Зберігання всіх даних, які розміщуються на сайті, відбувається за допомогою MongoDB. Це СУБД, яка дозволяє зберігати складні дані у форматі JSON-подібних документів.

Всього в БД є 14 таблиць: Users, Posts, News, Galleries, ContactUs, Groups, GroupPosts, GroupNews, GroupGalleries, GroupInfos, GroupFanlists, Events, Comments, CommentPostGroups.

Нижче приводиться опис деяких з них (табл. 2.2), (табл. 2.3), (табл. 2.4), (табл. 2.5), (табл. 2.6), (табл. 2.7):

Таблиця 2.2

Опис таблиці Users

Назва поля	Тип	Опис
_id	ObjectId	Ідентифікатор
fullName	String	Ім'я
nickName	String	Нік
email	String	Пошта
passwordHash	String	Хешований пароль
birthday	Date	Дата народження
role	Int32	Роль користувача

Продовження таблиці 2.2

1	2	3
avatarUrl	String	Посилання на фото
createdAt	Date	Дата і час створення
updatedAt	Date	Дата і час оновлення
__v	Int32	Версія документа

Таблиця 2.3

Опис таблиці Posts

Назва поля	Тип	Опис
_id	ObjectId	Ідентифікатор
title	String	Заголовок
text	String	Текст
tags	Array	Теги
viewsCount	Int32	Кількість переглядів
userCreator	ObjectId	Ідентифікатор користувача
imageUrl	String	Посилання на фото
createdAt	Date	Дата і час створення
updatedAt	Date	Дата і час оновлення
__v	Int32	Версія документа

Таблиця 2.4

Опис таблиці Comments

Назва поля	Тип	Опис
_id	ObjectId	Ідентифікатор
text	String	Текст
userCreator	ObjectId	Ідентифікатор користувача
post	ObjectId	Ідентифікатор поста
createdAt	Date	Дата і час створення
updatedAt	Date	Дата і час оновлення
__v	Int32	Версія документа

Таблиця 2.5

Опис таблиці Groups

Назва поля	Тип	Опис
_id	ObjectId	Ідентифікатор
title	String	Назва групи
subscribers	Int32	Кількість підписників
userCreator	ObjectId	Ідентифікатор користувача
imageUrl	String	Посилання на фото
createdAt	Date	Дата і час створення
updatedAt	Date	Дата і час оновлення
__v	Int32	Версія документа

Таблиця 2.6

Опис таблиці GroupInfos

Назва поля	Тип	Опис
_id	ObjectId	Ідентифікатор
othertitle	String	Інша назва
genres	String	Жанри
debut	String	Дата дебюту
agency	String	Агенство
groupMembers	String	Учасники групи
formerMembers	String	Колишні учасники
fandomName	String	Ім'я фандому
colorFandom	String	Колір фандому
history	String	Історія групи
discography	String	Дискографія
userCreator	ObjectId	Ідентифікатор користувача
group	ObjectId	Ідентифікатор групи
imageGroupMembersUrl	String	Посилання на фото учасників
imageLogoFandomUrl	String	Посилання на фото логотипу
imageHistoryUrl	String	Посилання на фото історії
createdAt	Date	Дата і час створення
updatedAt	Date	Дата і час оновлення

1	2	3
__v	Int32	Версія документа

Таблиця 2.7

Опис таблиці Events

Назва поля	Тип	Опис
_id	ObjectId	Ідентифікатор
date	Date	Дата події
time	String	Час події
name	String	Назва
location	String	Місце проведення
type	String	Тип події
text	String	Опис події
userCreator	ObjectId	Ідентифікатор користувача
group	ObjectId	Ідентифікатор групи
createdAt	Date	Дата і час створення
updatedAt	Date	Дата і час оновлення
__v	Int32	Версія документа

Інші таблиці, які не були зазначені (News, Galleries, ContactUs, GroupPosts, GroupNews, GroupGalleries, GroupFanlists), вони схожі за полями на таблицю Posts, окрім таблиці CommentPostGroups, бо вона має більш спільне з таблицею Comments.

Кожна таблиця є пов'язаною з таблицею User, за полем userCreator, це пов'язано з тим, що кожен елемент створюється користувачем. Також, таблиці Comments та CommentPostGroups зв'язані відповідно з Posts та GroupPosts, за допомогою post, таким чином, ми зберігаємо інформацію про те з яким саме постом має зв'язок коментар. Таблиці, які пов'язані з групою, мають відповідне поле group, за допомогою якого ми і визначаємо конкретну групу до якої відноситься елемент.

					ІПЗ.КР.Б-121-24-ІЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3. Проектування та реалізація алгоритмів роботи системи

Сайт має доволі зручний та зрозумілий функціонал, також він є доволі різноманітним для різних груп користувачів. Діаграма активності — це тип діаграми, яка використовується для моделювання динамічних аспектів системи. Вона показує потік управління та обробку даних у системі, відображаючи послідовність дій або станів, через які проходить система або процес [18]. Детальний алгоритм роботи системи з урахуванням ролей, діаграма яких знаходиться в додатку Б.

При вході в систему адміністратора, він може:

- створювати, редагувати, видаляти новини;
- перевіряти та видаляти повідомлення зі Зв'язку;
- переглядати статистику;
- керувати наповненнями сайту;
- керувати групами.

Власник може:

- створювати, редагувати, видаляти групу;
- у групі може створювати, редагувати, видаляти пости, новини, галерею, інформацію, події;
- переглядати статистику;
- керувати фан-листами;
- керувати групою в цілому.

Модератор:

- керує та видаляє коментарі під постами на сайті;
- також керує та видаляє коментарі під постами в групі.

Користувач може:

- переглядати всю інформацію на сайті;
- створювати, редагувати, видаляти пости, галерею;
- у групі може додавати, редагувати та видаляти фан-листи;

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

- писати коментарі під будь-яким постом;
- керувати власним профілем.

2.4. Реалізація додатку фан-сайту

Важливою частиною сайту є групи, їх створення та управління ними. Тому далі розглядемо реалізацію цього блоку.

Спочатку, була створена модель для груп, де у нас створюються та задаються поля (рис. 2.2).

```

3  const GroupSchema = new mongoose.Schema({
4    title:{
5      type:String,
6      required:true,
7    },
8    subscribers:{
9      type:Number,
10     default: 0,
11   },
12   userCreator: {
13     type: mongoose.Schema.Types.ObjectId,
14     ref: "User",
15     required:true,
16   },
17   imageUrl: String,
18 }, {
19   timestamps: true,
20 })

```

Рис. 2.2 - Схема GroupSchema у моделі групи

У цій моделі визначається схема GroupSchema для колекції "groups" у базі даних MongoDB за допомогою Mongoose. Mongoose - це ODM для MongoDB і Node.js. Він забезпечує зручний та інтуїтивно зрозумілий спосіб взаємодії з базою даних MongoDB, дозволяючи працювати з документами як з об'єктами в JavaScript [19].

Отже, ця схема визначає структуру документів у колекції, їхні поля та властивості. Тут є:

1. title (обов'язковий рядок) - назва групи.
2. subscribers (число, за замовчуванням 0) - кількість підписників.

3. userCreator (обов'язковий ObjectId, що посилається на колекцію User) - ідентифікатор користувача, який створив групу.
4. imageUrl (рядок) - URL-адреса зображення групи.

Крім того, завдяки параметру timestamps, у кожному документі будуть автоматично створені поля createdAt і updatedAt, що дозволяє відстежувати час створення та оновлення документа [20].

Далі в контролері групи є функція create (рис. 2.3).

```

3  const create = async (req,res) =>{
4      try{
5          const doc = new GroupModel({
6              title: req.body.title,
7              imageUrl: req.body.imageUrl,
8              userCreator: req.userId
9          })
10
11         const post = await doc.save()
12
13         res.json(post)
14     }
15     catch(err){
16         console.log(err)
17         res.status(500).json({
18             message: "Не вдалось оновити групу"
19         })
20     }
21 }

```

Рис. 2.3 - Функція create у контролері групи

Вона є асинхронним обробником запиту для створення нового документа у колекції груп. Функція приймає HTTP-запит (req) та HTTP-відповідь (res), і використовує модель GroupModel для збереження нової групи в базі даних.

У самій функції створюється новий екземпляр GroupModel (документ) з полями, що витягуються з тіла запиту (req.body) і з ідентифікатором користувача (req.userId).

За допомогою методу save, відбувається збереження документа в базі даних і якщо документ успішно збережений, то сервер відправляє клієнту JSON-відповідь з даними створеної групи. Якщо виникає помилка під час виконання блоку try, то ця помилка буде перехоплена в блоці catch, далі вона виводиться в консоль для діагностики. Також, відправляється відповідь з кодом статусу 500 (Internal Server Error).

Далі, в index.js, за допомогою модуля express будуть відбуватись запити. Express - це веб-фреймворк для створення веб-додатків на мові JavaScript з використанням середовища виконання Node.js. Він дозволяє швидко створювати масштабовані та ефективні серверні додатки. Express надає зручний і простий у використанні інтерфейс для обробки HTTP-запитів, включаючи маршрутизацію, обробку запитів та відповідей, middleware і багато іншого [21].

Запити, які стосуються груп (рис. 2.4):

```
76 app.get('/groups', GroupController.getAll)
77 app.get('/mygroups/:id', GroupController.getAllMy)
78 app.get('/groups/search', GroupController.searchGroupByTitle)
79 app.get('/groups/:id', GroupController.getOne)
80 app.post('/groups', checkAuth, groupCreateValidation, handleValidationErrors, GroupController.create)
81 app.patch('/groups/:id', checkAuth, groupCreateValidation, handleValidationErrors, GroupController.update)
82 app.delete('/groups/:id', checkAuth, GroupController.remove)
```

Рис. 2.4 - Запити, які стосуються груп

Тут є запити на виведення, пошук, створення, оновлення та видалення. Але зараз цікавить саме запит на створення, тому розберемо детальніше.

1. app.post('/groups', ...) - визначає, що обробник буде викликатися тільки при POST-запитах на шлях '/groups'. А саме, коли клієнтська сторона надсилає POST-запит на сервер з адресою '/groups', відбувається активація цього маршруту.

Для деяких перевірок далі використовуються middleware. Middleware - це функції, які виконуються перед тим, як запит до сервера досягне фактичного маршруту. Вони можуть виконувати різні завдання, такі як обробка даних запиту, перевірка аутентифікації користувача, валідація даних, обробка помилок, збереження даних у кеші тощо [22].

Розбираємо всі, які йдуть в цьому запиті:

2. checkAuth - middleware, який перевіряє автентифікацію користувача перед тим, як дозволити продовження обробки запиту. Це функція, яка перевіряє наявність токена аутентифікації в заголовках запиту та відповідність цього токена інформації про користувача в базі даних.

3. `groupCreateValidation` - middleware, який виконує валідацію даних, надісланих у POST-запиті, для створення групи. Він перевіряє, чи дані відповідають очікуваному формату та вимогам перед їх обробкою.

4. `handleValidationErrors` - middleware, який обробляє будь-які помилки валідації, які виникли в результаті `groupCreateValidation`. Він повертає користувачеві повідомлення про помилки і відправляє відповідь зі статусом помилки.

Коли користувач проходить всі перевірки, то далі він переходить до останнього етапу:

5. `GroupController.create` - обробник, який відповідає за обробку POST-запитів на створення групи. Він викликає раніше згаданий метод `create` в контролері `GroupController`, який створює та зберігає дані у БД.

Таким чином, на серверній частині, тобто на частині бекенду, реалізовано можливість створення груп, далі у частині клієнта, тобто фронтенду, відбувається продовження цього процесу.

Однак, перед тим як перейти до опису фронтед частини, варто обговорити один важливий момент. Серверна та клієнтська частини знаходяться на різних локальних хостах, для того щоб без проблем ці частини могли взаємодіяти між собою, я використала модуль `cors`.

Модуль `cors` використовується в середовищі Node.js для обробки CORS запитів. CORS - це механізм безпеки, який дозволяє веб-сайтам запитувати ресурси з інших доменів. Коли клієнтська сторона намагається зробити запит на сервер із іншої доменної адреси, браузер перевіряє, чи дозволено такий запит на сервері. Модуль `cors` додає необхідні заголовки до відповіді сервера, щоб браузер зробив цей запит. Так відбуваються запити для різних двох частин [23].

Повертаючись до фронтед частини, при заповненні форми групи викликається функція `onSubmit` (рис. 2.5).

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		36

```

51 const onSubmit = async () =>{
52   try {
53     setLoading(true)
54     const fields = {
55       title,
56       imageUrl,
57     }
58     const {data} = isEditing ? await axios.patch(`/groups/${id}`, fields) : await axios.post("/groups", fields)
59
60     const _id = isEditing ? id : data._id
61
62     navigate(`/groups/${_id}`)
63   } catch (error) {
64     console.warn(error)
65     console.log(error)
66     alert("Помилка при створені групи")
67   }
68 }

```

Рис. 2.5 - Функція onSubmit

Ця функція встановлює значення loading як true, збирає дані з форми, ініціює POST-запит до сервера для створення нової групи або PATCH-запит для редагування існуючої групи. Після успішного виконання запиту, користувач переадресовується на сторінку нової/редагованої групи. У випадку помилки виводиться повідомлення про помилку. Таким чином, відбувається створення або редагування групи у клієнтській частині програми.

Після створення групи для власника відкривається можливість додавання певних блоків у свою групу, для прикладу розберемо блок галереї.

Як і в групі, так для галереї є окрема модель. Вона створює схему, яка визначає структуру з наступними полями:

1. title (рядок, обов'язковий) - назва фото.
2. imageUrl (рядок, обов'язковий) - посилання на фото.
3. viewsCount (число, за замовчуванням 0) - кількість переглядів.
4. userCreator (посилання на об'єкт User, обов'язкове) - id того, хто створив фото.
5. group: (посилання на об'єкт Group, обов'язкове) - id групи в якій знаходиться фото.

Також схема автоматично додає часові мітки createdAt та updatedAt.

Все це зображено нижче (рис. 2.6).

```

3  const GroupGallerySchema = new mongoose.Schema({
4    title: {
5      type: String,
6      required: true,
7    },
8    imageUrl: {
9      type: String,
10     required: true,
11   },
12   viewsCount: {
13     type: Number,
14     default: 0,
15   },
16   userCreator: {
17     type: mongoose.Schema.Types.ObjectId,
18     ref: "User",
19     required: true,
20   },
21   group: {
22     type: mongoose.Schema.Types.ObjectId,
23     ref: "Group",
24     required: true,
25   },
26 }, {
27   timestamps: true,
28 })

```

Рис. 2.6 - схема GroupGallerySchema у моделі галереї групи

У контролері галереї є так само функція для створення фото, в цілому вона схожа за запитом до БД на створення групи. Далі, в загальному index.js, є POST-запит на створення, який так само перевіряє чи користувач увійшов в систему та валідує отриману інформацію.

Варто зазначити, що процес додавання фото потребує додаткової обробки. Розберемо цей момент детальніше (рис. 2.7).

```

16  //сховище для картинок
17  const storage = multer.diskStorage({
18    destination: (req, file, cb) => {
19      cb(null, "uploads")
20    },
21    filename: (req, file, cb) => {
22      cb(null, file.originalname)
23    }
24  })
25
26  const upload = multer ({storage})
27
28  app.use(express.json())
29  app.use(cors())
30  app.use("/uploads", express.static("uploads"))

```

Рис. 2.7 - Використання multer

З рисунку видно, що для обробки завантаження файлів використовується Multer. Це middleware для Node.js, яке використовується для обробки multipart/form-data, головним чином для завантаження файлів. Вона допомагає легко приймати файли, завантажені через форми HTML, і зберігати їх на сервері або в пам'яті [24].

Спочатку відбувається налаштування multer. Ми вказуємо:

1. destination (вказує, що файли зберігатимуться в директорії “uploads”).
2. filename(задає оригінальне ім'я файлу як ім'я файлу при збереженні).

