

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 «Веб-технології»)

на тему:

«Платформа для створення та публікації мемів»

Виконала студентка групи ВТ-21-1
РАДКІВСЬКА Аліна Валеріївна

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ГРОМСЬКИЙ Олександр Олександрович

Житомир – 2025

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **РАДКІВСЬКА Аліна Валеріївна**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Платформа для створення та публікації мемів»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **розробка та впровадження автоматизованої системи для взаємодії з цифровим контентом за допомогою сучасних інформаційних технологій.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р. С.	17.02.2025	17.02.2025
2	Савіцький Р. С.	19.02.2025	19.02.2025
3	Савіцький Р. С.	21.02.2025	21.02.2025

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра складається зі вступу, трьох розділів, висновку та списку використаних джерел.

Текстова частина викладена на 62 сторінках друкованого тексту. Список використаних джерел містить 27 найменувань і займає 3 сторінки. В роботі наведено 44 рисунків та 16 таблиць. Загальний обсяг роботи – 74 сторінок.

У роботі поставлені основні завдання на розробку системи для створення та публікації мемів. Система включає модулі публікацій, користувачі, підписок, коментарів та вподобань. Також було проведено порівняльний аналіз аналогів. Було обрано клієнт-серверну архітектуру, яка працює у зв'язці з сучасними підходами до веброзробки. Наведені основні сценарії використання системи, логіка взаємодії між модулями, діаграми даних та алгоритми роботи ключових модулів. Спроектowana база даних з використанням PostgreSQL. Клієнтська частина розроблена за допомогою фреймворку Next.js із використанням TanStack Query, а серверна частина за допомогою фреймворку NestJS. Для збереження зображень використано UploadThing.

Ключові слова: КОРИСТУВАЧ, МЕМ, ПЛАТФОРМА, ПУБЛІКАЦІЯ, РЕДАКТОР.

					Житомирська політехніка. 25.121.21.000 - ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата	Пояснювальна записка до випускної кваліфікаційної роботи бакалавра		
Розроб.	Радківська А.В.						
Керівник	Савіцький Р.С.						
Рецензент	Громський О.О.						
Зав. каф.	Вакалюк Т.А.						
					Літ.	Арк.	Аркушів
						3	74
					ФІКТ Гр. ВТ-21-1		

ABSTRACT

The explanatory note to the bachelor's final qualification work consists of an introduction, three sections, a conclusion and a list of used sources.

The text part is laid out on 62 pages of the printed text. The list of used sources contains 27 names and occupies 3 pages. 44 images and 16 tables are given in the work. The total volume of the work is 74 pages.

The main tasks for the development of a system for creating and publishing memes are set in the work. The system includes modules for posts, users, follows, comments, and likes. A comparative analysis of analogues was also conducted. A client-server architecture was chosen, which works in conjunction with modern approaches to web development. The main scenarios of using the system, the logic of interaction between modules, data diagrams and algorithms of operation of key modules are given. Designed database using PostgreSQL. The client part is developed using the Next.js framework using TanStack Query, and the server part using the NestJS framework. UploadThing is used to save images.

Keywords: EDITOR, MEME, PLATFORM, PUBLICATION, USER.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ	9
1.1. Визначення та постановка задачі	9
1.2. Аналіз існуючого програмного забезпечення	10
1.3. Визначення архітектури програмного забезпечення	15
1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення	19
Висновки до першого розділу	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ТА ПУБЛІКАЦІЇ МЕМІВ	22
2.1. Визначення варіантів використання об'єктно-орієнтованої структури системи	22
2.2. Розробка бази даних системи	26
2.3. Проєктування та реалізація алгоритмів роботи платформи	29
2.4. Реалізація функціоналу платформи	33
Висновки до другого розділу	39
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ	41
3.1. Розгортання платформи	41
3.2. Структура інтерфейсу та користувацька взаємодія	42
3.3. Тестування роботи програмного продукту	52
Висновки до третього розділу	56
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
ДОДАТКИ	63
Додаток А	64
Додаток Б	69
Додаток В	70

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

ШІ – штучний інтелект

API – Application Programming Interface, дослівно: інтерфейс прикладного програмування

DDD – Domain-Driven Design, дослівно: предметно-орієнтоване проектування

HTTP – Hypertext Transfer Protocol, дослівно: протокол передачі гіпертекстових документів

IoC – Inversion of Control, дослівно: інверсія керування

JSON – JavaScript Object Notation

JWT – JSON Web Token

OAuth – Open Authorization, дослівно: відкрита авторизація

ORM – Object-Relational Mapping, дослівно: об'єктно-реляційна проєкція

SSG – Static Site Generation, дослівно: статична генерація сторінок

SSR – Server-Side Rendering, дослівно: серверна генерація сторінок

TS – TypeScript

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Платформа для створення та публікації мемів орієнтована на ефективну взаємодію між користувачами і публікаціями, а також має забезпечити надійну і швидку роботу для великої кількості користувачів.

Актуальність теми зумовлена зростанням популярності цифрового контенту, зокрема мемів, які стали важливим елементом онлайн-комунікації. Користувачі мають необхідність ефективної взаємодії з іншими користувачами, швидкого обміну контентом, а також наявність зручного редактора для обробки зображень і можливостей генерації контенту за допомогою штучного інтелекту. Така система сприятиме підвищенню зручності користування, безпеки та ефективності контентного середовища.

Зокрема, меми відіграють дедалі важливішу роль у цифровій культурі, політичній сатирі, маркетингу та соціальній активності. Вони є засобом самовираження, формування громадської думки та навіть інструментом мобілізації спільнот. Розробка платформи, яка спрощує створення й поширення такого контенту, відповідає потребам сучасного користувача та сприяє розвитку культури інтернету.

Крім того, наявні сервіси часто не поєднують у собі всі необхідні функціональні можливості – зручний редактор, соціальну взаємодію, AI-генерацію та персоналізовану стрічку. Створення комплексного рішення з урахуванням новітніх технологій, таких як машинне навчання, адаптивний інтерфейс та хмарна інфраструктура, дозволить запропонувати конкурентоспроможний продукт на ринку цифрових платформ.

Метою роботи є проектування та реалізація платформи для створення, пошуку та обговорення мемів, що забезпечить швидку взаємодію між користувачами, ефективні інструменти для роботи з фото матеріалами. Система покликана забезпечити надійну роботу в умовах високого навантаження, підвищити зручність користування та підтримати розвиток культури інтернету.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

Завданням випускної кваліфікаційної роботи бакалавра є виконання таких задач:

- постановка основних задач системи та узагальнення її структури;
- аналіз аналогів;
- побудова архітектури, що буде задовольняти всі вимоги системи;
- вибір та обґрунтування засобів реалізації системи;
- визначити варіанти використання платформи;
- розробка функціоналу та реалізація алгоритмів платформи;
- розробка UI/UX для клієнтської частини платформи;
- розгортання платформи на віддаленому сервері;
- мануальне та юніт тестування розробленого функціоналу.

Об’єкт дослідження: інформаційні технології для автоматизації взаємодії між користувачами та публікаціями, яка включатиме механізми створення та поширення мемів, а також взаємодії з контентом (коментарі, вподобання). Досліджується можливість оптимізації процесів управління платформою для забезпечення високої продуктивності та безперебійної роботи при великій кількості користувачів.

Предмет дослідження: процес розробки та впровадження автоматизованої системи для взаємодії з цифровим контентом.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тезу на тему «Підбір рекомендацій публікацій з використанням content-based та collaborative-based фільтрації» [1].

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Визначення та постановка задачі

Основною метою роботи є розробка платформи для створення та публікації мемів, яка забезпечить користувачам можливість швидко та зручно генерувати контент, взаємодіяти з іншими користувачами. Система спрямована на підвищення якості користувацького досвіду через зручний редактор мемів, систему підписок вподобань та коментарів. Для реалізації цієї мети необхідно виконати такі завдання:

1. Розробка архітектури системи:

- визначення загальної концепції платформи, її основних функцій та ролей користувачів (Гість, Користувач);
- побудова архітектури клієнт-серверної взаємодії на основі сучасних вебтехнологій;
- проектування та створення бази даних з використанням PostgreSQL.

2. Розробка функціональних модулів backend частини системи:

- модуль користувачів;
- модуль авторизації за допомогою JWT-токенів зі змогою авторизуватися за допомогою Google;
- модуль публікації мемів та управління контентом;
- модуль підписки на акаунти користувачів;
- модуль вподобань;
- модуль коментування публікацій (включаючи можливість додавання зображення у коментарі);
- модуль роботи з і збереженням зображень у віддаленому сховищі.