Потім створюємо змінну upload, яка у нас middleware з налаштованим сховищем. Далі, робимо директорію “uploads” доступною для клієнтів через шлях “/uploads”. Це дозволить завантажувати файли на сервер і робити їх доступними для завантаження за вказаним шляхом. Трішки нижче визначаємо маршрут для обробки POST-запитів на шлях “/upload”(рис. 2.8).

```

42 //загрузка картинки
43 app.post('/upload', checkAuth, upload.single("image"),(req, res) =>{
44     res.json({
45         url:`/uploads/${req.file.originalname}`
46     })
47 })

```

Рис. 2.8 - POST-запит на шлях “/upload”

Цей маршрут обробляє завантаження одного файлу з автентифікацією та повертає клієнту URL завантаженого файлу.

На фронтенді завантаження фото відбувається наступним чином (рис. 2.9).

```

34 const handleChangeFile = async (event) => {
35     try {
36         const formData = new FormData()
37         const fileName = event.target.files[0]
38         formData.append("image", fileName)
39         const {data} = await axios.post("/upload",formData)
40         setImageUrl(data.url)
41     } catch (error) {
42         console.warn(error)
43         alert("Помилка при загрузці фото")
44     }
45 };

```

Рис. 2.9 - Функція завантаження

Функція `handleChangeFile` створює `FormData` з вибраним файлом, відправляє його на сервер за допомогою `axios.post`, встановлює URL завантаженого зображення у стані, та обробляє можливі помилки, виводячи їх у консоль і показуючи повідомлення користувачу.

Варто відмітити, що у фронтенд частині, я доволі часто працюю з модулем `axios`. Цей модуль - це бібліотека для HTTP-запитів, яка працює в браузері та `Node.js`. Вона дозволяє робити асинхронні запити до серверів, отримувати та відправляти дані, а також обробляти відповіді сервера. `Axios` підтримує `promises` (дозволяє працювати з асинхронним кодом), що робить роботу з асинхронними запитами більш зручною та читабельною. Тобто, за допомогою `axios`, я можу робити запити з фронтенд частини на бекенд [25].

Також, на клієнті для завантаження даних та управління їх станами, використовую `redux`. Це бібліотека управління станом для додатків `JavaScript`, особливо популярна в додатках `React`. Вона допомагає управляти станом додатку централізовано, що полегшує розробку складних додатків, особливо тих, що мають багато взаємопов'язаних компонентів.

Основна концепція `Redux` - це управління станом додатку через один “store”. У `Redux` стан додатку зберігається в одному місці - у `store`. Компоненти можуть читати стан з цього `store` і оновлювати його, висилаючи “дії” (actions), які описують, які зміни необхідно зробити у стані. Ці дії обробляються “редукторами” (reducers), які змінюють стан відповідно до вказаних дій [26].

Розглянемо приклад на завантаження всіх груп (рис. 2.10)

```
4 export const fetchGroups = createAsyncThunk("groups/fetchGroups", async()=>{
5   const {data} = await axios.get("/groups")
6   return data
7 })
```

Рис. 2.10 - Функція `createAsyncThunk`

Ця функція створює `thunk` (тобто функцію, яка може виконувати асинхронні дії) для отримання груп. При виклику `fetchGroups`, він виконує запит до сервера за допомогою `Axios`, отримує дані груп і повертає їх. Потім йде наступне (рис. 2.11):

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

```

18  const initialState = {
19      groups: {
20          items: [],
21          status: "loading",
22      },
23  }
24
25  const groupsSlice = createSlice({
26      name: "groups",
27      initialState,
28      reducers: {},
29      extraReducers: {
30          [fetchGroups.pending]: (state) => {
31              state.groups.items = []
32              state.groups.status = "loading"
33          },
34          [fetchGroups.fulfilled]: (state, action) => {
35              state.groups.items = action.payload
36              state.groups.status = "loaded"
37          },
38          [fetchGroups.rejected]: (state) => {
39              state.groups.items = []
40              state.groups.status = "error"
41          },
42      },
43  })

```

Рис. 2.11 - Задавання стану

Спочатку налаштовуємо початковий стан додатку, включаючи порожній список груп та статус "завантаження". Використовуючи Redux Toolkit, створюється "slice" стану для груп. За допомогою extraReducers, визначаються реакції на асинхронні дії: якщо дані ще завантажуються, то стан оновлюється на "завантаження", якщо запит успішно виконаний, то стан оновлюється з отриманими даними груп та статусом "завантажено", у випадку відхилення запиту - вказується статус "помилка".

Всі повні фрагменти кодів, які наводились вище, зберігаються в додатку В.

Висновки до другого розділу

У другому розділі проведено детальне проектування фан-сайту. У першому підрозділі, аналізувалися вимоги до програмного продукту, включаючи як функціональні, так і нефункціональні вимоги різних типів користувачів. Були визначені часові характеристики реакції системи та варіанти її використання.

У другому підрозділі розроблено схему бази даних, яка включає в себе опис структури збереження даних.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

У третьому підрозділі наведено узагальнені алгоритми роботи системи та описано взаємодію її елементів під час роботи програмного комплексу.

У четвертому параграфі детально розглянуто реалізацію програмного комплексу. Наведено фрагменти складного коду, які потребують пояснення, з посиланням на додатки, де міститься відповідний код. Це включає у себе опис методів та структур системи, які були розроблені на основі виявлених вимог.

Цей розділ включає в себе детальний огляд проектування, реалізації та алгоритмів роботи системи, що визначається як ключовий етап у розробці програмного забезпечення.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ФАН-САЙТОМ

3.1. Порядок встановлення та налаштування параметрів системи

Для того щоб система коректно працювала її потрібно правильно встановити та налаштувати, а для цього необхідно розгорнути кожен елемент та налаштувати їх. Подальший опис архітектури візуалізований нижче (рис. 3.1).

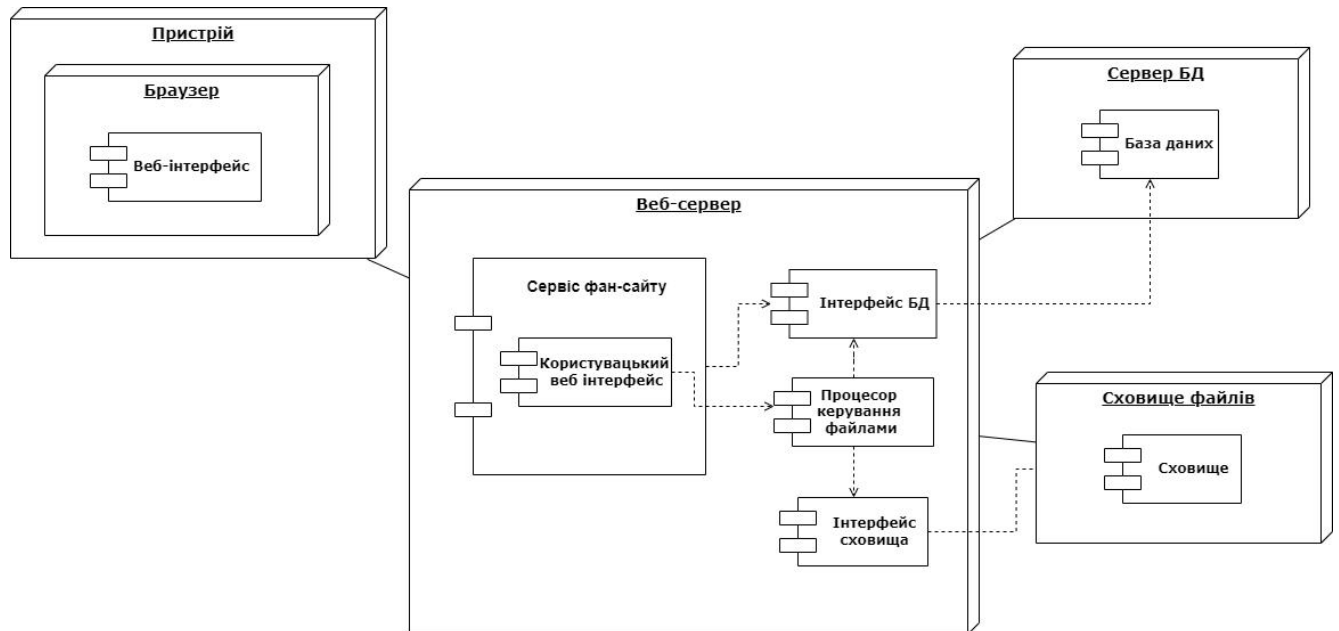


Рис. 3.1 - Діаграма розгортання

Дл опису фізичної архітектури системи та візуалізації розподілу програмного забезпечення на апаратних ресурсах було створено діаграму розгортання. Вона показує, як компоненти програмного забезпечення розподіляються на фізичних машинах, серверах або інших пристроях. Це дозволяє зрозуміти, як система працює на апаратному рівні [27].

Ця діаграма демонструє:

1. Пристрій (Клієнт)

- Браузер: Використовується для доступу до системи через веб-інтерфейс.
- Веб-інтерфейс: Інтерфейс для користувача, через який він взаємодіє з системою.

2. Веб-сервер

- Сервіс фан-сайту: Основний сервіс, що обробляє запити користувачів.

- Користувацький веб інтерфейс: Компонент, який забезпечує взаємодію з користувачем через веб-браузер.
- Інтерфейс БД: Компонент, що забезпечує зв'язок з базою даних на сервері БД.
- Процесор керування файлами: Компонент, що обробляє файли, можливо, забезпечує завантаження, зберігання і керування файлами.
- Інтерфейс сховища: Компонент, що забезпечує зв'язок зі сховищем файлів.

3. Сервер БД (База даних)

- База даних: Зберігає дані, необхідні для роботи системи.

4. Сховище файлів

- Сховище: Зберігає файли, які обробляє або зберігає система.

Користувач за допомогою браузера взаємодіє з веб-інтерфейсом, який знаходиться на веб-сервері. Веб-інтерфейс на веб-сервері передає запити користувача до сервісу фан-сайту. Сервіс фан-сайту обробляє запити користувачів, використовуючи інтерфейс БД для взаємодії з базою даних на сервері БД та процесорі керування файлами для обробки файлів. Процесор керування файлами використовує інтерфейс сховища для доступу до сховища файлів. Інтерфейс БД забезпечує зв'язок та обмін даними між веб-сервером і сервером бази даних. Інтерфейс сховища забезпечує зв'язок та обмін даними між веб-сервером і сховищем файлів.

Ця діаграма допомагає зрозуміти, як розподілені основні функції системи між різними компонентами та як ці компоненти взаємодіють один з одним.

3.2. Структура інтерфейсу

Розберемо детальніше як виглядає, як побудований і як працювати зі сайтом в цілому. На сайт може зайти будь-який користувач і без проблем переглядати весь вміст сайту. При заході на головну сторінку його чекатиме наступне (рис. 3.2).

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

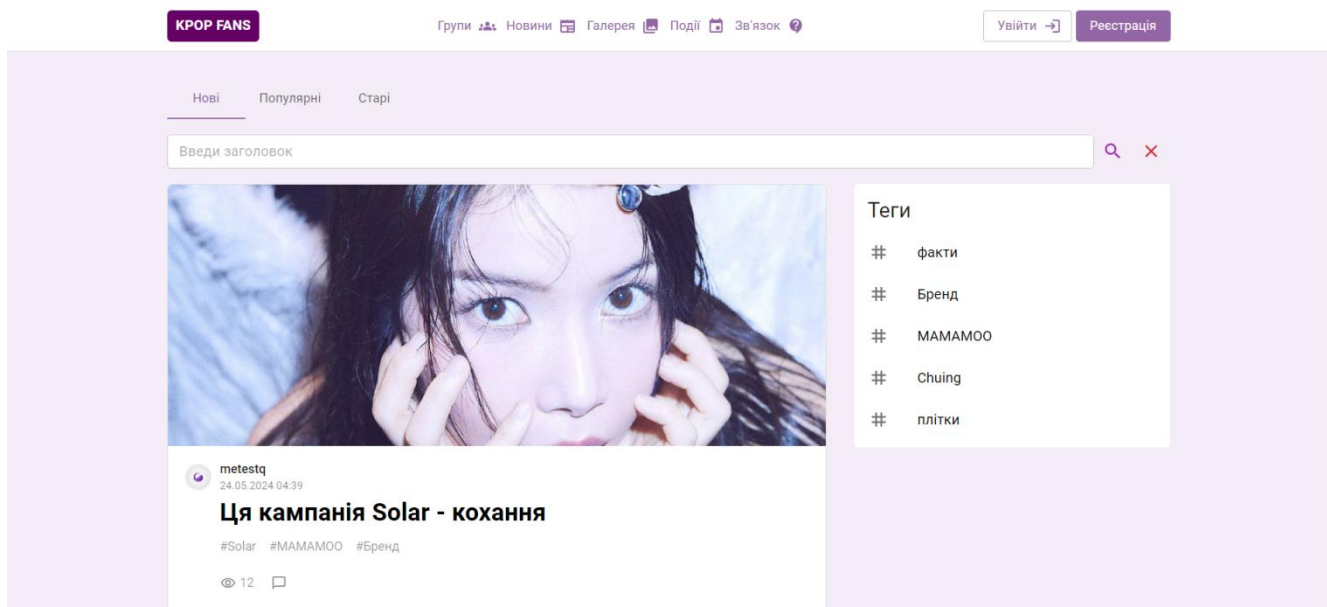


Рис. 3.2 - Головна сторінка сайту

На головній сторінці знаходяться всі пости, які можна фільтрувати за часом створення, тобто нові чи старі, або за популярністю поста, точніше за кількістю переглядів. Також, користувач може шукати пости. Це виглядатиме наступним чином (рис. 3.3).

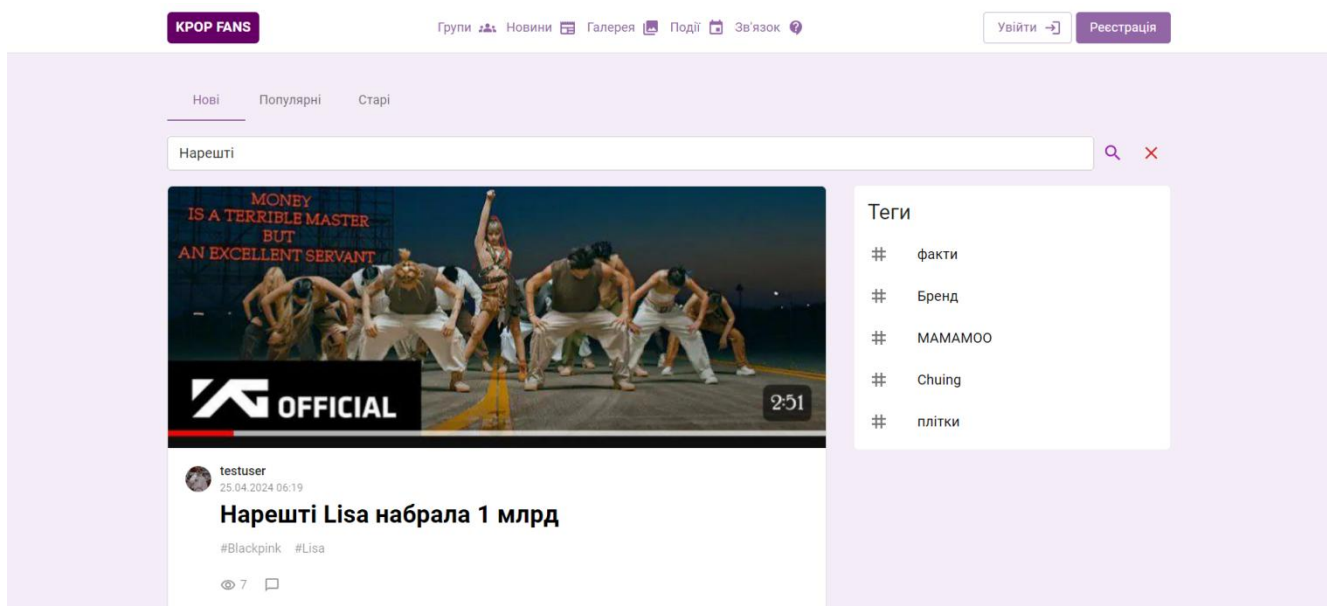


Рис. 3.3 - Пошук за назвою поста

У правій частині екрану знаходиться перелік 5 тегів, які користувачі використовують при створенні поста. Вони виводяться рандомним чином.