3. Розробка додаткового віддаленого сервісу, а саме генерації зображень за описом за допомогою штучного інтелекту.

4. Тестування та впровадження:

- проведення ручного тестування вебзастосунку;
- проведення юніт-тестування серверної частини програмного продукту;

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

- оптимізація продуктивності для забезпечення швидкодії;
- аналіз користувацького досвіду та внесення необхідних покращень.

Результатом реалізації поставленого завдання стане багатофункціональна вебплатформа, що дозволить користувачам створювати, публікувати та обговорювати меми, використовуючи технології фронтенду (Next.js, TanStack Query, Konva, Zustand) та бекенду (NestJS). Система забезпечить зручну взаємодію, ефективне управління контентом, що сприятиме створенню активної спільноти навколо платформи.

1.2. Аналіз існуючого програмного забезпечення

Існує багато платформ для створення, поширення та обговорення контенту, включаючи меми. Завдяки розвитку соціальних мереж та онлайн-спільнот, користувачі отримали можливість не лише переглядати, а й активно взаємодіяти з контентом, залишаючи коментарі, вподобання та поширюючи матеріали.

Аналіз аналогів є надзвичайно корисним інструментом для визначення сильних і слабких сторін уже існуючих сервісів. Це дозволяє краще сформулювати функціональні вимоги до власного програмного продукту, враховуючи реальні потреби цільової аудиторії та ефективні механізми взаємодії.

Для порівняння розглянемо популярні сервіси: Instagram [2], Reddit [3] та 9GAG [4]. Ці платформи відрізняються своєю цільовою аудиторією, алгоритмами ранжування контенту, можливостями монетизації та форматом взаємодії між користувачами.

Розглянемо кожен аналог ближче. Розпочнімо із соцмережі Instagram (рис. 1.1). Instagram є найбільш універсальною платформою, яка дозволяє користувачам створювати візуальний контент, застосовувати фільтри та взаємодіяти через підписки, лайки та коментарі. Проте він не має спеціалізованих функцій для створення мемів.

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

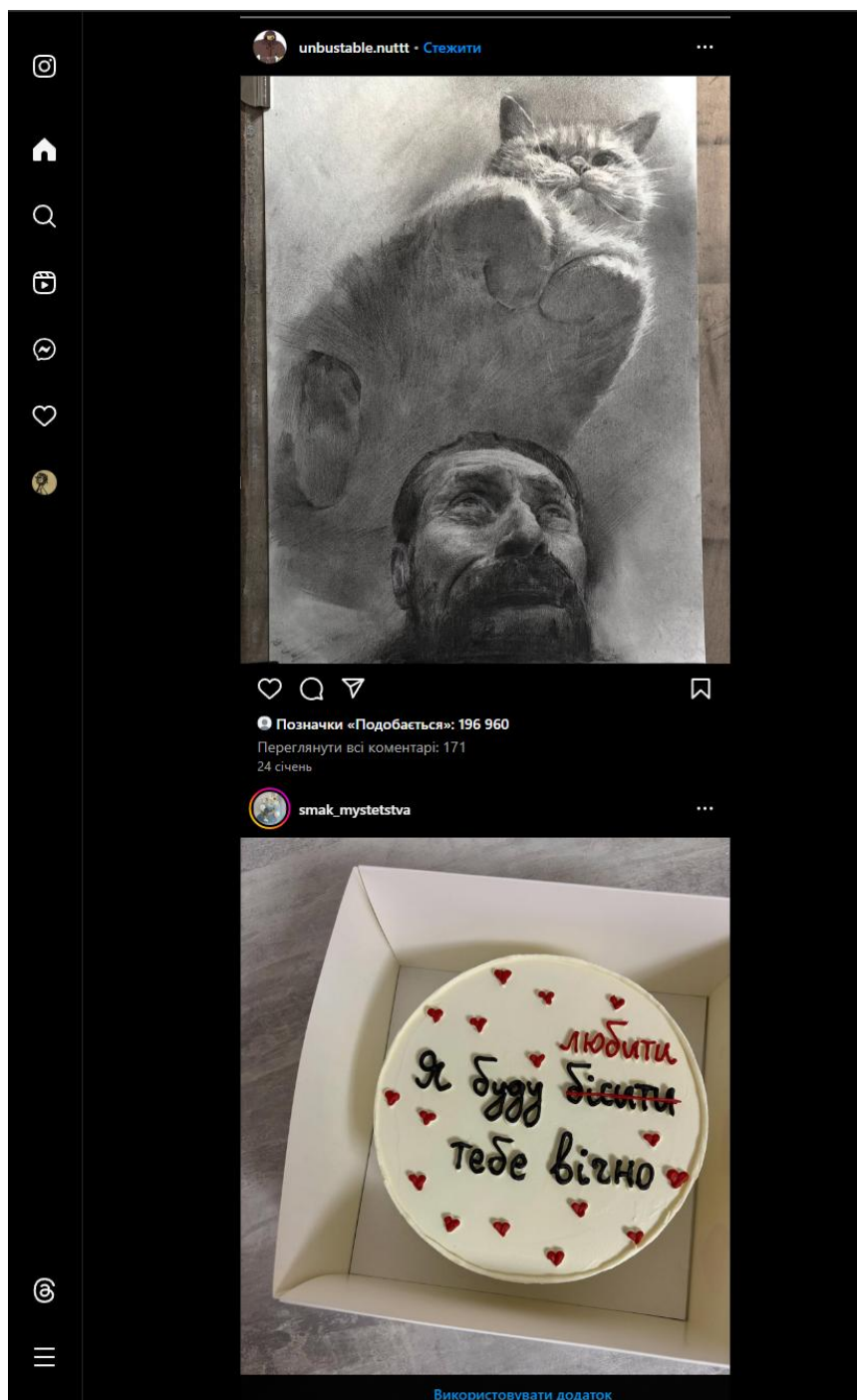


Рисунок 1.1. Публікації у соцмережі Instagram

Instagram вирізняється своєю універсальністю та приваблює широку аудиторію завдяки можливості створювати яскравий візуальний контент, який легко редагується за допомогою фільтрів. Платформа забезпечує активну взаємодію між користувачами – через підписки, вподобайки та коментарі. Також відкриває можливості для просування через рекламу та рекомендаційні алгоритми. Однак, попри свою популярність, Instagram має дещо інше призначення для

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

користування – поширення фотографій. Його алгоритми також можуть впливати на охоплення публікацій, що ускладнює органічне поширення певного контенту, особливо нішевого або сатиричного спрямування.

Reddit, на відміну від Instagram, побудований на принципі спільнот за інтересами, де користувачі голосують за контент, що дає можливість організованого обговорення, проте на платформі немає вбудованих інструментів для створення мемів (рис. 1.2).