Щоб розширити свої можливості на сайті, необхідно або увійти в свій акаунт (рис. 3.4), якщо він є, або зареєструвати новий (рис. 3.5).

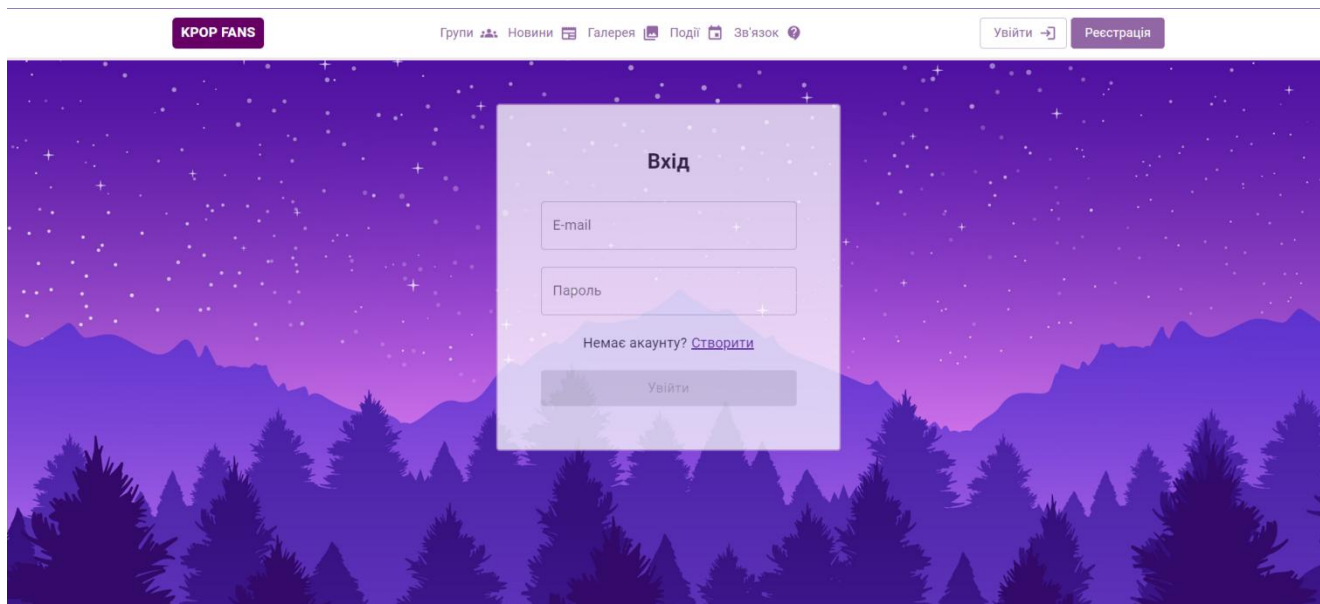


Рис. 3.4 - Сторінка Входу

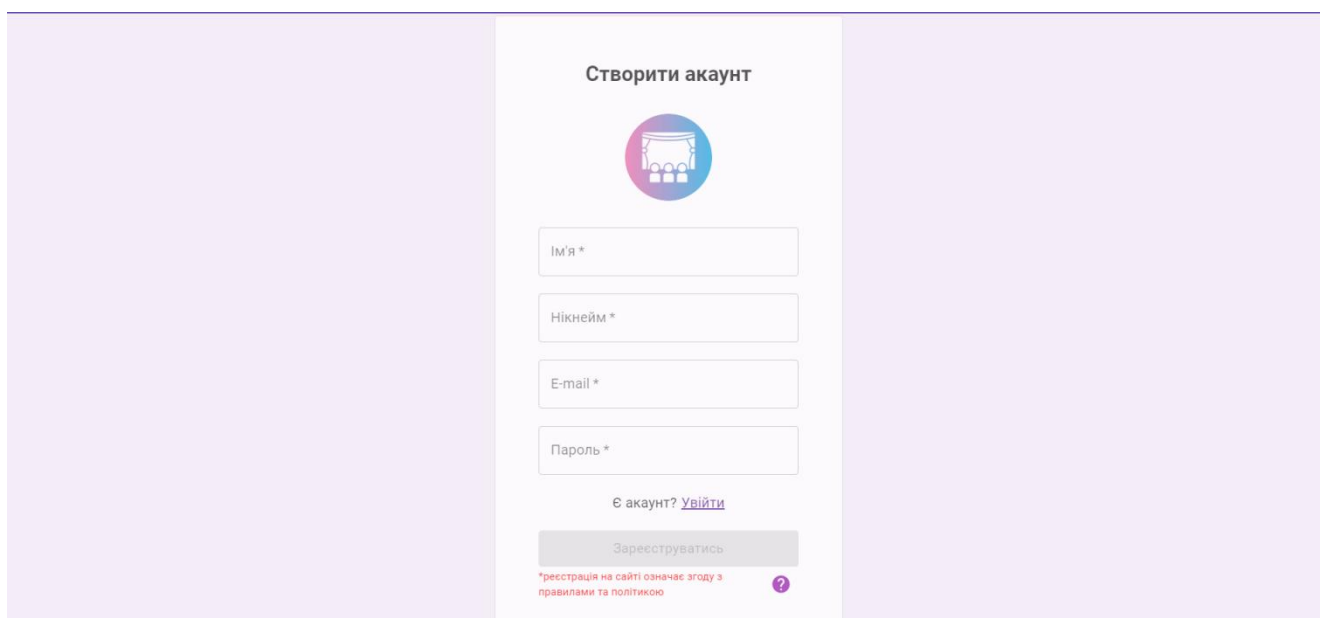


Рис. 3.5 - Сторінка Реєстрації

Для того щоб увійти, то необхідно ввести свою пошту та пароль, щоб створити акаунт, то потрібно заповнити поля імені, придумати нік собі, ввести власну пошту та пароль.

Коли користувач увійшов, то вигляд головної сторінки зміниться, у верхній частині сайту буде світитись нікнейм, який посилає на акаунт користувача, та кнопка виходу. На самій головній з'явиться кнопка, за допомогою якої можна писати власні пости (рис. 3.6).

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

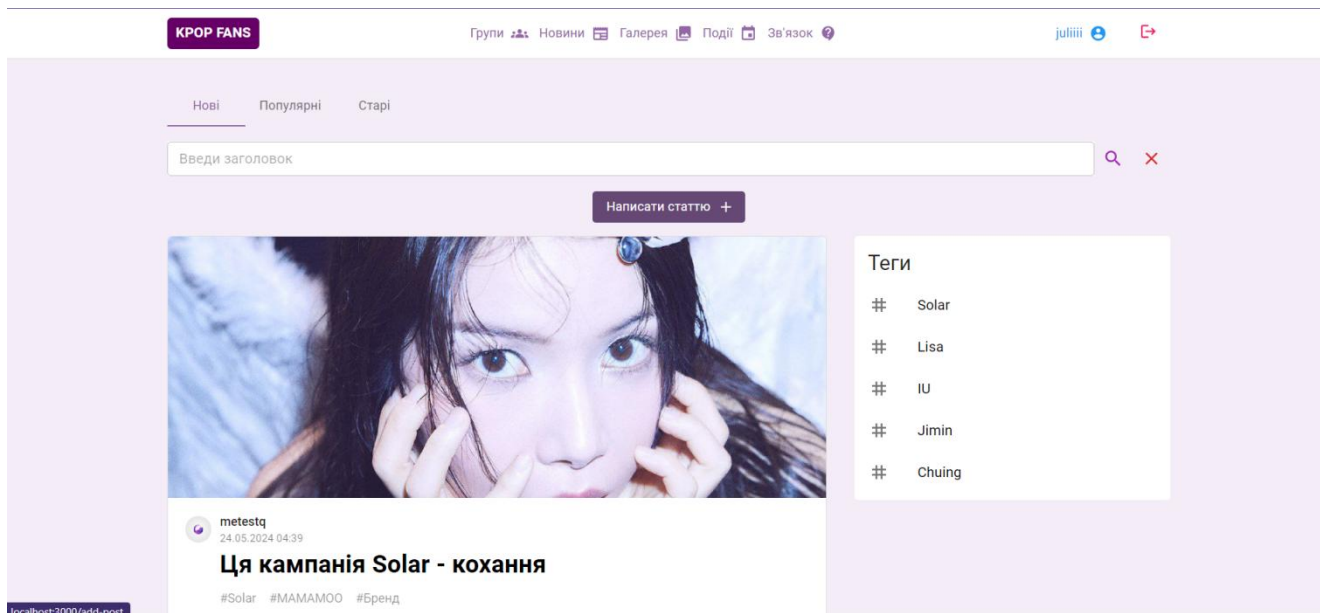


Рис. 3.6 - Головна сторінка для зареєстрованого користувача
Сторінка для додавання посту виглядає наступним чином (рис. 3.7):

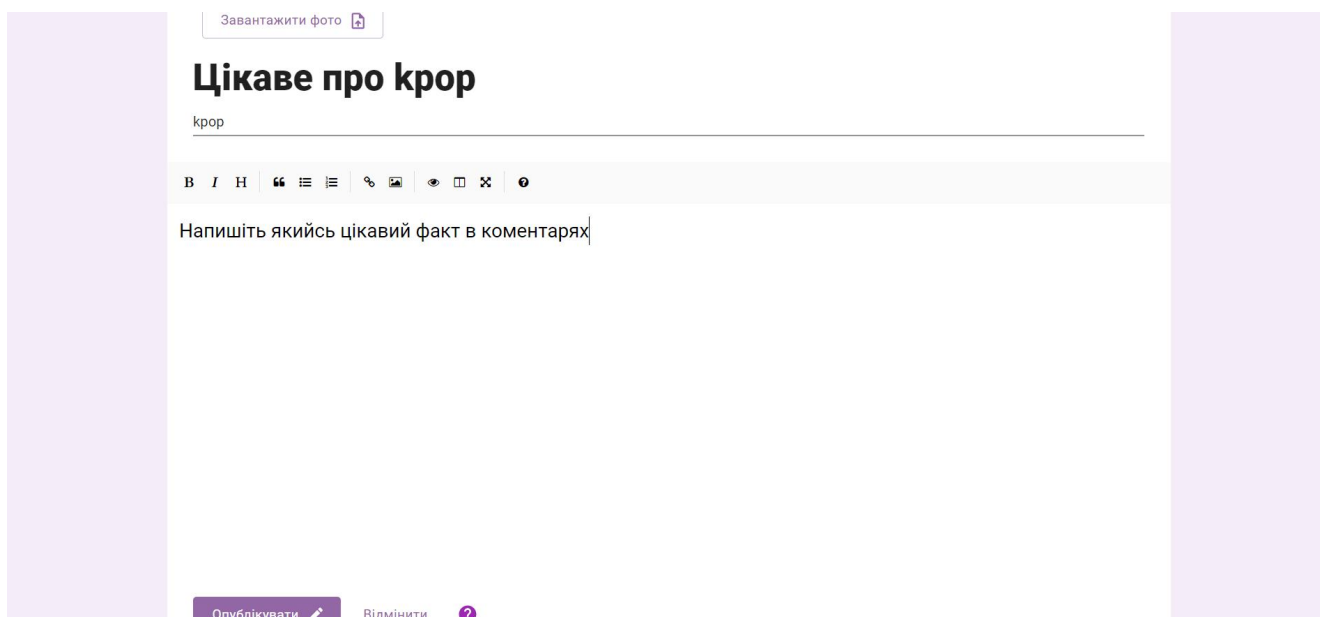


Рис. 3.7 - Сторінка додавання постів

Тут є можливість завантажити фото, написати заголовок, теги, якийсь текст, також є кнопки Опублікувати, Відмінити та посилання на сторінку з політикою сайту. Якщо ми відмінимо, то опинимось на головній сторінці. Якщо опублікуємо, то ми створимо пост та нас перенаправить на створений повний пост. У повному пості ми бачимо всю інформацію, кількість переглядів та кількість коментарів. Нижче ми можемо написати коментар (рис. 3.8).

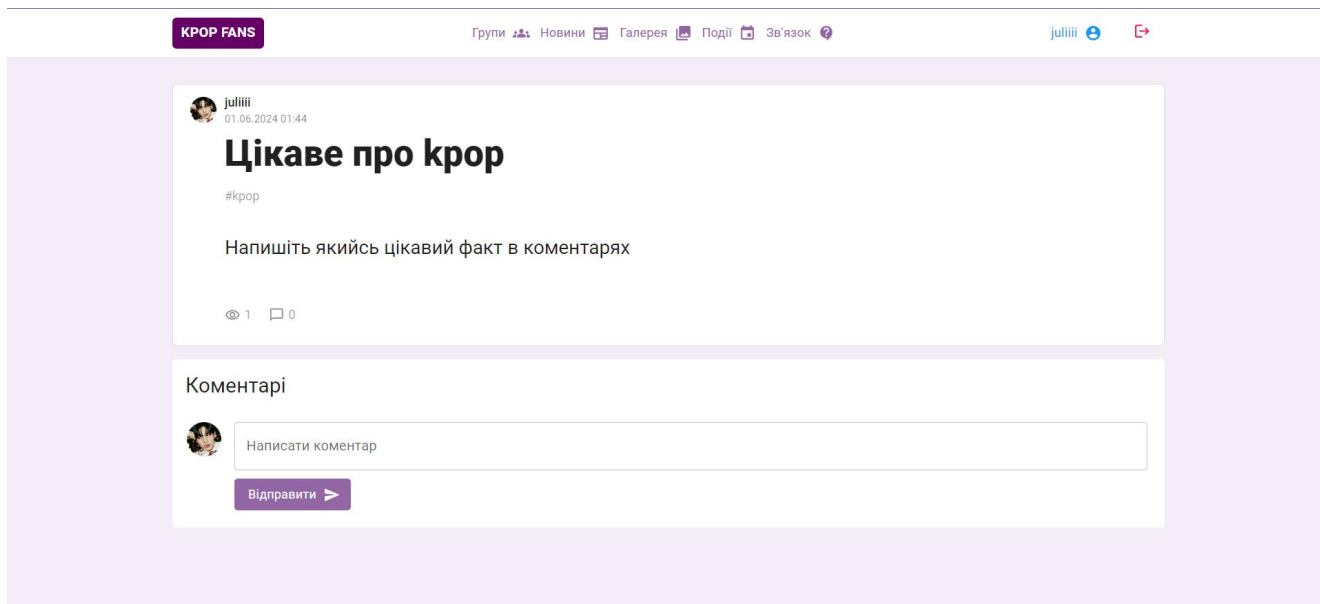


Рис. 3.8 - Повний пост з рядком коментаря

Коли ми напишемо коментар та опублікуємо його, то у цьому блоці з'явиться наш коментар, який ми можемо видалити (рис. 3.9).

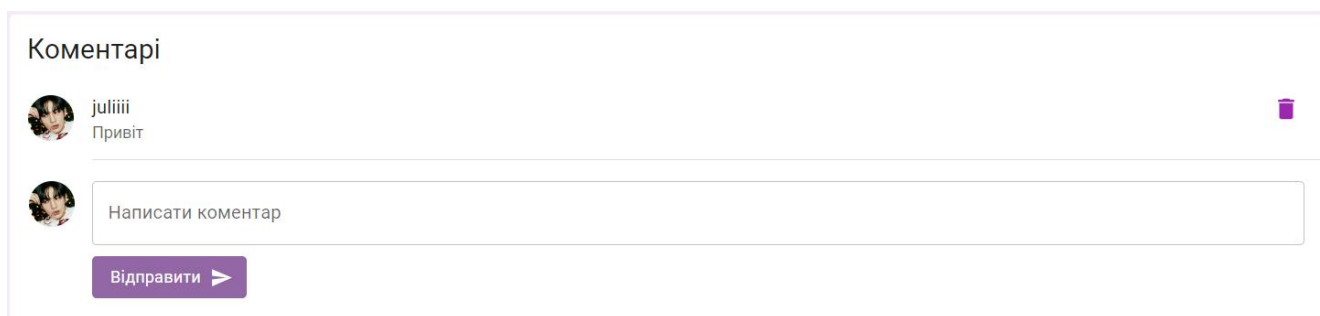


Рис. 3.9 - Блок коментарів

На сайті є роль модератора, для нього цей блок буде виглядати по іншому. У нього з'явиться можливість видаляти коментарі кожного, не тільки свого (рис. 3.10).