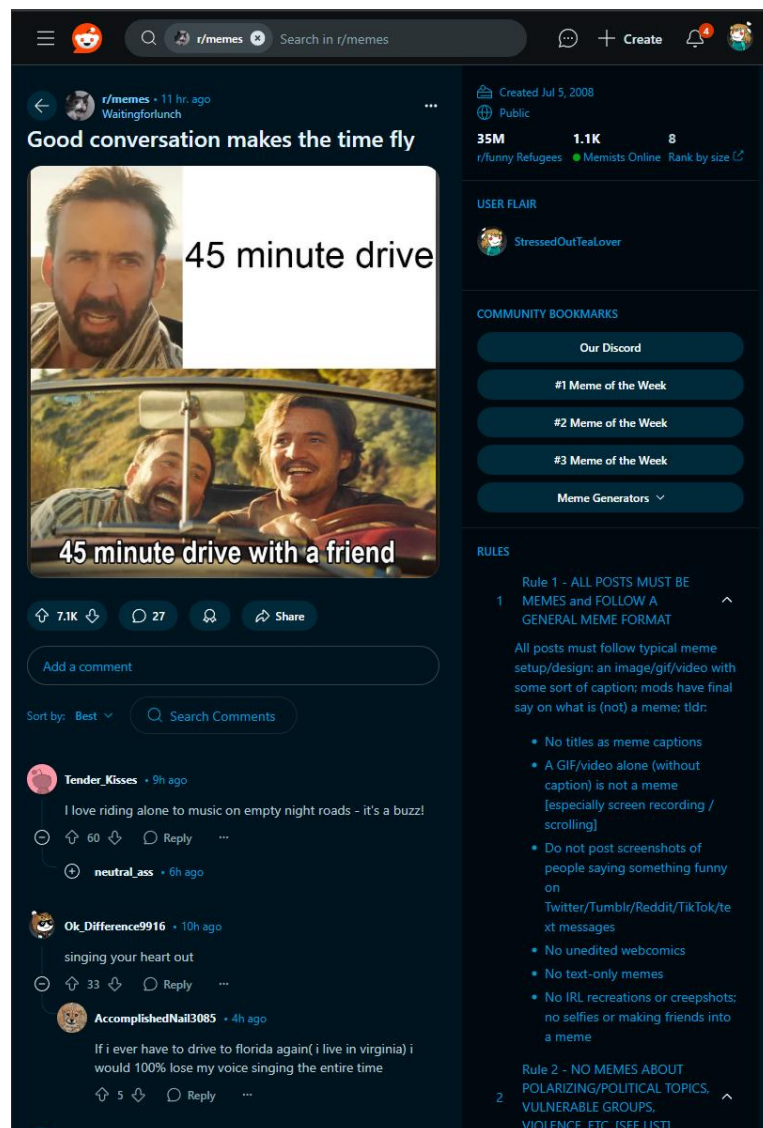


Рисунок 1.2. Обговорення публікації на платформі Reddit

Reddit вирізняється унікальною структурою, що базується на тематичних спільнотах, де кожен користувач може знайти контент за власними інтересами. Такий формат сприяє глибшим дискусіям, а система голосування дозволяє спільноті самостійно визначати релевантність та якість публікацій. Крім того,

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

механізми модерації в більшості спільнот дають змогу підтримувати порядок та специфіку кожного підрозділу платформи. Водночас Reddit не передбачає інструментів для безпосереднього створення візуального гумористичного контенту, як-от меми. А ще, через відкритість і різноманітність аудиторії, у певних спільнотах можуть виникати конфлікти, що впливають на комфорт користувачів.

9GAG спеціалізується саме на мемах та розважальному контенті (рис. 1.3). Взаємодія здійснюється через лайки, коментарі та репости. Однак платформа більше орієнтована на споживання контенту, ніж на його створення, оскільки вбудованого редактора мемів немає.

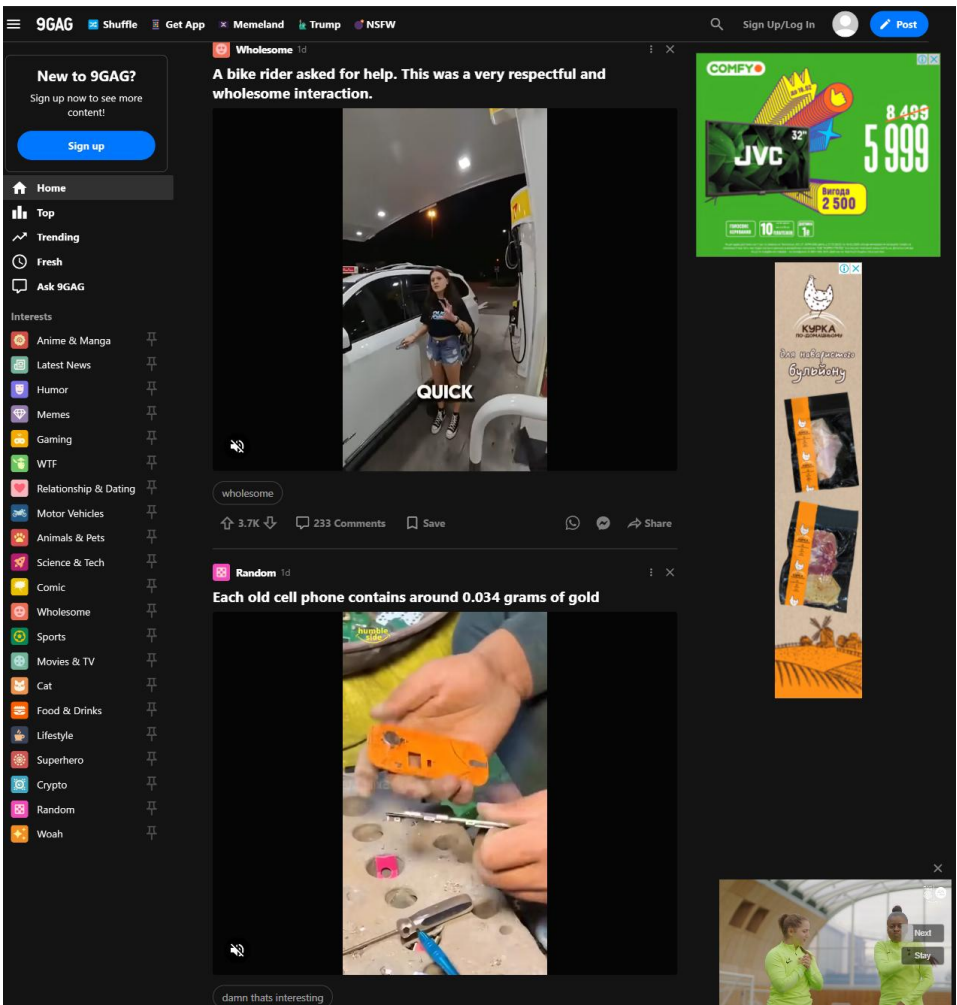


Рисунок 1.3. Головна сторінка 9GAG

9GAG є платформою, яка цілком зосереджена на розважальному контенті, особливо мемах, що робить її привабливою для користувачів, які шукають легкий гумор і візуальні жарти. Завдяки простому інтерфейсу, перегляд контенту не потребує зусиль, а популярні публікації швидко набирають оберти завдяки системі

лайків, коментарів і репостів. Це середовище стимулює вірусне поширення жартів, однак воно менш сприятливе для активної творчості, адже платформа не пропонує вбудованих інструментів для створення власних мемів. Основна аудиторія споживає контент радше пасивно, ніж активно долучається до його продукування. Крім того, значна кількість реклами може відволікати або знижувати загальне враження від користування сайтом.

Порівняння характеристик цих сервісів можна переглянути в порівняльній таблиці 1.1.

Таблиця 1.1

Таблиця порівняння аналогів

Характеристика	Instagram	Reddit	9GAG
Основний контент	Фото, відео	Текст, зображення, відео	Мем-картинки, GIF, відео
Формат взаємодії	Підписки, лайки, коментарі	Голосування, коментарі, підписки на субредіти	Лайки, коментарі, репости
Редактор контенту	Фільтри, ефекти	Відсутній	Відсутній
Цільова аудиторія	Загальна, блогери, бренди	Спільноти за інтересами	Любителі мемів, гумору

Насамкінець, порівняймо функціональні вимоги аналогів у таблиці 1.2.

Таблиця 1.2

Функціональні можливості

Функціонал	Instagram	Reddit	9GAG	Розроблювана система
Створення мемів	ні	ні	ні	так

Публікація контенту	так	так	так	так
Взаємодія з публікаціями	так	так	так	так
Взаємодія з іншими користувачами	так	так	ні	так
Пошук публікацій	так	так	так	так
Генерація мемів за допомогою ІІІ	ні	ні	ні	так

Підсумовуючи, Instagram добре підходить для публікації та обміну контентом – там зручно лайкати, коментувати й шукати пости, а також спілкуватися з іншими, але створити мем чи згенерувати його за допомогою ІІІ не вийде. Reddit сильний у спілкуванні та пошуку по темах, там легко обговорювати контент і взаємодіяти з іншими, проте редактора мемів або ІІІ-функцій теж немає. 9GAG має простий і зрозумілий доступ до великої кількості мемів, з якими можна взаємодіяти, але не дозволяє створювати їх самостійно, мало можливостей для спілкування і повністю відсутня підтримка ІІІ.

Аналіз аналогічних платформ допоміг визначити сильні та слабкі сторони існуючих рішень. Виявлено, що жодна з проаналізованих платформ не пропонує повноцінного редактора мемів а функції взаємодії з контентом (підписки, лайки, коментарі) мають свої обмеження. Таким чином, аналіз аналогів допоміг визначити оптимальний набір функцій та розставити пріоритети у розробці платформи.

1.3. Визначення архітектури програмного забезпечення

Для розробки платформи для створення та публікації мемів було обрано клієнт-серверну архітектуру, у якій клієнтська частина взаємодіє із сервером для

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

отримання, обробки та збереження даних (див. рис.1.4). Такий підхід забезпечує масштабованість, модульність та можливість розширення функціональності.