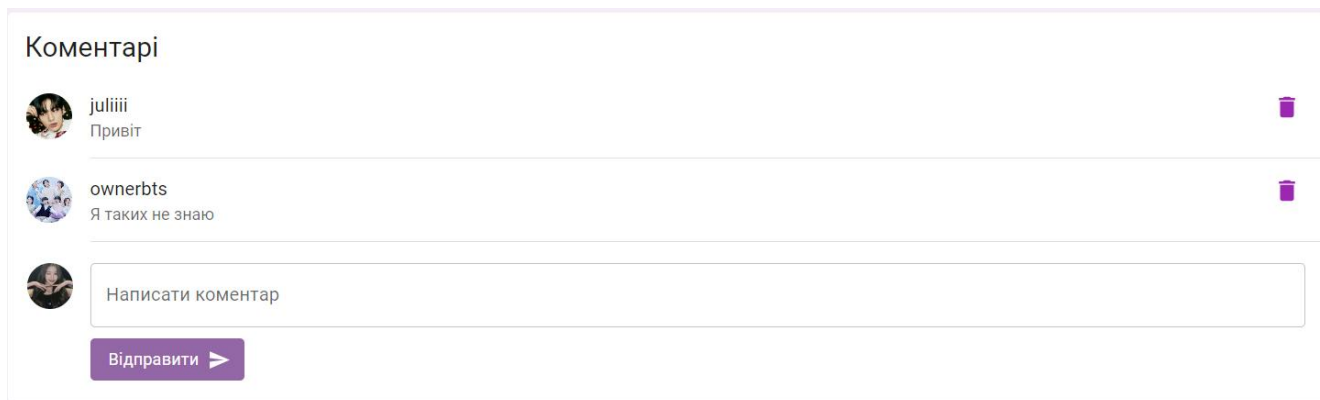


Рис. 3.10 - Блок коментарів для модератора

В цілому, на сайті є не тільки пости, а й групи, де знаходиться перелік всіх груп, які є на цьому сайті (рис. 3.11).

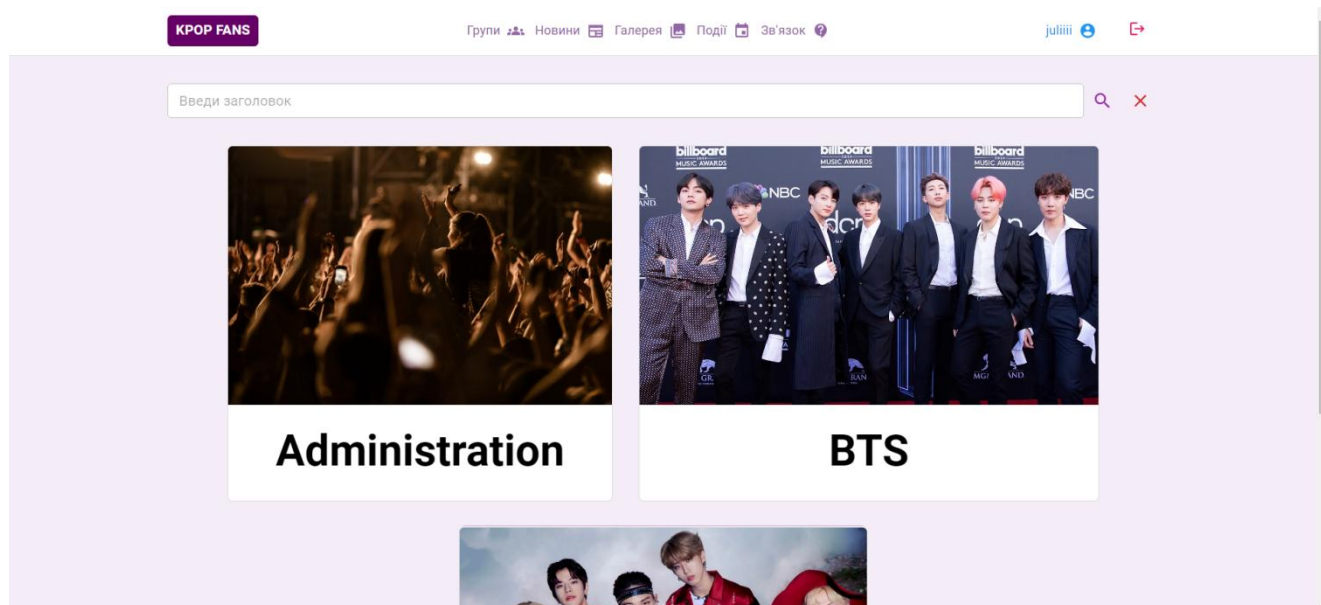


Рис. 3.11 - Сторінка груп

Є новиний блок, яким управляє адміністратор, у нього для цього з'являється спеціальна кнопка написати новину (рис. 3.12).

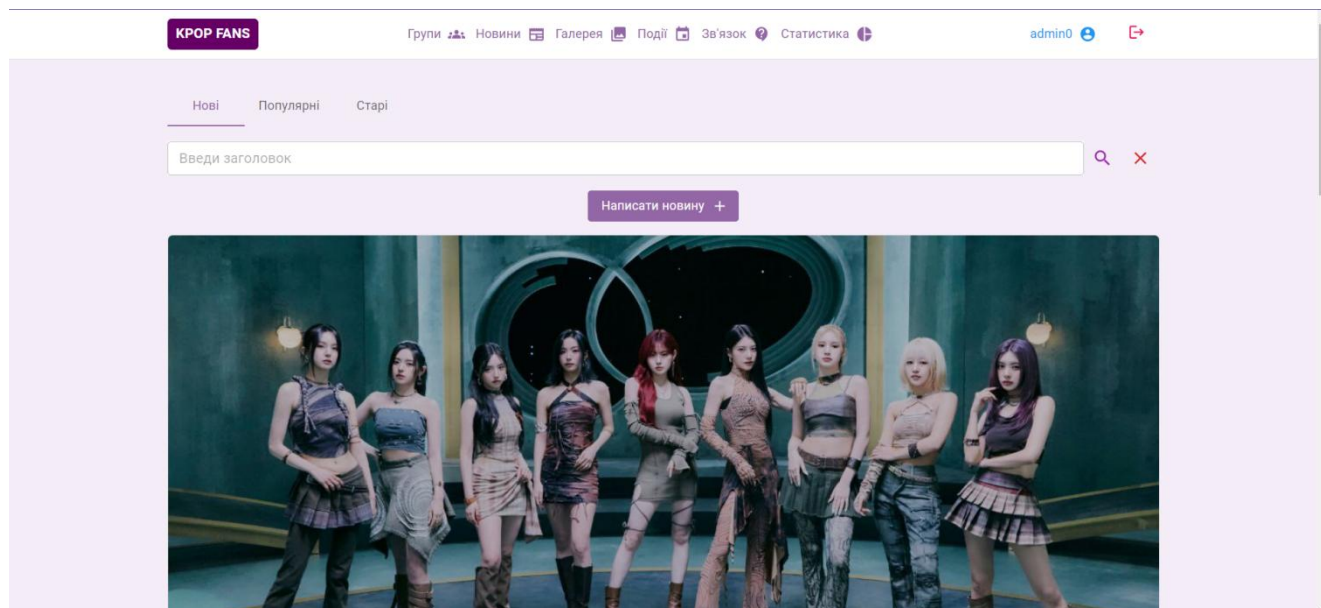


Рис. 3.12 - Сторінка новин

На сайті є Галерея, куди всі зареєстровані користувачі можуть завантажувати тематичні фото, які можуть набирати певні перегляди (рис. 3.13).

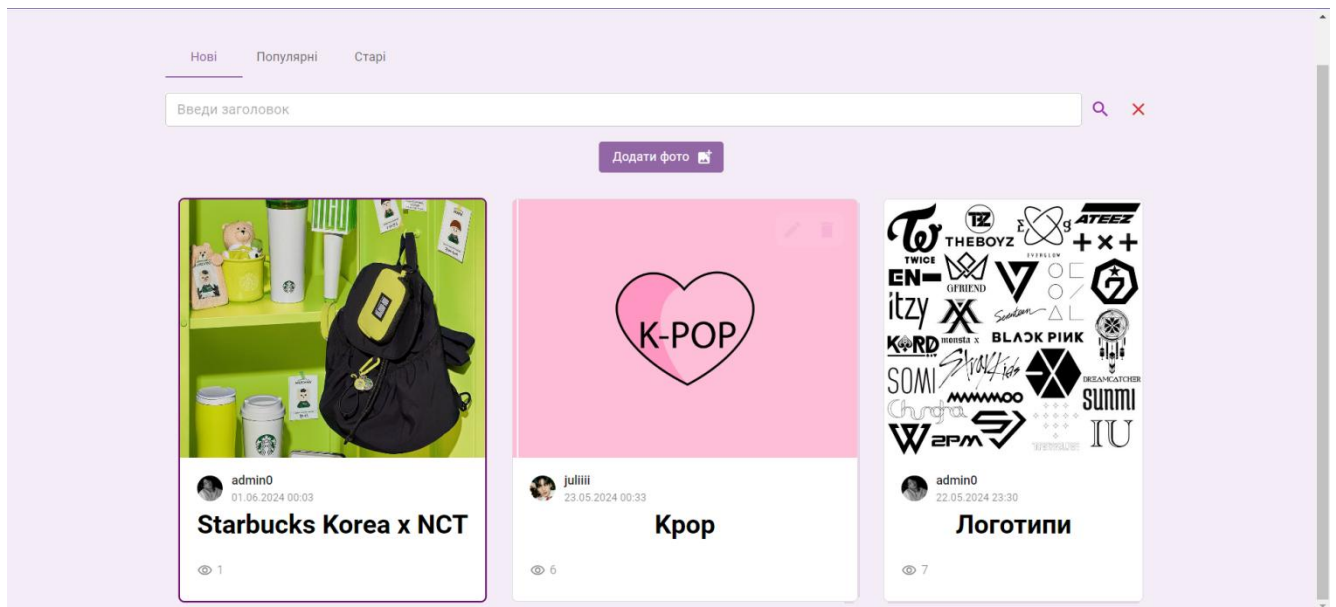


Рис. 3.13 - Сторінка Галерея

Також є окрема сторінка сьогоднішніх подій, куди виводяться всі сьогоднішні події груп сайту (рис. 3.14). Якщо ж подій не буде, то буде повідомлення, що подій немає.

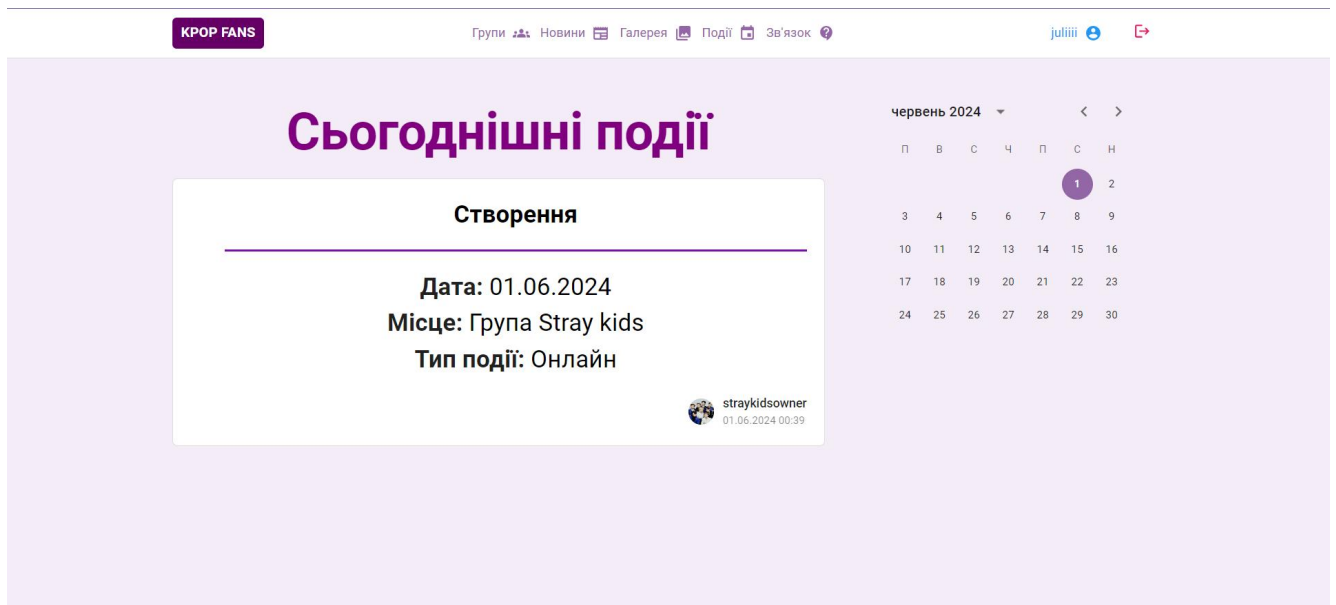


Рис. 3.14 - Сторінка Події

У кожного зареєстрованого користувача є свій профіль. У профілі є детальна інформація про користувача, також зверху у блоці інформації є три кнопки. Перша - необхідна для редагування профілю, друга - для зміни паролю, а третя посилає нас на сторінку з правами сайту. Після інформації, нижче, завантажуються всі публікації, які робив користувач (рис. 3.15).

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

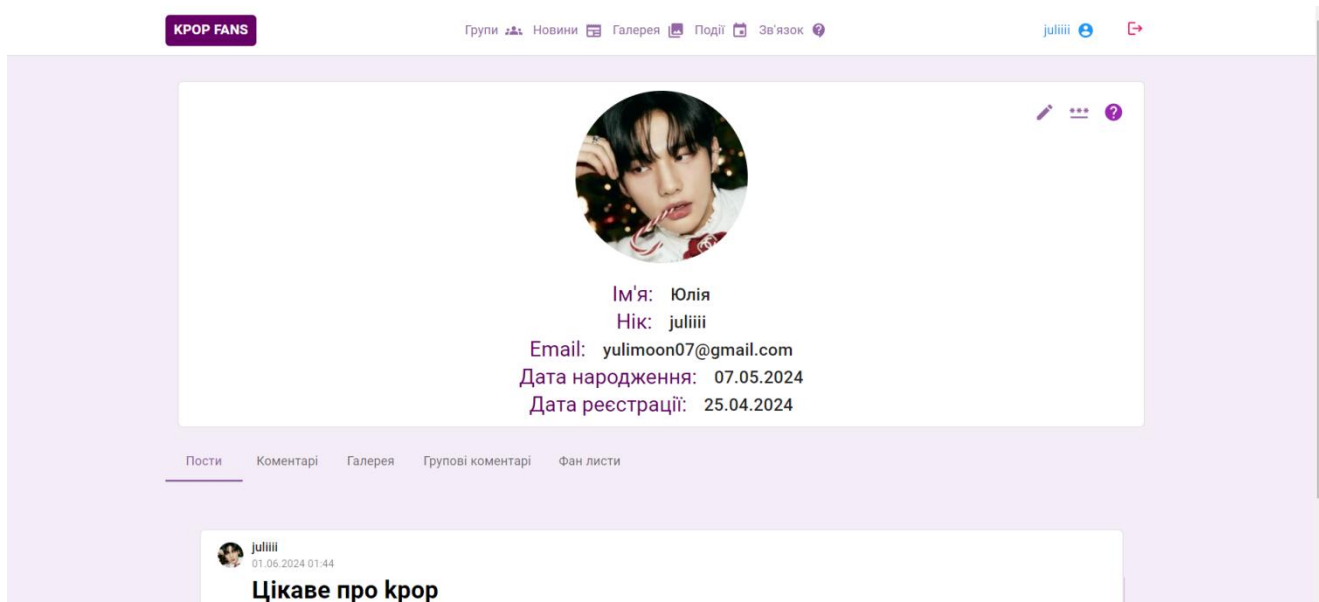


Рис. 3.15 - Профіль користувача

На сайті є вкладка зв'язку, туди зареєстрований користувач може написати повідомлення для адміністратора. На цій сторінці, нижче, для нього висвічуються всі отримані повідомлення, які він може переглядати та видаляти (рис. 3.16).

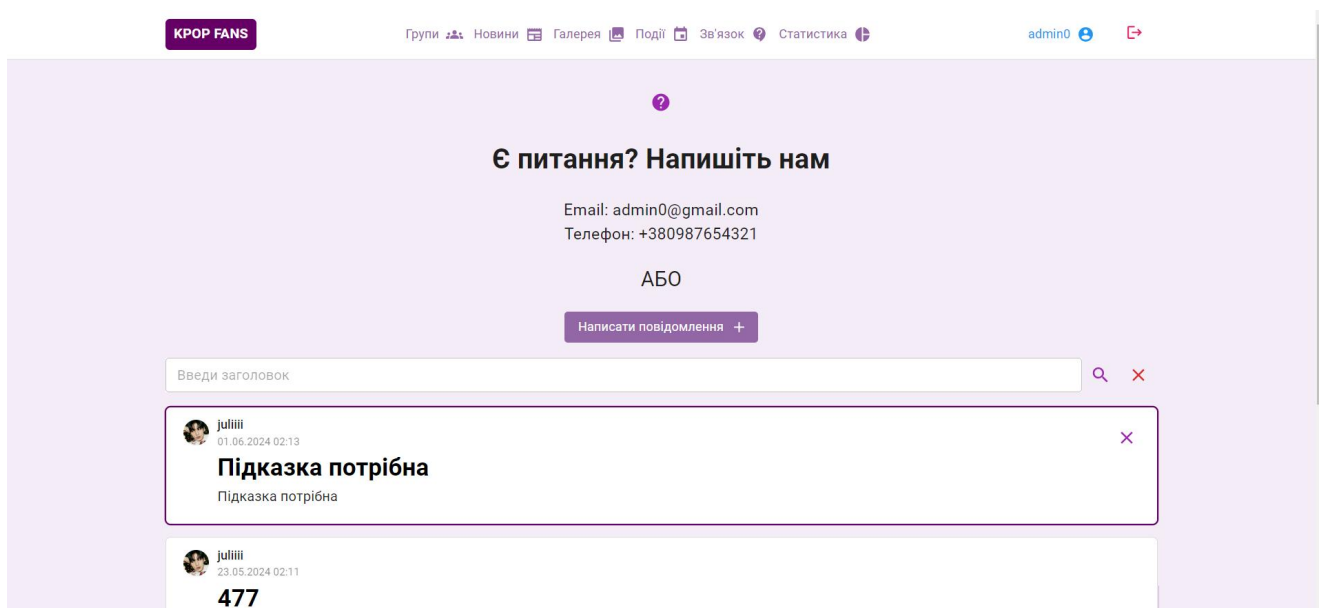


Рис. 3.16 - Вигляд сторінки зв'язку в ролі адміністратора

Для адміністратора окремо є сторінка зі статистикою сайту. Тут розташована статистика основна по певним категоріям публікацій. Також є статистика про групи сайту та всієї інформації, яка стосується груп. Третя вкладка - містить в собі певну статистику про користувачів. Таким чином, адміністратор, може переглядати статистику, яка його цікавить (рис. 3.17).

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

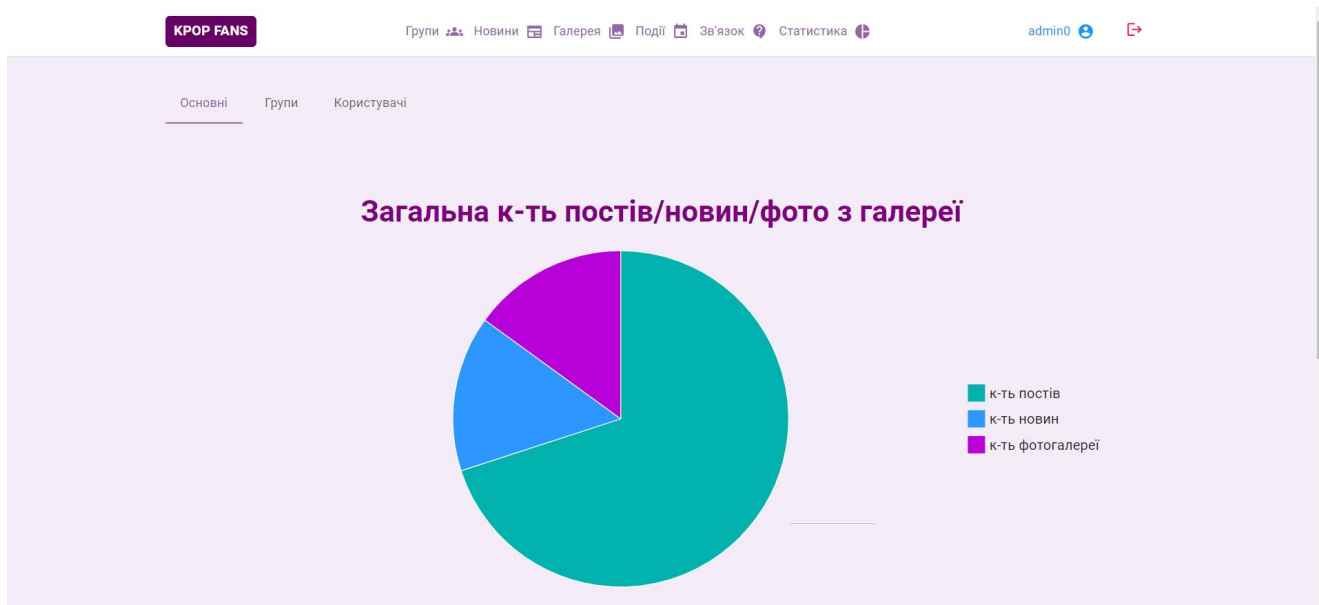


Рис. 3.17 - Статистика сайту

Найважливішим елементом сайту є групи, за кожну групу відповідає певний власник. Від ролі власника одного з фан-груп розглянемо вигляд та наповнення групи. Сторінка групи виглядає наступним чином (рис. 3.18):

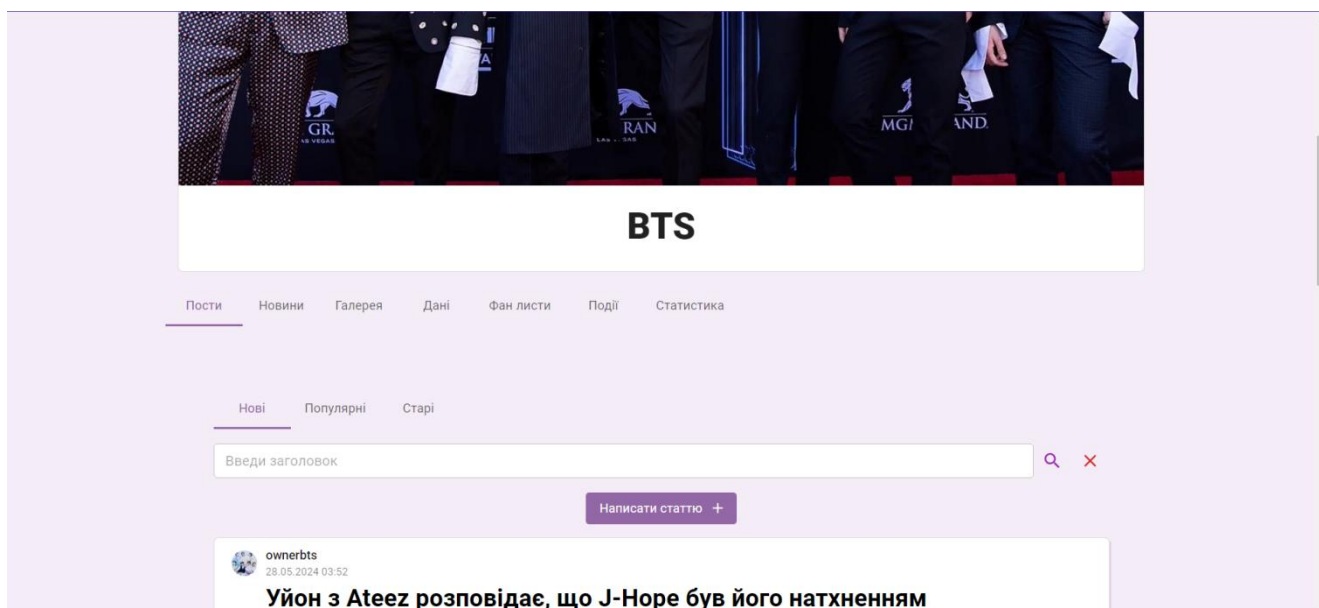


Рис. 3.18 - Вигляд групи з блоком Пости

Сторінка складається з фото групи та назви. Нижче розташовані вкладки групи, такі як Пости, Новини, Галерея, Дані, Фан листи, Події та Статистика. На фото вище (див. рис. 3.18) зображено блок постів, куди власник може постити якісь тематичні пости, а користувачі переглядати та залишати коментарі.

Далі, є блок новин, де власник може публікувати новити, які стосуються цієї групи (рис. 3.19).

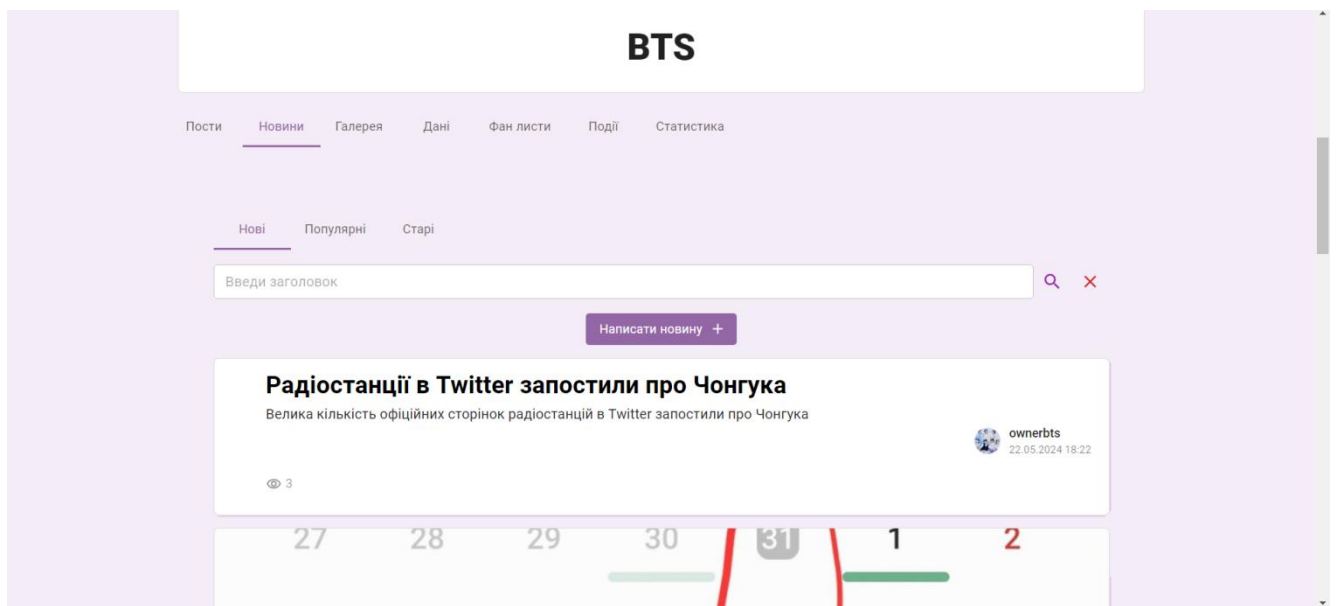


Рис. 3.19 - Вигляд групи з блоком Новини

Також, є галерея, куди власник може завантажувати якийсь фотоконтент (рис. 3.20).

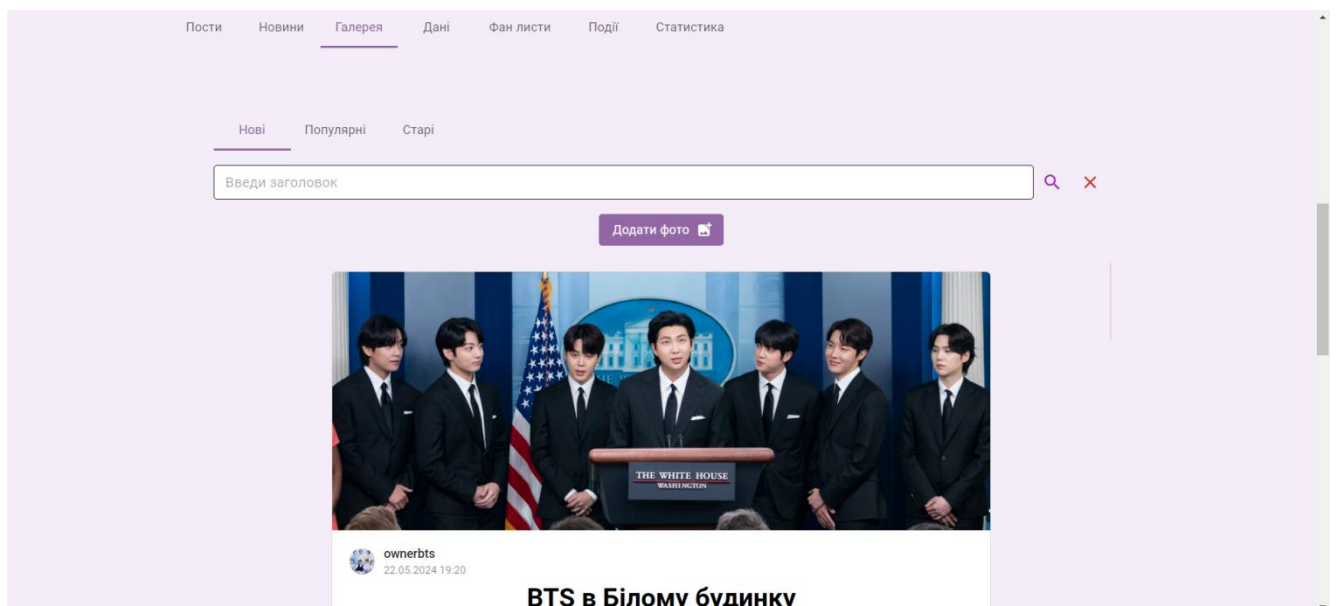


Рис. 3.20 - Вигляд групи з блоком Галерея

Є вкладка з даними про саму групу, де детально описується інформація про групу, час створення, учасників, історію, дискографію та багато іншого (рис. 3.21).

Пости
Новини
Галерея
Дані
Фан листи
Події
Статистика

Заповнити +

Інші назви:

Bangtan Boys

Жанри:

Поп, хіп-хоп, диско, рок, k-поп, j-поп

Дебют:

13 червня 2013

Агенство:

Основні дані:

Рис. 3.21 - Частина блоку з даними групи

Для звичайних, зареєстрованих користувачів є можливість в групі публікувати свою творчість, яка стосується цієї групи, побажання та інше (рис. 3.22).