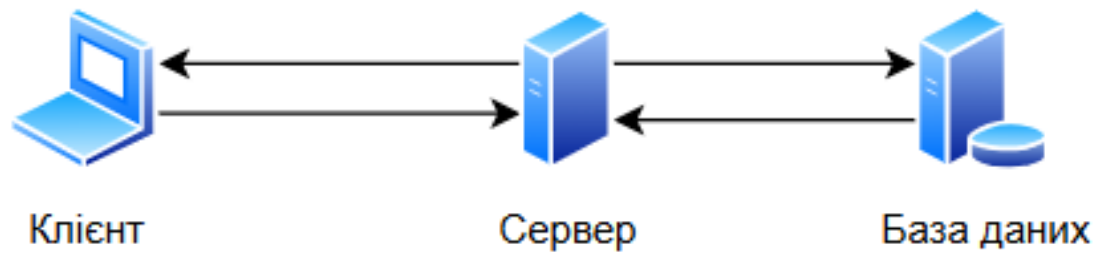


Рисунок 1.4. Схематичне зображення клієнтсько-серверного архітектури

Для взаємодії ШІ з клієнтською частиною також було створено простий серверний елемент, з яким взаємодіє клієнтська частина програмного продукту. Узагальнюючи архітектура платформи має такий вигляд як на рисунку 1.5.

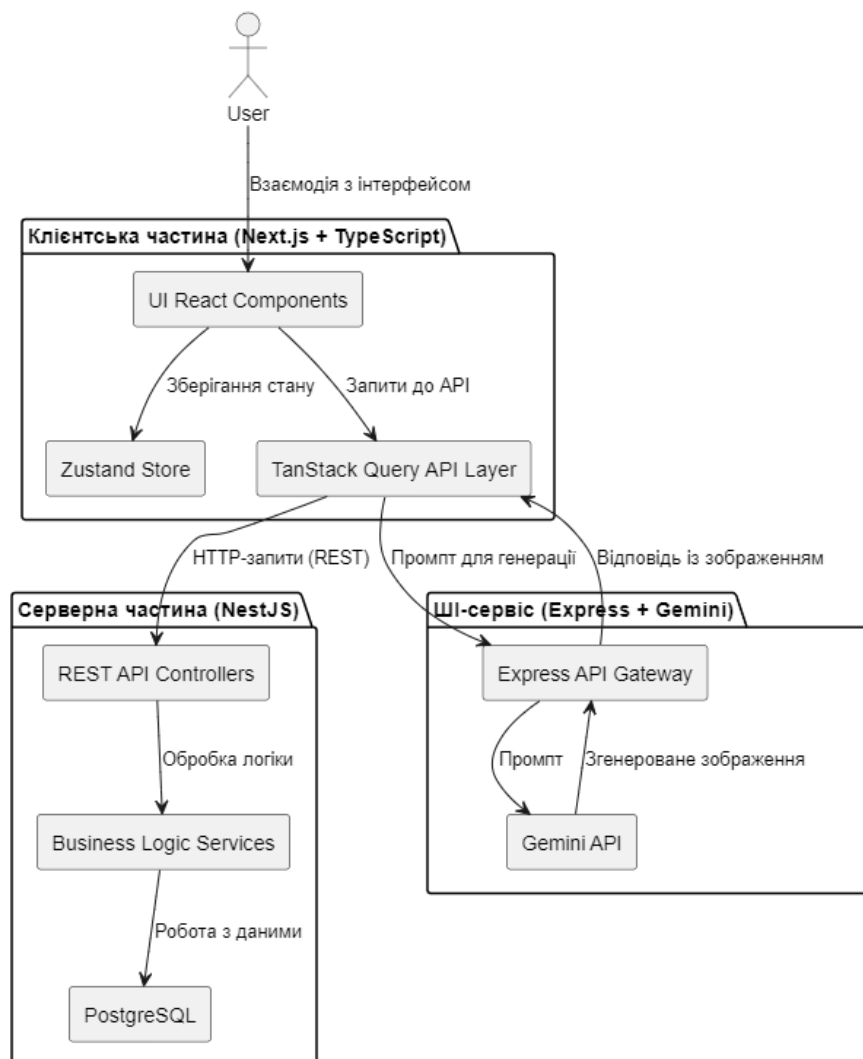


Рисунок 1.5. Діаграма архітектури вебплатформи.

Клієнтська частина реалізована за допомогою Next.js, що відповідає за інтерфейс користувача, взаємодію з редактором мемів та відправку запитів до сервера [5]. Для створення сучасного, адаптивного та уніфікованого інтерфейсу було використано компонентну бібліотеку shadcn, яка поєднує гнучкість Tailwind CSS із попередньо налаштованими, зручними у використанні UI-компонентами. Завдяки цьому вдасться значно пришвидшити розробку, забезпечити візуальну цілісність застосунку та підтримку темної/світлої теми.

У свою чергу, сервер, створений на базі NestJS [6], обробляє запити, реалізує бізнес-логіку та зберігає дані у базі даних PostgreSQL. NestJS дотримується шаблону модульної архітектури, який значною мірою натхненний принципами модульного моноліту та предметно-орієнтованого проектування (DDD) [7]. Він також включає елементи впровадження залежностей і інверсії управління (IoC) через декоратори та рефлексію метаданих. В архітектурі використано такі патерни Modularization та Dependency Injection. Перший патерн передбачає розділення застосунків на окремі модулі (редактор мемів, система коментарів, вподобань, тощо). Це дозволяє легше розширювати функціональність та підтримувати кодову базу. Другий патерн впроваджений для управління залежностями між компонентами серверної частини. Це дозволяє гнучко налаштовувати сервіси, підвищує тестованість та забезпечує слабе зв'язування між модулями.

Серверна частина реалізує менеджмент користувачів та публікацій, систему підписок та сповіщень, взаємодіє з клієнтською частиною через RESTFull API [8].

Обрана архітектура забезпечує гнучкість, масштабованість та можливість подальшого розширення функціоналу без значних змін у базовій структурі застосунку. Завдяки модульному підходу кожен компонент системи розробляється окремо, що полегшує тестування, обслуговування та внесення змін без ризику порушення роботи всього застосунку. Масштабованість досягається завдяки можливості додавання нових модулів або оптимізації існуючих без необхідності перероблювати всю систему. Крім того, використання базових патернів NestJS дозволяє легко інтегрувати нові сервіси та адаптувати застосунок до вимог бізнесу.

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

Таким чином, архітектура сприяє ефективному розвитку проєкту, дозволяючи розширювати функціональність поступово, без значного збільшення складності коду та збереження високої продуктивності.

Клієнтська частина реалізована за допомогою Next.js з підтримкою TypeScript, що забезпечує високий рівень типізації, покращену підтримку інструментів розробки та зменшує кількість помилок на етапі компіляції [9]. Використання Next.js дозволяє поєднувати серверну та клієнтську логіку, забезпечуючи швидкий рендеринг сторінок (через SSR і SSG) та покращену індексацію контенту пошуковими системами.

Для ефективного управління асинхронними запитами до API використовується TanStack Query (раніше React Query), який забезпечує кешування, повторні запити, автоматичне оновлення даних та глобальний контроль за статусом завантаження [10]. Це суттєво покращує досвід користувача завдяки швидкій та безпечній роботі з даними.

Стан інтерфейсу керується за допомогою Zustand – легковагового менеджера стану, який забезпечує просту декларативну синтаксисну модель, хорошу масштабованість та високу продуктивність [11]. Його використання дозволяє уникати надмірної складності, характерної для більш важких бібліотек, таких як Redux. За допомогою нього набагато зручніше керувати великими обсягами даних без зайвого нагромадження властивостей клієнтських компонентів.

Особливу увагу приділено зручності користувача під час створення мемів. У застосунку реалізовано зручний редактор зображень за допомогою бібліотеки Konva, що дозволяє додавати текст, зображення та емоції, змінювати розміри та положення частин зображення на полотні [12]. Для генерації зображень за текстовими описами додано інтеграцію зі стороннім ШІ-сервісом. Для цього був створений окремий мікросервіс на базі Express, який здійснює взаємодію з Gemini API [13]. Він приймає запити з клієнта, передає їх до ШІ, обробляє відповіді та надсилає результати у вигляді зображень назад до фронтенду.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Такий підхід до реалізації клієнтської частини дозволяє створити динамічний, реактивний та зручний для користувача інтерфейс з високим рівнем інтерактивності та широкими можливостями персоналізації контенту.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до програмного забезпечення

У процесі розробки системи було використано кілька ключових інструментів, що забезпечують ефективність, продуктивність та зручність роботи. Одним із головних інструментів стало середовище розробки Visual Studio Code. Це легке та водночас потужне середовище, яке підтримує широкий набір розширень для різних мов програмування та технологій. Завдяки вбудованим можливостям підсвічування синтаксису, автоматичного доповнення коду та підтримки систем контролю версій, Visual Studio Code дозволяє розробникам працювати швидше та ефективніше.