Пости
Новини
Галерея
Дані
Фан листи
Події
Статистика

Нові
Популярні
Старі

Введи заголовок

Написати фан лист +



Рис. 3.22 - Фан-листи групи

Кожен власник має можливість у своїй групі створювати події, які стосуються групи, потім ці події будуть ще додатково на сторінці всіх сьогоdnішніх подій сайту. Звичайно, події можна переглядати і в самій групі (рис. 3.23).

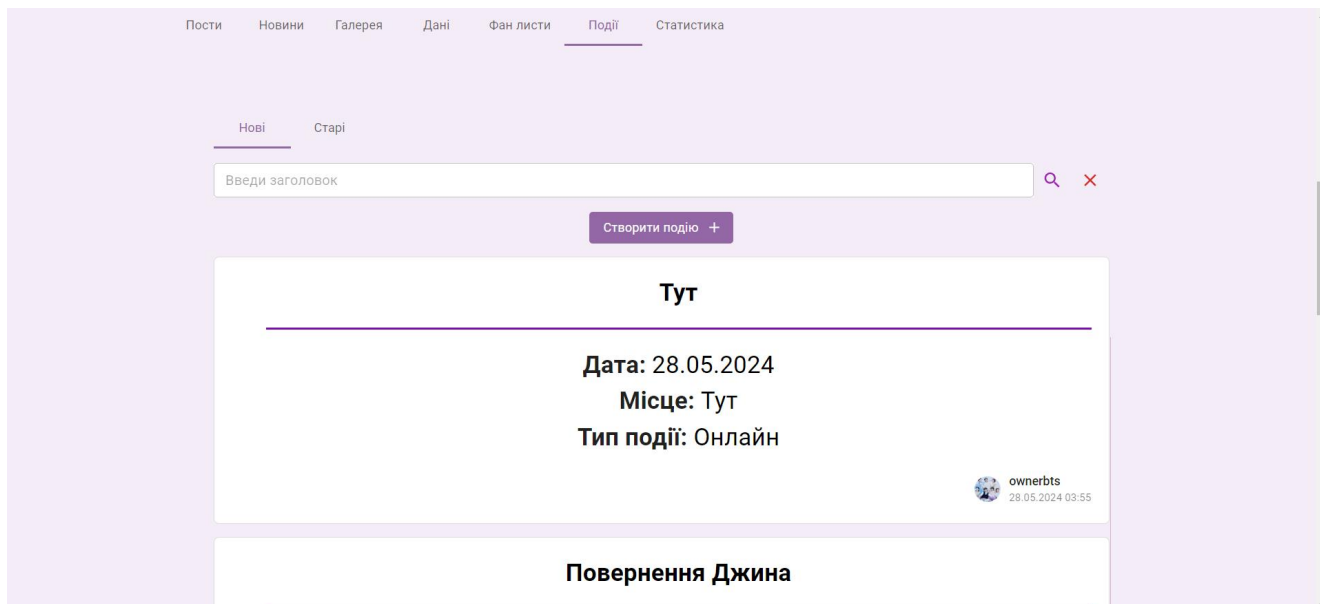


Рис. 3.23 - Події групи

Для власника ще є окремо вкладка статистика. У ній містяться всі статистичні дані по цій групі, які може зацікавити власника (рис. 3.24).

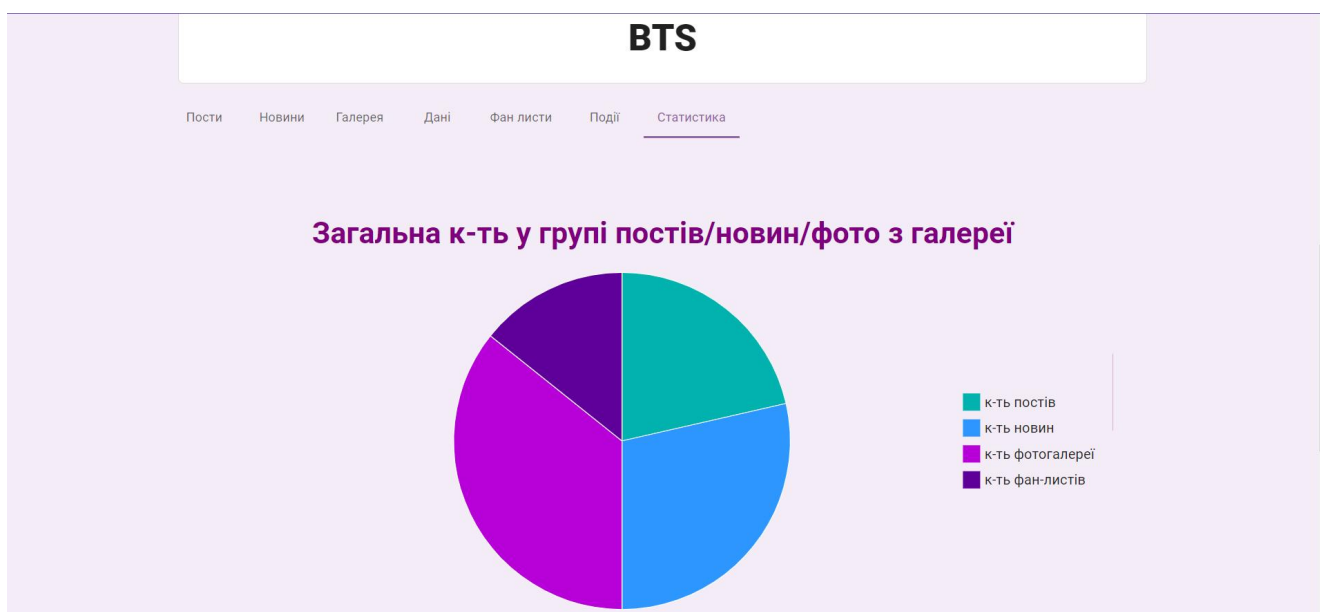


Рис. 3.24 - Статистика групи

Таким чином, ми завершили огляд всієї структури сайту і розглянули всі ключові моменти.

3.3. Тестування роботи програмного комплексу

Важливою частиною життєвого циклу розробки ПЗ є тестування програмного продукту.

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

Тестування - це процес оцінки програмного продукту з метою виявлення дефектів, забезпечення якості та відповідності вимогам. Воно включає в себе різні види та методи тестування [28].

Для тестування свого сайту я використаю ручний метод [29], тобто буду виконувати тест-кейси вручну без використання автоматизованих інструментів. Тест-кейс - це детально описаний набір дій, умов та очікуваних результатів, який використовується для перевірки відповідності програмного забезпечення заданим вимогам. Тест-кейси є основою процесу тестування та допомагають визначити, чи працює система або її компонент коректно [30].

Тест-кейс 1

У цьому тест-кейсі, я спробую як не зареєстрований користувач зайти на деякі вкладки, які доступні тільки зареєстрованим.

Таблиця 3.1

Тест-кейс 1

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Перехід за адресою /add-post	Мене не пустить на цю вкладку і перенаправить на головну	Як і очікувалось	Виконано
2	Перехід за адресою /add-gallery	Мене не пустить на цю вкладку і перенаправить на головну	Як і очікувалось	Виконано
3	Перехід за адресою /add-contactUs	Мене не пустить на цю вкладку і перенаправить на головну	Як і очікувалось	Виконано
4	Перехід за адресою /user	Мене не пустить на цю вкладку і перенаправить на головну	Як і очікувалось	Виконано
5	Перехід за адресою /groups/664bcca5bc2e302a60ae2510/add-fanlist	Мене не пустить на цю вкладку і перенаправить на головну	Як і очікувалось	Виконано

Тест-кейс 2

У цьому тест-кейсі, спробую перевірити реєстрацію, вхід у систему валідацію деяких форм і чи вони не створять публікації з неправильними полями.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 3.2

Тест-кейс 2

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Відправка пустої форми реєстрації	Мені не доступна кнопка відправки	Як і очікувалось	Виконано
2	Введення полів не вірних форматів та відправка форми	Видасть повідомлення про те, що не вдалось зареєструвати користувача	Як і очікувалось	Виконано
3	Вхід під поштою, яка не була зареєстрована на сайті	Повідомлення, що не вдалось авторизуватись	Як і очікувалось	Виконано
4	Публікація посту з неправильно введеними даними	Повідомлення про помилку створення	Як і очікувалось	Виконано
5	Публікація посту з правильними даними	Успішна публікація та перехід до неї	Як і очікувалось	Виконано
6	Публікація в галерею без назви фото	Повідомлення про помилку при створені	Як і очікувалось	Виконано

Тест-кейс 3

У цьому тест-кейсі, я спробую завантажити та опублікувати деякі зображення з різними розширеннями - jfif та gif.

Таблиця 3.3

Тест-кейс 3

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Завантаження у форму картинки формату jfif	Без проблем завантажиться та відобразиться	Як і очікувалось	Виконано
2	Відправка форми та створення публікації	Без проблем створиться і відобразиться	Як і очікувалось	Виконано
3	Перегляд публікації з фото формату jfif	Без дефектів відобразиться картинка	Як і очікувалось	Виконано

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

1	2	3	4	5
4	Завантаження у форму картинки формату gif	Без проблем завантажиться та відобразиться	Як і очікувалось	Виконано
5	Відправка форми та створення публікації	Без проблем створиться і відобразиться	Як і очікувалось	Виконано
6	Перегляд публікації з фото формату gif	Без дефектів відобразиться анімована картинка	Як і очікувалось	Виконано

Отриманий результат (рис. 3.25):

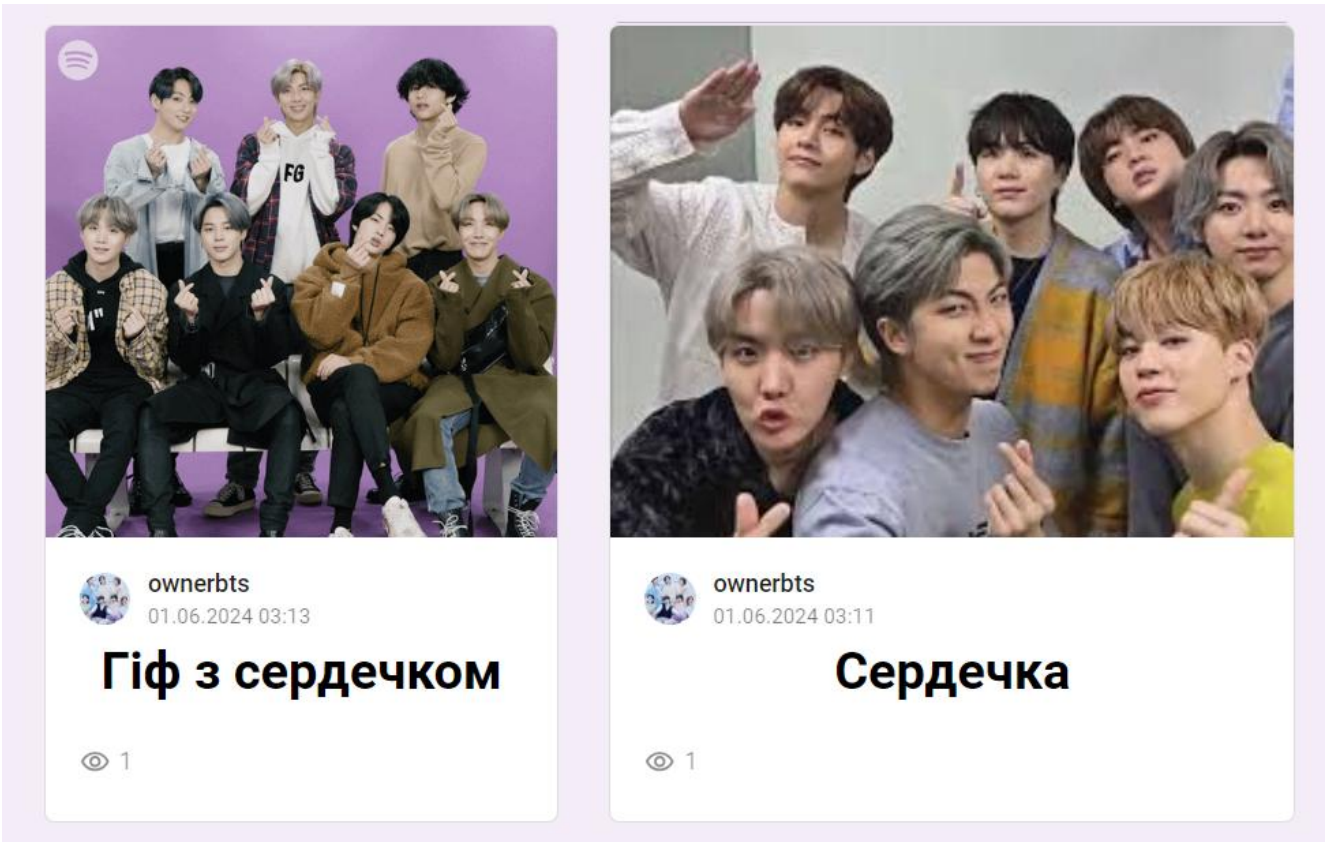


Рис. 3.25 - Опубліковані фото різних форматів

Отже, було протестовано всі випадки, у яких могли б виникнути якісь непередбачувані моменти і було з’ясовано, що все працює правильно.

Висновки до третього розділу

У третьому розділі було розглянуто інтерфейс сайту та як працювати з ним. У першому підрозділі був присвячений порядку встановлення та налаштування

параметрів системи. Також було докладно описано як розгортається кожен компонент системи та сама система.

У другому підрозділі розглядалась структура інтерфейсу веб-орієнтованої системи. Інтерфейс було розроблено з урахуванням принципів зручності та інтуїтивності, що дозволило легко орієнтуватися в системі без спеціальної підготовки. Кожен компонент інтерфейсу забезпечував швидкий доступ до відповідних функцій та інформації, що сприяло ефективному використанню системи.

У третьому підрозділі було проведено тестування роботи програмного комплексу. Виявлено, що в результаті тестування сайт виявив високу робочу надійність.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У процесі виконання даної роботи було досягнуто основної мети – розробка та впровадження фан-сайту, що відповідає всім поставленим вимогам. Робота складалася з кількох етапів, включаючи аналіз, проектування, реалізацію та тестування системи.

Перший розділ був присвячений визначенню поняття фан-сайту, його основних елементів та технічних завдань для розробки. Аналіз схожих продуктів дозволив порівняти три аналоги, визначити їх основний функціонал, переваги та недоліки, що стало важливим етапом у формуванні технічних вимог до нашого сайту. Було обрано архітектуру веб-системи. Це рішення базувалося на аналізі різних архітектур і їх відповідності потребам проекту. Також були визначені інструментальні засоби. Визначення цих технічних аспектів стало основою для подальшого проектування та реалізації фан-сайту.

У другому розділі було проведено детальне проектування фан-сайту. Аналіз вимог до програмного продукту включав функціональні та нефункціональні вимоги для різних типів користувачів, що дозволило сформулювати чіткі часові характеристики реакції системи та варіанти її використання. Розроблена схема бази даних описує структуру збереження даних, що забезпечує ефективне управління інформацією на сайті. Узагальнені алгоритми роботи системи та опис взаємодії її елементів визначають, як функціонують різні компоненти під час роботи програмного комплексу. Реалізація програмного комплексу включала детальний опис методів та структур системи, що були розроблені на основі виявлених вимог.

Третій розділ був присвячений дослідженню інтерфейсу сайту та його тестуванню. Спочатку детально описано порядок встановлення та налаштування параметрів системи, а також процес розгортання кожного компонента. Інтерфейс веб-системи було розроблено таким чином, що він дозволяє користувачам легко орієнтуватися в системі без спеціальної підготовки. Кожен елемент інтерфейсу забезпечує швидкий доступ до необхідних функцій та інформації, що сприяє ефективному використанню системи. Проведене тестування програмного

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

комплексу підтвердило високу надійність сайту, що є важливим показником якості розробленої системи. Цей розділ підсумовує розробку, демонструючи готовність програмного продукту до впровадження та використання.