Для тестування API-запитів було обрано Postman. Цей інструмент значно полегшує розробку та налагодження API, дозволяючи створювати, тестувати та аналізувати запити до сервера в зручному графічному інтерфейсі [14]. Використання Postman допомагає автоматизувати перевірку роботи API, що сприяє швидкому виявленню помилок та покращенню якості сервісу.

Для роботи з базами даних були проаналізовані найпопулярніші варіанти: MySQL, PostgreSQL та MongoDB. Порівняння цих баз даних можна побачити у таблиці 1.3. Обрано PostgreSQL, що є потужною реляційною системою керування базами даних. Вона забезпечує високу продуктивність, підтримує складні запити, транзакції та механізми безпеки. Для зручного керування базою даних та її локального розгортання використовується Docker, що дозволяє створювати ізольовані контейнери, полегшуючи управління середовищем розробки та підготовку до глобального розгортання [15]. Використання контейнеризації гарантує стабільність роботи та однакові умови для всіх середовищ.

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

Порівняння баз даних

Система	Продуктивність	Безпека	Масштабованість
PostgreSQL	Висока	Висока	Гнучка
MySQL	Середня	Середня	Середня
MongoDB	Висока	Середня	Висока

З метою забезпечення чистоти, читабельності та узгодженості коду було інтегровано ESLint та Prettier. ESLint допомагає виявляти помилки та неточності у JavaScript/TypeScript-кодi, а Prettier автоматично форматує код згідно з визначеними стилістичними правилами. Це сприяє покращенню ментального стану будь-якого програміста та дозволяє зосередитись на логіці, а не на форматуванні коду.

Для керування версіями проєкту використовується Git у поєднанні з хмарним середовищем git.ztu.edu.ua. Це дозволяє ефективно відстежувати зміни, створювати гілки для окремих функцій або виправлень, а також швидко повертатися до стабільних версій за потреби.

Вибір кожного інструмента обумовлений його популярністю, функціональністю та відповідністю поставленим вимогам. Використання сучасних рішень дозволяє досягти високої продуктивності, гнучкості та масштабованості проєкту.

Висновки до першого розділу

У першому розділі було проведено детальний аналіз існуючих аналогів, визначено архітектуру програмного забезпечення, обґрунтовано вибір інструментальних засобів і сформульовано основні завдання.

Аналіз аналогів дозволив визначити ключові функціональні вимоги до системи, зокрема інтеграцію спеціалізованого редактора мемів та генерацію зображень за допомогою ШІ.

Для реалізації проєкту було обрано клієнтно-серверну архітектуру, яка забезпечує масштабованість і гнучкість. Клієнтська частина створена на основі

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

Next.js, що дозволяє розробити зручний інтерфейс користувача, а серверна частина базується на NestJS, що забезпечує ефективну обробку запитів і взаємодію з базою даних PostgreSQL. Використання патернів модульної архітектури та впровадження залежностей значно спрощує підтримку та розвиток платформи. При виборі технологічного стеку враховувалися продуктивність, безпека та масштабованість.

Загалом, архітектура та вибрані технології забезпечують високу продуктивність, надійність і можливість подальшого розширення функціоналу платформи.

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ТА ПУБЛІКАЦІЇ МЕМІВ

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

Система є вебплатформою для створення, редагування, публікації та взаємодії з ними. Вона використовує архітектуру та логіку взаємодії основних об'єктів у вебдодатку для створення і публікації мемів. Система базується на стеку Next.js та TanStack Query для фронтенду, NestJS для бекенду, базі даних PostgreSQL, сервісі UploadThing для зберігання зображень та бібліотеці Konva, що є основою для редактора зображень. Основними функціями системи є додавання створення мемів, зокрема за допомогою штучного інтелекту, їх публікація, взаємодія користувачів через лайки, коментарі, підписки. У системі визначено дві основні ролі: Користувач і Гість (див. рис.2.1).



Рисунок 2.1. Діаграма варіантів використання

Користувачі можуть створювати меми (з редактором зображень та ШІ), коментувати, вподобувати публікації та підписуватися на інші акаунти. Також є зручний глобальний пошук публікацій, що враховує опис мемів, ключові слова та автора публікації. Натомість гість має обмежений функціонал, що дає можливість переглядати контент та функції платформи, зокрема користуватися редактором зображень та генерувати зображення за допомогою ШІ, але не дозволяє взаємодіяти з авторизованими користувачами та публікувати власний контент. Основні прецеденти описані в таблицях 2.1 – 2.8.

Таблиця 2.1

Опис прецеденту «Створення мему»

Назва	Створення мему
Опис	Користувач створює зображення через редактор, додає текст та інше оформлення
Виконавці	Користувач

Таблиця 2.2

Опис прецеденту «Публікація мему»

Назва	Створення мему
Опис	Користувач публікує створений мем для перегляду іншими користувачами
Виконавці	Користувач

Таблиця 2.3

Опис прецеденту «Пошук публікацій»

Назва	Пошук публікацій
Опис	Користувач шукає публікації за ключовими словами або іншими фільтрами
Виконавці	Користувач, Гість

Таблиця 2.4

Опис прецеденту «Генерація мему за описом»

Назва	Генерація мему за описом
Опис	Користувач вводить опис, система створює мем за допомогою ІІІ
Виконавці	Користувач

Таблиця 2.5

Опис прецеденту «Лайк публікації»

Назва	Лайк публікації
Опис	Користувач ставить вподобання під обраною публікацією
Виконавці	Користувач

Таблиця 2.6

Опис прецеденту «Перегляд стрічки публікацій»

Назва	Підписка на користувача
Опис	Користувач переглядає стрічку з новими мемами або мемами користувачів, на які підписаний користувач
Виконавці	Користувач, Гість

Таблиця 2.7

Опис прецеденту «Підписка на користувача»

Назва	Підписка на користувача
Опис	Користувач підписується на інший акаунт для відстеження нових публікацій
Виконавці	Користувач

Таблиця 2.8

Опис прецеденту «Перегляд профілю користувача»

Назва	Підписка на користувача
Опис	Користувач підписується на інший акаунт для відстеження нових публікацій
Виконавці	Користувач, Гість

Розглянемо діаграму класів, що розміщена на рисунку 2.2.

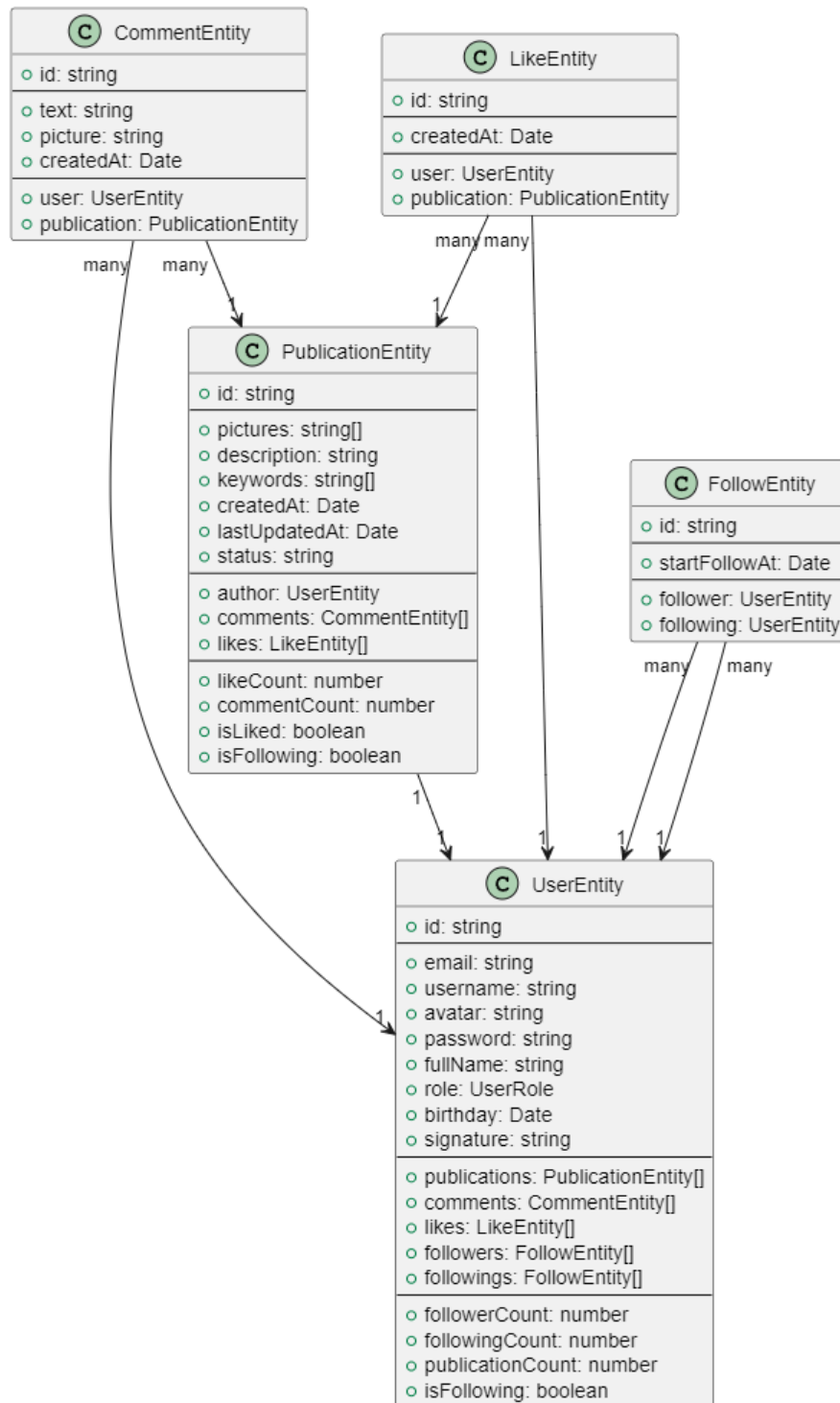


Рисунок 2.2. Діаграма класів

Діаграма класів описує основні сутності системи для платформи створення та публікації мемів, а також взаємозв'язки між ними. Користувач (User) є центральною фігурою: він може створювати публікації (Publication), залишати коментарі (Comment), ставити вподобання (Like), підписуватися на інших

користувачів (Follow). Кожна публікація містить масив зображень, опис, ключові слова та дані про її статус. Для зручного отримання потрібної інформації, не зберігаючи зайвих даних використані віртуальні поля. Наприклад, у користувача є поле, яке показує, скільки людей на нього підписані, або скільки він сам підписок має. Ці числа не зберігаються у таблиці напряму, а рахуються за даними про підписки. Так само є поле, яке показує, скільки активних публікацій має користувач. У публікації віртуальні поля відображають, скільки лайків і коментарів вона має, а також чи поставив лайк чи підписаний на автора поточний користувач.

Ця структура забезпечує гнучку та розширювану модель даних, необхідну для розвитку соціальної активності на платформі.

2.2. Розробка бази даних системи

Для зберігання інформації про користувачів та публікації системи використовується база даних PostgreSQL [16], з якою бекенд частина Платформи спілкується за допомогою ORM TypeORM [17]. Короткий опис можна побачити в таблицях 2.9 - 2.13.

Таблиця 2.9

Таблиця «User»

Поле	Тип	Nullable
id	string	ні
email	string	ні
username	string	ні
avatar	string	так
password	string	так
full_name	string	так
birthday	date	так
signature	string	так

Таблиця «Publication»

Поле	Тип	Nullable
id	string	ні
pictures	string[]	ні
description	string	ні
keywords	string[]	ні
author_id	FK → user	ні
status	string	ні
created_at	Date	ні

Таблиця 2.11

Таблиця «Comment»

Поле	Тип	Nullable
id	string	ні
text	string	ні
picture	string	так
user_id	FK → user	ні
publication_id	FK → publication	ні
created_at	Date	ні

Таблиця 2.12

Таблиця «Like»

Поле	Тип	Nullable
id	string	ні
user_id	FK → user	ні
publication_id	FK → publication	ні
created_at	Date	ні

Таблиця «Follow»

Поле	Тип	Nullable
id	string	ні
follower_id	FK → user	ні
following_id	FK → user	ні
created_at	Date	ні

Основною таблицею є таблиця користувачів, яка зберігає унікальну електронну пошту, зашифрований пароль, нікнейм, а також за потреби повне ім'я. Додатково враховуються дата народження, персональний підпис та аватар.

Публікації є ще однією ключовою сутністю, що містить зображення у вигляді масиву посилань, опис, ключові слова, дату створення та останнього оновлення. До кожної публікації прив'язано автора, її статус (наприклад, опублікована або прихована).

Коментарі дозволяють користувачам взаємодіяти з контентом, маючи текстовий вміст, за потреби зображення, і прив'язку до публікації та користувача, що його залишив. Також зберігається дата створення коментаря.

Функціональність вподобань реалізується через окрему таблицю, яка фіксує зв'язок між користувачем, що вподобав публікацію, та самою публікацією разом із часом взаємодії.

Соціальний аспект системи реалізовано через таблицю підписок, у якій зазначено, хто саме підписався на кого та коли розпочалося відстеження. Це дозволяє будувати мережу зв'язків між користувачами.

Загальна структура бази даних спрямована на забезпечення стабільної та безпечної взаємодії користувачів із платформою, надаючи можливість ефективно працювати з контентом та його творцями, стимулювати соціальну активність і формувати прозоре інформаційне середовище.

2.3. Проектування та реалізація алгоритмів роботи платформи

Для більшості функцій на платформі, наприклад створення мемів, публікації, коментування і тому подібне, користувачам потрібно увійти на вебсайт. Діаграма активності на рис. 2.3 показує процес авторизації та реєстрації в системі, від введення логіну користувача та пароля, а також реєстраційної форми, до успішного входу в систему.

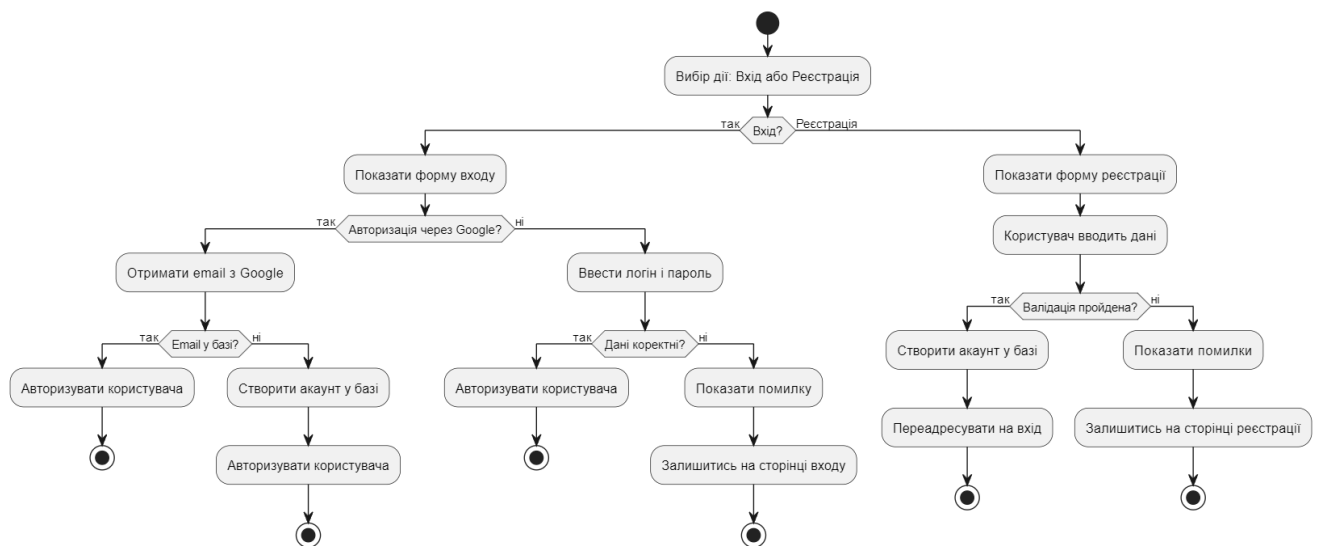


Рисунок 2.3. Діаграма активності для авторизації

Під час входу в систему користувачеві буде виведено сторінку з формою для входу, де він має ввести свої облікові дані. Якщо введені дані є неправильними, то користувач залишиться на сторінці для входу, для повторного введення свого логіну та паролю. Якщо ж користувач ввів коректні дані, то він отримує доступ до функціоналу системи відповідно до його ролі. Варто згадати, що користувач має попередньо існувати в системі, тому перед авторизацією користувач має зареєструватися. Аналогічно до авторизації, під час реєстрації має відобразитися реєстраційна форма, де користувач має заповнити всі обов'язкові поля так, щоб було пройдено валідацію. Після валідації даних на клієнті, вони відправляються на сервер і там також відбувається перевірка на правильність даних. Також є можливість спростити процес авторизації за допомогою OAuth та Google пошти [18]. Користувач має змогу обрати «Авторизуватись через Google» та виконати вхід

в свій профіль. Якщо такого не існує, він автоматично створюється використовуючи персональні дані Google акаунту.