Таким чином, робота включала всі необхідні етапи для розробки функціонального та надійного фан-сайту, що відповідає всім поставленим вимогам. Результати дослідження мають практичну цінність та можуть бути використані для подальшого вдосконалення програмного комплексу та розвитку нових функціональних можливостей.

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 6 Things You Should Know If You Want To Be A Fansite [Електронний ресурс] – Режим доступу до ресурсу: <https://pageone.ph/6-things-you-should-know-if-you-want-to-be-a-fansite/>
2. What is a Fan Site? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.withtheband.co/post/what-is-a-fan-site>
3. Weverse - Official for All Fans [Електронний ресурс] – Режим доступу до ресурсу: <https://weverse.io/>
4. FANS - For Fans & Artists [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jypfans.com/>
5. PODOAL : KPOP, Idol, Vote [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=kr.fanbridge.podoal&hl=uk&gl=US>
6. Архітектура програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: <https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya>
7. Архітектура застосунку: компоненти та популярні патерни [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/arkhitektura-zastosunku/>
8. Мікросервісна архітектура [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d>
9. Learn About the Microservices Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.oracle.com/en/solutions/learn-architect-microservice/>
10. Що таке React JS? Як почати вивчати Реакт? Навички для react developer [Електронний ресурс] – Режим доступу до ресурсу: <https://cases.media/article/what-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer>
11. Що таке Node.js та для чого він потрібен? [Електронний ресурс] – Режим доступу до ресурсу: https://hyperhost.ua/info/uk/shho-take-nodejs-ta-dlya-cogo-vin-potriben?gad_source=1&gclid=CjwKCAjwjeuyBhBuEiwAJ3vuoeORglHYEHqWILE5bVqMT9ldaWrw_jSWGug8GgwKP47NbiaoeC-cOBoCn-oQAvD_BwE

					ІІЗ.КР.Б-121-24-ІІЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

12. Що таке MongoDB? вступ, Archітектура, функції та приклад [Електронний ресурс] – Режим доступу до ресурсу: <https://www.guru99.com/uk/what-is-mongodb.html>

13. Аналіз вимог [Електронний ресурс] – Режим доступу до ресурсу: <https://visuresolutions.com/uk/requirements-management-traceability-guide/requirements-analysis/>

14. Що таке функціональні вимоги: приклади, визначення, повний посібник [Електронний ресурс] – Режим доступу до ресурсу: <https://visuresolutions.com/uk/blog/functional-requirements/>

15. Нефункціональні вимоги: приклади, типи, підходи [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/>

16. Інструкція, як будувати UML-діаграми [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forums/topic/40575/>

17. UML Class Diagram Tutorial [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>

18. Дізнайтеся все про діаграму активності UML [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mindonmap.com/uk/blog/uml-activity-diagram/>

19. Вступ до Mongoose для MongoDB та Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://code.tutsplus.com/uk/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527a>

20. Timestamps [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/docs/timestamps.html>

21. Express [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/uk/>

22. What is middleware? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.redhat.com/en/topics/middleware/what-is-middleware>

					ІПЗ.КР.Б-121-24-ІПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

23. Cross-origin resource sharing (CORS) [Електронний ресурс] – Режим доступу до ресурсу: <https://portswigger.net/web-security/cors>
24. Multer [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/en/resources/middleware/multer.html>
25. Making HTTP requests with Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://circleci.com/blog/making-http-requests-with-axios/>
26. Для чого і коли використовується Redux [Електронний ресурс] – Режим доступу до ресурсу: <https://foxminded.ua/shcho-take-redux/>
27. Діаграма розгортання [Електронний ресурс] – Режим доступу до ресурсу: <https://www.guru99.com/uk/deployment-diagram-uml-example.html>
28. Що таке тестування пз, його етапи, види, інструменти [Електронний ресурс] – Режим доступу до ресурсу: <https://university.sigma.software/what-is-software-testing/>
29. В чому різниця між автоматизованим і ручним тестуванням? [Електронний ресурс] – Режим доступу до ресурсу: <https://university.sigma.software/manual-testing-vs-automation-testing/>
30. Правила оформлення теми тест-кейса [Електронний ресурс] – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/course-materials/test-case-topic/>

ДОДАТКИ

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

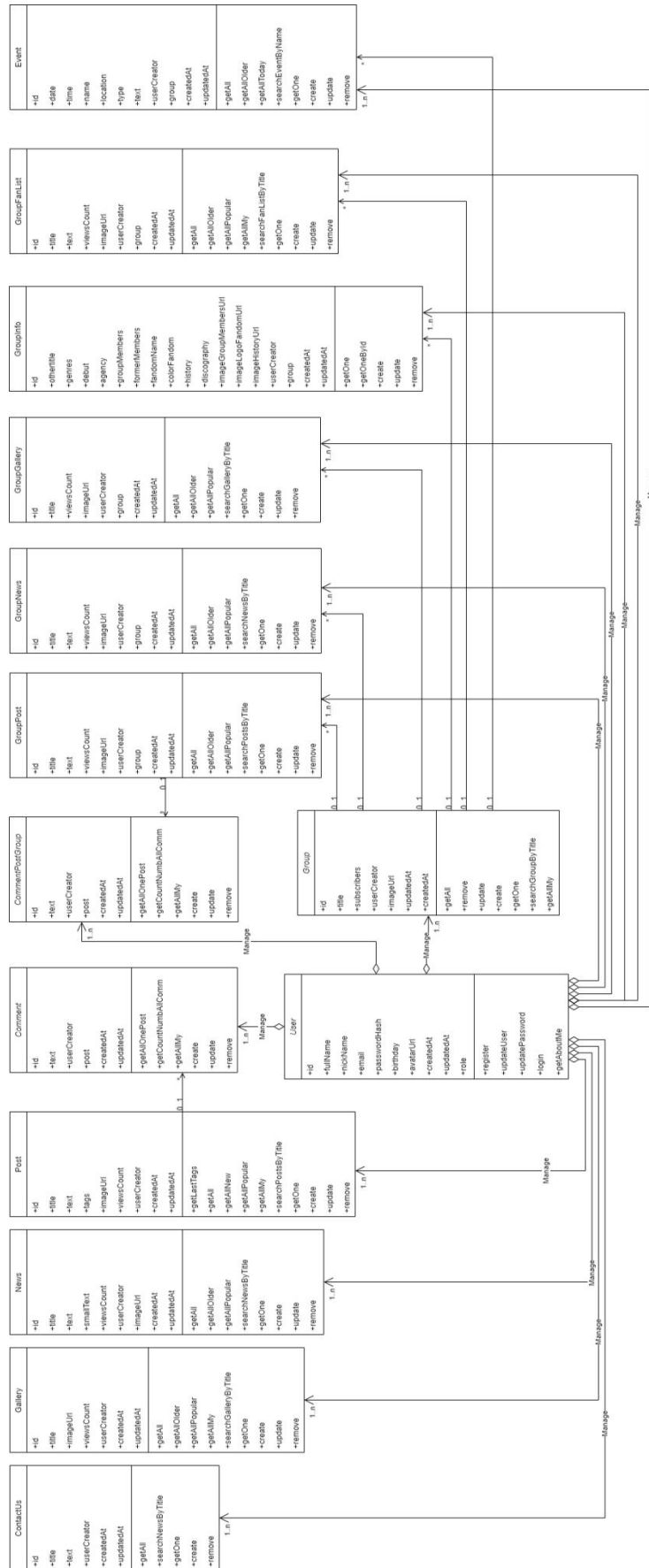


Рис. А.1 - Діаграма класів

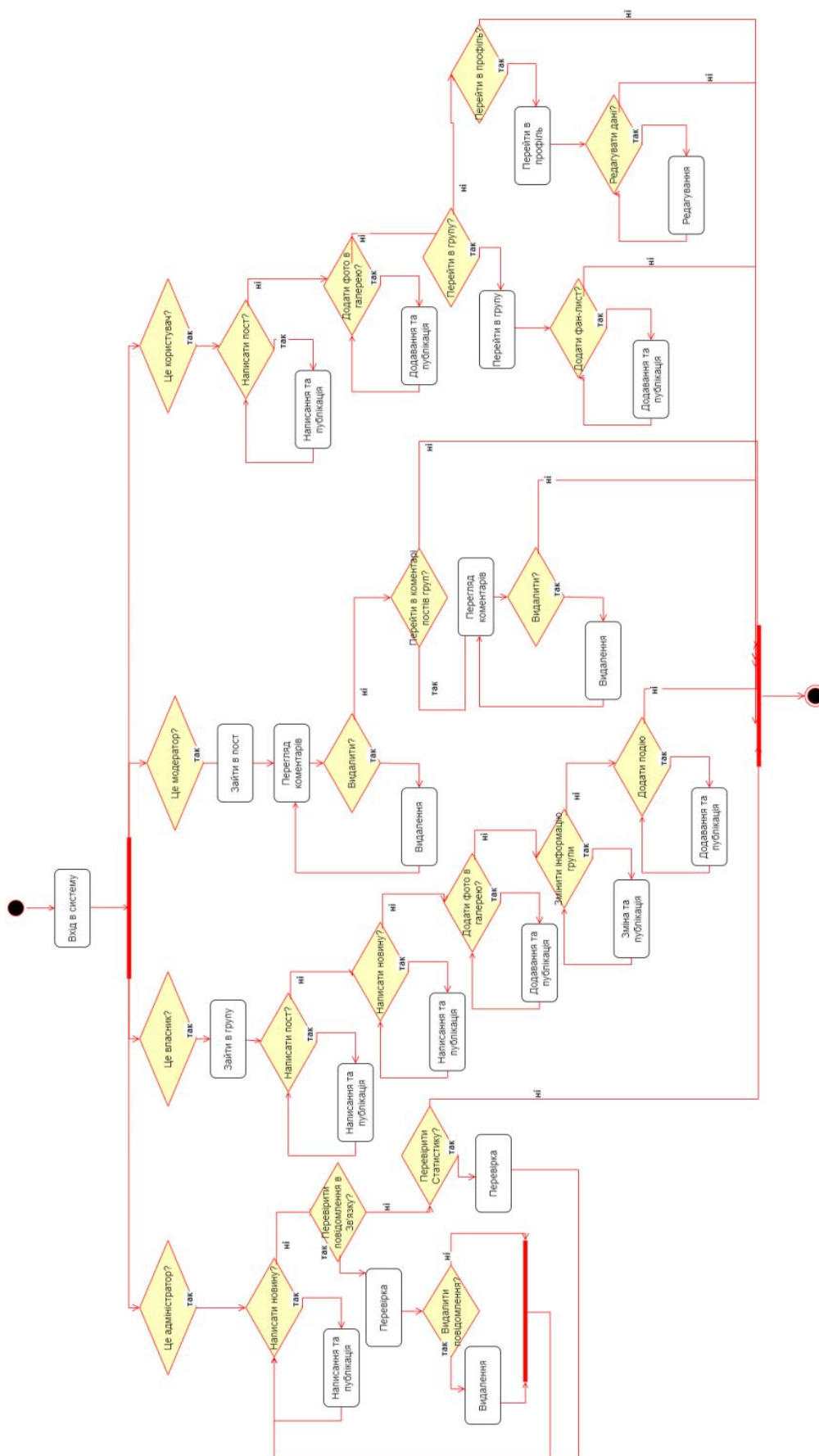


Рис. Б.1 - Діаграма активності

Лістинг group.js:

```
var mongoose = require('mongoose');

const GroupSchema = new mongoose.Schema({
  title: {
    type:String,
    required:true,
  },
  subscribers: {
    type:Number,
    default: 0,
  },
  userCreator: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required:true,
  },
  imageUrl: String,
},{
  timestamps: true,
})

module.exports = mongoose.model("Group", GroupSchema)
```

Лістинг GroupController.js:

```
const GroupModel = require('../models/Group');

const create = async (req,res) => {
  try{
    const doc = new GroupModel({
      title: req.body.title,
      imageUrl: req.body.imageUrl,
      userCreator: req.userId
    })
    const post = await doc.save()
    res.json(post)
  }
  catch(err){
    console.log(err)
    res.status(500).json({
      message: "Не вдалось оновити групу"
    })
  }
}

const getAll = async (req,res) => {
  try{
    const posts = await GroupModel.find().populate("userCreator").sort({ title: 1 }).exec()
    res.json(posts)
  }
  catch(err){
    console.log(err)
    res.status(500).json({
      message: "Не вдалось отримати групи"
    })
  }
}

const searchGroupByTitle = async (req, res) => {
  try {
    const titleQuery = req.query.title;
    if (!titleQuery) {
```

```

        return res.status(400).json({ message: 'Потрібен параметр title для пошуку' });
    }
    const posts = await GroupModel.find({ title: { $regex: titleQuery, $options: 'i' } }).populate("userCreator").sort({ title:
1 }).exec();
    res.json(posts);
  } catch (err) {
    console.log(err)
    res.status(500).json({
      message: "Не вдалось отримати групи"
    })
  }
};
const getOne = async (req, res) => {
  try {
    const groupId = req.params.id;
    const doc = await GroupModel.findOne(
      { _id: groupId },
    ).populate('userCreator');
    if (!doc) {
      return res.status(404).json({
        message: 'Група не знайдена',
      });
    }
    res.json(doc);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось отримати групу',
    });
  }
};
const getAllMy = async (req, res) => {
  try {
    const userId = req.params.id;
    const posts = await GroupModel.find({ userCreator: userId }).populate("userCreator").sort({ $natural: -1 }).exec();
    res.json(posts)
  }
  catch(err){
    console.log(err)
    res.status(500).json({
      message: "Не вдалось отримати статті"
    })
  }
}
const update = async (req, res) => {
  try {
    const groupId = req.params.id;
    const updatedGroup = await GroupModel.findOneAndUpdate(
      { _id: groupId },
      {
        title: req.body.title,
        imageUrl: req.body.imageUrl,
        /*userCreator: req.userId*/
      },
      { new: true }
    );
    if (!updatedGroup) {
      return res.status(404).json({
        message: 'Група не знайдена',
      });
    }
    res.json({
      success: true,

```

Змн.	Арк.	№ докум.	Підпис	Дата

```

    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: "Не вдалось оновити групу"
    });
  }
};
const remove = async (req, res) => {
  try {
    const groupId = req.params.id;
    const doc = await GroupModel.findOneAndDelete({ _id: groupId });
    if (!doc) {
      return res.status(404).json({
        message: 'Група не знайдена',
      });
    }
    res.json({
      success: true,
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: 'Не вдалось видалити групу',
    });
  }
};
module.exports = {create, getAll, getOne, update, remove, searchGroupByTitle, getAllMy}

```

Лістинг groupGallery.js:

```

var mongoose = require('mongoose');

const GroupGallerySchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  imageUrl: {
    type: String,
    required: true,
  },
  viewsCount: {
    type: Number,
    default: 0,
  },
  userCreator: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  group: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Group",
    required: true,
  },
}, {
  timestamps: true,
})
module.exports = mongoose.model("GroupGallery", GroupGallerySchema)

```

Лістинг index.js:

```

var express = require('express');
var mongoose = require('mongoose');

```

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

var multer = require('multer');
var cors = require('cors');

const {registerValidator, loginValidator, postCreateValidation, commentValidation, groupCreateValidation,
groupPostCreateValidation, groupNewsCreateValidation, groupGalleryCreateValidation, groupInfoCreateValidation,
contactUsCreateValidation,updateValidator, passwordValidator, eventCreateValidation} = require( './validations/index')
const {checkAuth, handleValidationErrors }= require('./utils/index');
const {UserController, PostController, CommentController, GroupController, GroupPostController, GroupNewsController,
GroupGalleryController, GroupInfoController, GroupFanListController, CommentPostGroupController, NewsController,
GalleryController, ContactUsController, EventController, StatisticController} = require('./controllers/index');
mongoose.connect("mongodb+srv://yulimoon07:*****@cluster0.eij6abo.mongodb.net/fansite?retryWrites=true&w=majority
&appName=Cluster0"
).then(()=>console.log("DB connection success"))
.catch((err)=>console.log("DB error",err))
const app = express()
//сховище для картинок
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "uploads")
  },
  filename: (req, file, cb) => {
    cb(null, file.originalname)
  }
})
const upload = multer ( {storage})
app.use(express.json())
app.use(cors())
app.use("/uploads", express.static("uploads"))
// реєстрація
app.post('/auth/register', registerValidator, handleValidationErrors, UserController.register)
// авторизація
app.post('/auth/login', loginValidator, handleValidationErrors, UserController.login)
app.patch('/auth/update',checkAuth, updateValidator, handleValidationErrors, UserController.update)
app.patch('/auth/update-pass',checkAuth, passwordValidator, handleValidationErrors, UserController.updatePassword)
//app.patch('/auth/:id/update-pass',checkAuth, passwordValidator, handleValidationErrors, UserController.update)
//загрузка картинки
app.post('/upload', checkAuth, upload.single("image"),(req, res) =>{
  res.json({
    url: `/uploads/${req.file.originalname}`
  })
})
//про користувача
app.get('/auth/me', checkAuth, UserController.getAboutMe)
//теги
app.get('/tags', PostController.getLastTags)
app.get('/posts/tags', PostController.getLastTags)
//CRUD для постів
app.get('/posts', PostController.getAll)
app.get('/posts/search', PostController.searchPostsByTitle)
app.get('/posts/popular', PostController.getAllPopular)
app.get('/posts/new', PostController.getAllNew)
app.get('/myposts', checkAuth, PostController.getAllMy)
app.get('/posts/:id', PostController.getOne)
app.post('/posts', checkAuth, postCreateValidation, handleValidationErrors, PostController.create)
app.delete('/posts/:id', checkAuth, PostController.remove)
app.patch('/posts/:id', checkAuth,postCreateValidation, handleValidationErrors, PostController.update)
//CRUD для коментарів
app.post('/posts/:id/comment', checkAuth, commentValidation, handleValidationErrors, CommentController.create)
app.get('/posts/:id/comment', CommentController.getAllOnePost)
app.get('/posts/:id/commentnumb', CommentController.getCountNumbAllComm)
app.patch('/comment/:id', checkAuth, commentValidation, handleValidationErrors, CommentController.update)
app.delete('/comment/:id', checkAuth, CommentController.remove)
app.get('/mycomment', checkAuth, CommentController.getAllMy)

```

					ПІЗ.КР.Б-121-24-ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

//CRUD для групи
app.get('/groups', GroupController.getAll)
app.get('/mygroups/:id', GroupController.getAllMy)
app.get('/groups/search', GroupController.searchGroupByTitle)
app.get('/groups/:id', GroupController.getOne)
app.post('/groups', checkAuth, groupCreateValidation, handleValidationErrors, GroupController.create)
app.patch('/groups/:id', checkAuth, groupCreateValidation, handleValidationErrors, GroupController.update)
app.delete('/groups/:id', checkAuth, GroupController.remove)
//CRUD для постів групи
app.post('/groups/:id/group-post', checkAuth, groupPostCreateValidation, handleValidationErrors, GroupPostController.create)
app.get('/groups/:id/group-post', GroupPostController.getAll)
app.get('/groups/:id/group-post/old', GroupPostController.getAllOlder)
app.get('/groups/:id/group-post/popular', GroupPostController.getAllPopular)
app.get('/groups/:id/group-post/search', GroupPostController.searchPostsByTitle)
app.get('/group-post/:id', GroupPostController.getOne)
app.patch('/group-post/:id', checkAuth, groupPostCreateValidation, handleValidationErrors, GroupPostController.update)
app.delete('/group-post/:id', checkAuth, GroupPostController.remove)
//CRUD для новин
app.get('/news', NewsController.getAll)
app.get('/news/search', NewsController.searchNewsByTitle)
app.get('/news/old', NewsController.getAllOlder)
app.get('/news/popular', NewsController.getAllPopular)
app.get('/news/:id', NewsController.getOne)
app.post('/news', checkAuth, groupNewsCreateValidation, handleValidationErrors, NewsController.create)
app.patch('/news/:id', checkAuth, groupNewsCreateValidation, handleValidationErrors, NewsController.update)
app.delete('/news/:id', checkAuth, NewsController.remove)
//CRUD для новин групи
app.post('/groups/:id/group-news', checkAuth, groupNewsCreateValidation, handleValidationErrors, GroupNewsController.create)
app.get('/groups/:id/group-news', GroupNewsController.getAll)
app.get('/groups/:id/group-news/old', GroupNewsController.getAllOlder)
app.get('/groups/:id/group-news/popular', GroupNewsController.getAllPopular)
app.get('/groups/:id/group-news/search', GroupNewsController.searchNewsByTitle)
app.get('/group-news/:id', GroupNewsController.getOne)
app.patch('/group-news/:id', checkAuth, groupNewsCreateValidation, handleValidationErrors, GroupNewsController.update)
app.delete('/group-news/:id', checkAuth, GroupNewsController.remove)
//CRUD для галереї
app.post('/gallery', checkAuth, groupGalleryCreateValidation, handleValidationErrors, GalleryController.create)
app.get('/gallery/search', GalleryController.searchGalleryByTitle)
app.get('/gallery/old', GalleryController.getAllOlder)
app.get('/gallery/popular', GalleryController.getAllPopular)
app.get('/gallery', GalleryController.getAll)
app.get('/gallery/:id', GalleryController.getOne)
app.patch('/gallery/:id', checkAuth, groupGalleryCreateValidation, handleValidationErrors, GalleryController.update)
app.delete('/gallery/:id', checkAuth, GalleryController.remove)
app.get('/mygallery', checkAuth, GalleryController.getAllMy)
//CRUD для галереї групи
app.post('/groups/:id/group-gallery', checkAuth, groupGalleryCreateValidation, handleValidationErrors,
GroupGalleryController.create)
app.get('/groups/:id/group-gallery', GroupGalleryController.getAll)
app.get('/groups/:id/group-gallery/search', GroupGalleryController.searchGalleryByTitle)
app.get('/groups/:id/group-gallery/old', GroupGalleryController.getAllOlder)
app.get('/groups/:id/group-gallery/popular', GroupGalleryController.getAllPopular)
app.get('/group-gallery/:id', GroupGalleryController.getOne)
app.patch('/group-gallery/:id', checkAuth, groupGalleryCreateValidation, handleValidationErrors, GroupGalleryController.update)
app.delete('/group-gallery/:id', checkAuth, GroupGalleryController.remove)
//CRUD для повідомлень Зв'язку
app.get('/contact-us', ContactUsController.getAll)
app.get('/contact-us/search', ContactUsController.searchContactUsByTitle)
app.get('/contact-us/:id', ContactUsController.getOne)
app.post('/contact-us', checkAuth, contactUsCreateValidation, handleValidationErrors, ContactUsController.create)
app.delete('/contact-us/:id', checkAuth, ContactUsController.remove)
//CRUD для даних групи
app.post('/groups/:id/group-info', checkAuth, groupInfoCreateValidation, handleValidationErrors, GroupInfoController.create)

```

```

app.get('/groups/:id/group-info', GroupInfoController.getOne)
app.get('/group-info/:id', GroupInfoController.getOneById)
app.patch('/group-info/:id', checkAuth, groupInfoCreateValidation, handleValidationErrors, GroupInfoController.update)
app.delete('/group-info/:id', checkAuth, GroupInfoController.remove)
//CRUD для фан листів групи
app.post('/groups/:id/group-fanlist', checkAuth, groupPostCreateValidation, handleValidationErrors, GroupFanListController.create)
app.get('/groups/:id/group-fanlist', GroupFanListController.getAll)
app.get('/groups/:id/group-fanlist/search', GroupFanListController.searchFanListByTitle)
app.get('/groups/:id/group-fanlist/old', GroupFanListController.getAllOlder)
app.get('/groups/:id/group-fanlist/popular', GroupFanListController.getAllPopular)
app.get('/group-fanlist/:id', GroupFanListController.getOne)
app.patch('/group-fanlist/:id', checkAuth, groupPostCreateValidation, handleValidationErrors, GroupFanListController.update)
app.delete('/group-fanlist/:id', checkAuth, GroupFanListController.remove)
app.get('/mygroup-fanlist', checkAuth, GroupFanListController.getAllMy)
//CRUD для коментарів постів групи
app.post('/group-post/:id/group-postcom', checkAuth, commentValidation, handleValidationErrors,
CommentPostGroupController.create)
app.get('/group-post/:id/group-postcom', CommentPostGroupController.getAllOnePost)
app.get('/group-post/:id/group-postcomnum', CommentPostGroupController.getCountNumbAllComm)
app.patch('/group-postcom/:id', checkAuth, commentValidation, handleValidationErrors, CommentPostGroupController.update)
app.delete('/group-postcom/:id', checkAuth, CommentPostGroupController.remove)
app.get('/mygroup-postcom', checkAuth, CommentPostGroupController.getAllMy)
//CRUD для подій групи
app.post('/groups/:id/event', checkAuth, eventCreateValidation, handleValidationErrors, EventController.create)
app.get('/groups/:id/event', EventController.getAll)
app.get('/groups/:id/event/old', EventController.getAllOlder)
app.get('/groups/:id/event/search', EventController.searchEventByName)
app.get('/event/:id', EventController.getOne)
app.get('/today-event', EventController.getAllToday)
app.patch('/event/:id', checkAuth, eventCreateValidation, handleValidationErrors, EventController.update)
app.delete('/event/:id', checkAuth, EventController.remove)
//Get статистика
app.get('/countPostNewsPhoto', StatisticController.getCountPostNewsPhoto)
app.get('/totalViewsCount', StatisticController.getTotalViewsCount)
app.get('/countGroup', StatisticController.getCountGroup)
app.get('/countGroupPostNewsPhoto', StatisticController.getCountGroupPostNewsPhoto)
app.get('/totalGroupViewsCount', StatisticController.getTotalGroupViewsCount)
app.get('/myCountGroupPostNewsPhoto/:id', StatisticController.getMyCountGroupPostNewsPhoto)
app.get('/myTotalGroupViewsCount/:id', StatisticController.getMyTotalGroupViewsCount)
app.get('/countUser', StatisticController.getCountUser)
app.get('/usersCountByRole', StatisticController.getUsersCountByRole)
app.listen(process.env.PORT || 5000, (err) => {
  if (err){
    return console.log(err)
  }
  console.log("Server OK")
})

```

Лістинг AddGroup.jsx:

```

import React from 'react';
import TextField from '@mui/material/TextField';
import Paper from '@mui/material/Paper';
import Button from '@mui/material/Button';
import SimpleMDE from 'react-simplemde-editor';
import CreateIcon from '@mui/icons-material/Create';
import UploadFileIcon from '@mui/icons-material/UploadFile';
import Stack from '@mui/material/Stack';
import HelpIcon from '@mui/icons-material/Help';
import IconButton from '@mui/material/IconButton';
import { Link } from 'react-router-dom';

import 'easymde/dist/easymde.min.css';
import styles from './AddGroup.module.scss';

```

					ПІЗ.КР.Б-121-24-ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import {selectrIsAuth} from '../redux/slices/auth';
import { useSelector, useDispatch } from 'react-redux';
import {Navigate, useNavigate, useParams} from "react-router-dom"
import axios from '../..../axios';
export const AddGroup = () => {
  //const imageUrl = '';
  const {id} = useParams()
  const navigate = useNavigate()
  const isAuth = useSelector(selectrIsAuth)
  const userData = useSelector((state)=> state.auth.data)
  const [isLoading, setLoading] = React.useState(false);
  const [title, setTitle] = React.useState('');
  const [imageUrl, setImageUrl] = React.useState('');
  const inputFileRef = React.useRef(null)
  const isEditing =Boolean(id)
  const handleChangeFile = async (event) => {
    try {
      const formData = new FormData()
      const fileName = event.target.files[0]
      formData.append("image", fileName)
      const {data} = await axios.post("/upload",formData)
      setImageUrl(data.url)
    } catch (error) {
      console.warn(error)
      alert("Помилка при загрузці фото")
    }
  };
  const onClickRemoveImage = () => {
    setImageUrl("")
  };
  const onSubmit = async () =>{
    try {
      setLoading(true)
      const fields = {
        title,
        imageUrl,
      }
      const {data} = isEditing ? await axios.patch(`/groups/${id}`, fields) : await axios.post("/groups",
fields)
      const _id = isEditing ? id : data._id
      navigate(`/groups/${_id}`)
    } catch (error) {
      console.warn(error)
      console.log(error)
      alert("Помилка при створені групи")
    }
  }
  React.useEffect(()=>{
    if (id){
      axios.get(`/groups/${id}`).then(({data}) =>{
        setTitle(data.title)
        setImageUrl(data.imageUrl)
      })
    }
  },[])

  if (userData?.role !== 1 && userData?.role !== 3) {
    return <Navigate to="/" />;
  }
  return (
    <Paper style={{ padding: 30 }}>

```

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						74
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    <Button onClick={() => inputFileRef.current.click()} endIcon={<UploadFileIcon />} style={{ margin:
10 }} variant="outlined" size="large">
      Завантажити фото
    </Button>
    <input ref={inputFileRef} type="file" onChange={handleChangeFile} hidden />
    {imageUrl && (
      <>
        <Button variant="contained" color="error" onClick={onClickRemoveImage}>
          Видалити
        </Button>
        <img className={styles.image} src={`http://localhost:5000${imageUrl}`} alt="Uploaded" />
      </>
    )}

    <TextField
      classes={{ root: styles.title }}
      variant="standard"
      placeholder="Назва групи"
      value={title}
      onChange={(e) => setTitle(e.target.value)}
      fullWidth
    />
    <div className={styles.buttons}>
      <Button onClick={onSubmit} endIcon={<CreateIcon />} size="large" variant="contained">
        {isEditing ? "Зберегти": "Опублікувати"}
      </Button>
      <a href="/groups">
        <Button size="large">Відмінити</Button>
      </a>
      <Link to="/privacy-policy">
        <IconButton color="secondary">
          <HelpIcon />
        </IconButton>
      </Link>
    </div>
  </Paper>
);
};

```

Лістинг groups.js:

```

import {createSlice, createAsyncThunk} from "@reduxjs/toolkit"
import axios from "../../axios"

export const fetchGroups = createAsyncThunk("groups/fetchGroups", async()=>{
  const {data} = await axios.get("/groups")
  return data
})
export const fetchSearchGroups = createAsyncThunk("groups/fetchSearchGroups", async(searchQuery)=>{
  const {data} = await axios.get(`/groups/search?title=${encodeURIComponent(searchQuery)}`)
  return data
})
export const fetchRemoveGroup = createAsyncThunk("groups/fetchRemoveGroup", async(id)=>{
  await axios.delete(`/groups/${id}`)
})
const initialState = {
  groups: {
    items: [],
    status: "loading",
  },
}
const groupsSlice = createSlice({
  name: "groups",
  initialState,

```

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    reducers: {},
    extraReducers: {
      [fetchGroups.pending]: (state)=>{
        state.groups.items = []
        state.groups.status = "loading"
      },
      [fetchGroups.fulfilled]: (state,action)=>{
        state.groups.items = action.payload
        state.groups.status = "loaded"
      },
      [fetchGroups.rejected]: (state)=>{
        state.groups.items = []
        state.groups.status = "error"
      },
      [fetchSearchGroups.pending]: (state)=>{
        state.groups.items = []
        state.groups.status = "loading"
      },
      [fetchSearchGroups.fulfilled]: (state,action)=>{
        state.groups.items = action.payload
        state.groups.status = "loaded"
      },
      [fetchSearchGroups.rejected]: (state)=>{
        state.groups.items = []
        state.groups.status = "error"
      },
      [fetchRemoveGroup.pending]: (state, action)=>{
        state.groups.items = state.groups.items.filter(obj => obj._id !== action.meta.arg)
      },
    }
  })
  export const groupsReducer = groupsSlice.reducer

```

					ПЗ.КР.Б-121-24-ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		