На рис. 2.4-2.6 зображені діаграми активності для процесу створення та публікації мему, що містить 2 доріжки. Доріжки використовуються для групування дій, які виконуються одним актором. У даній діаграмі першу доріжку представляє Користувач, а другу – Система.

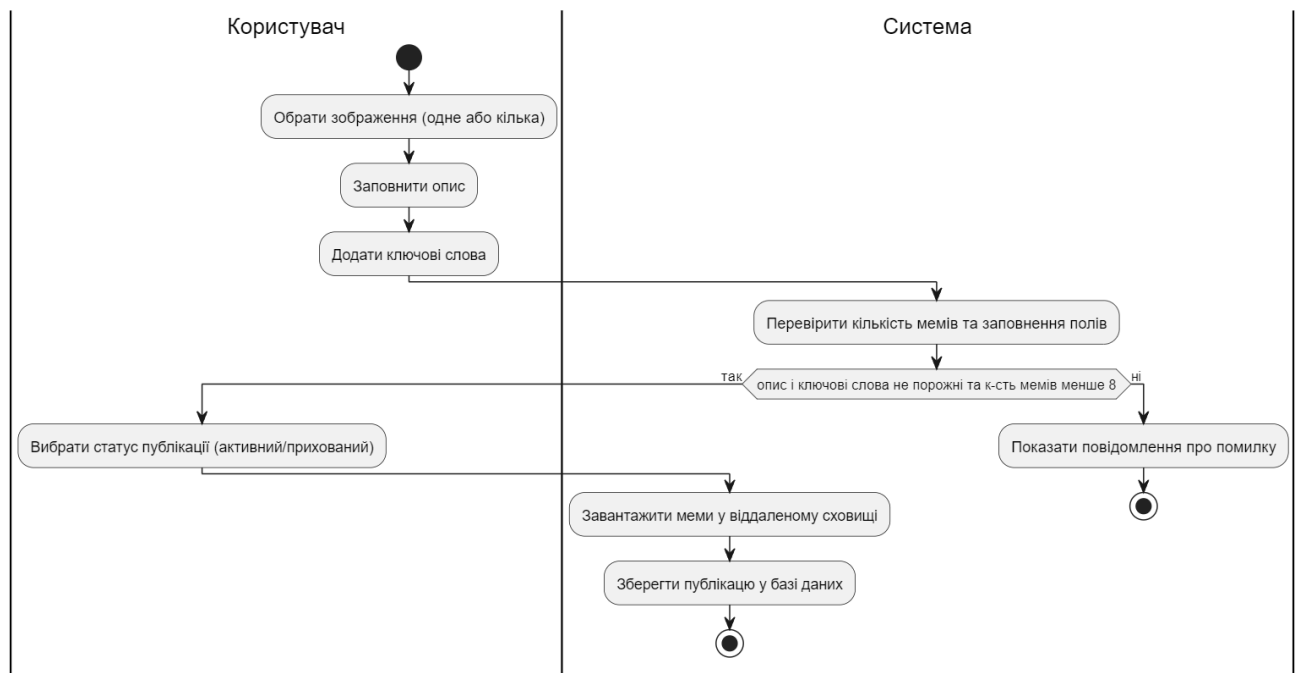


Рисунок 2.4. Діаграма активності для публікації мему

Ця діаграма відображає процес створення мему з подальшим збереженням у системі. Користувач починає з вибору одного або кількох зображень, які будуть основою майбутньої публікації. Далі він заповнює текстовий опис, щоб передати ідею або контекст, а також додає ключові слова – вони допомагають іншим користувачам знаходити мем за змістом чи темою.

Після цього система перевіряє, чи заповнені всі обов’язкові поля (опис і ключові слова), а також чи не перевищено ліміт створених мемів – максимальна дозволена кількість становить 8. Якщо всі умови виконані, користувач обирає статус публікації: активний, щоб мем одразу був доступний публіці, або прихований, якщо він поки не має бути опублікований.

Потім система переходить до технічних дій: завантажує файли зображень до віддаленого сховища uploadthing, а також записує всі дані про публікацію в базу.

На етапі створення публікації користувач може генерувати зображення за допомогою штучного інтелекту.

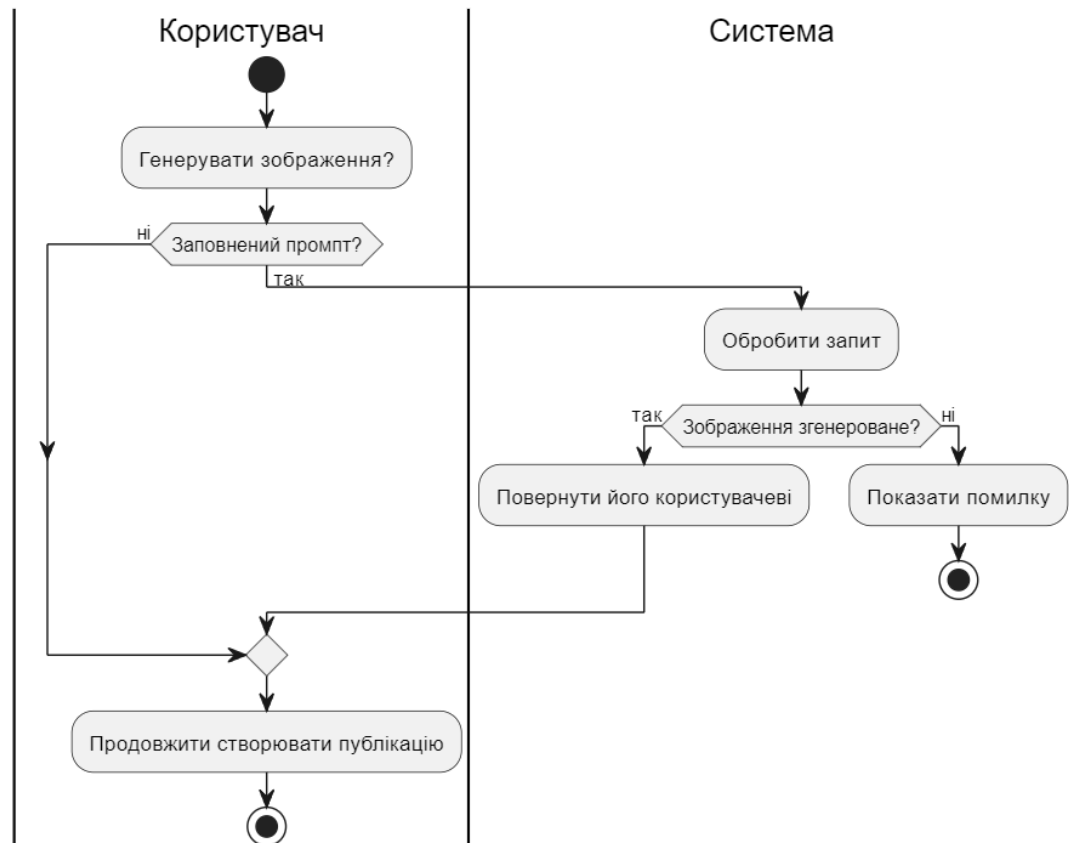


Рисунок 2.5. Діаграма активності генерації зображення за допомогою штучного інтелекту

На діаграмі описується процес генерації зображення як частину створення мему. Все починається з ініціативи користувача: він вирішує, чи хоче згенерувати зображення. Якщо так, система перевіряє, чи заповнено поле з промптом – текстовим описом, який слугує основою для генерації.

Якщо промпт заповнений, система надсилає запит до генератора зображень (у нашому випадку нейромережі Gemini), після чого очікує на відповідь. Якщо зображення успішно згенероване, воно одразу повертається користувачеві для подальшого використання. Якщо ж виникає помилка – наприклад, генерація не вдалася – система інформує користувача про це через повідомлення про помилку, і

процес завершується. Після цього користувач повертається до редагування деталей публікації.

Усі зображення (зокрема й ті, що згенеровані ШІ) можна редагувати у редакторі зображень. Якщо користувач на етапі перегляду доданих зображень не задоволений їх виглядом.

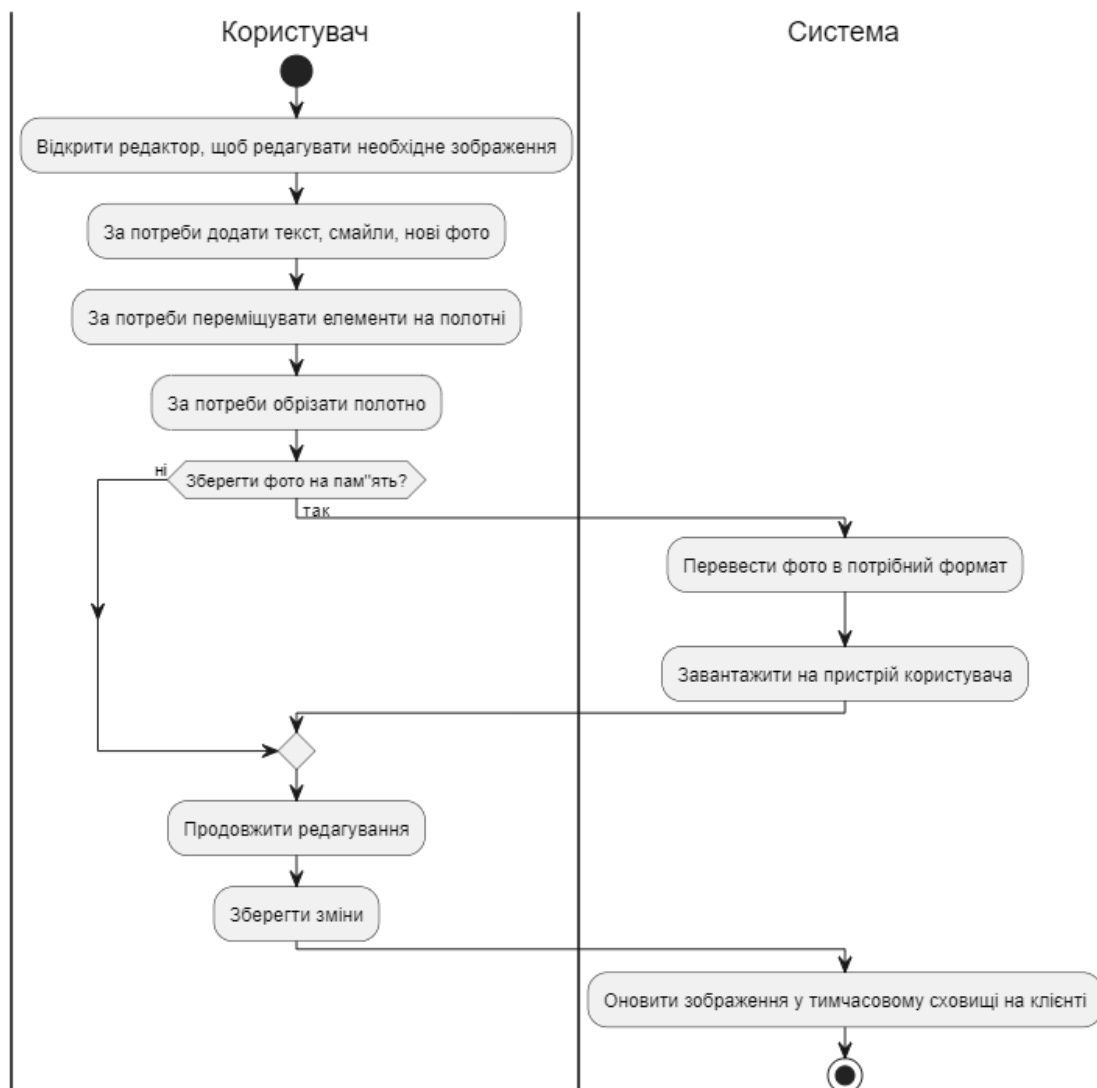


Рисунок 2.6. Діаграма активності редагування зображення для створення мему

Ця діаграма описує процес взаємодії користувача з редактором зображень у додатку для створення мемів. Користувач починає з відкриття редактора, де обирає конкретне зображення для редагування. У цьому редакторі він має можливість додавати нові візуальні елементи. Також користувач може змінювати розташування об'єктів на полотні, переміщуючи їх у потрібне місце, або обрізати саму межу полотна, щоб краще оформити композицію мему.

Після цього система перевіряє, чи хоче користувач зберегти зображення на свій пристрій. Якщо так – система конвертує його у відповідний формат та ініціює завантаження на локальну пам'ять. Якщо користувач не обирає цю дію, процес триває без цього кроку.

Коли користувач завершує зміни, він зберігає оновлений варіант. У відповідь система записує нову версію зображення у тимчасове сховище на клієнті, що дозволяє використовувати його у подальших кроках або публікувати пізніше.

2.4. Реалізація функціоналу платформи

Серверна логіка побудована на основі фреймворку NestJS, що забезпечує масштабованість і модульність. У реалізації застосовано патерн Singleton, що гарантує єдиний екземпляр кожного сервісу в межах усього життєвого циклу додатку, підвищуючи ефективність роботи з ресурсами та стабільність поведінки ін'єктованих залежностей. Візуалізація цього патерну можна побачити на рис. 2.7.

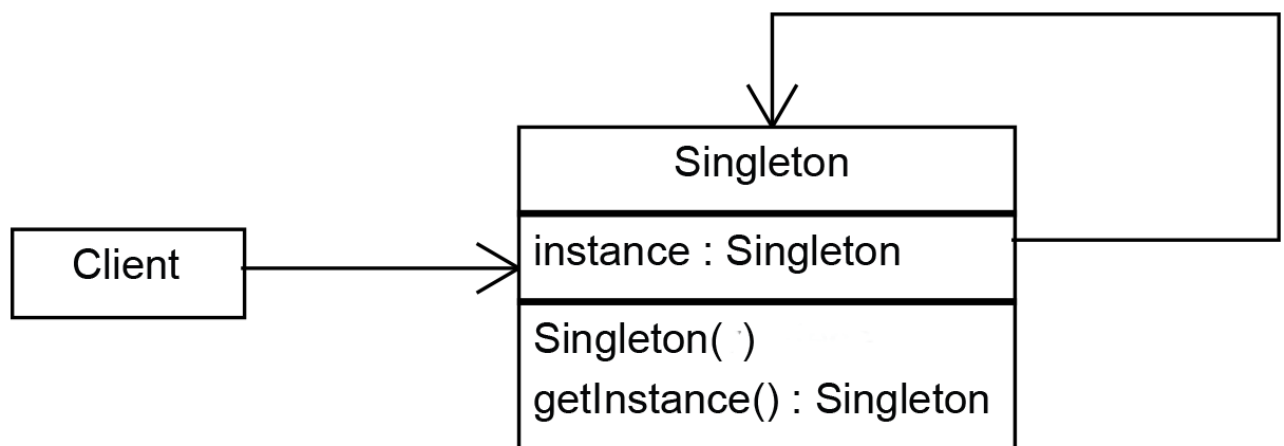


Рисунок 2.7. Візуалізація патерну Singleton

Центральним елементом платформи є модуль обробки публікацій. Основна функція полягає в створенні, оновленні, фільтрації, перегляді та видаленні записів користувачів. Для отримання публікацій реалізовано гнучку систему фільтрації за описом, ключовими словами та авторством. Запити формуються через query builder з урахуванням складених умов пошуку та сортування, наприклад, за датою створення. У процесі створення публікації здійснюється автоматичне завантаження

зображень, після чого отримані URL-адреси зберігаються разом із метаданими в базі даних.

Завантаження файлів здійснюється через окремий сервіс, що взаємодіє з хмарним сховищем, використовуючи API від Uploadthing [19]. Завантаження файлів на сховище можна переглянути на рисунку 2.8.

```
async uploadFiles(files: Express.Multer.File[]): Promise<string[]> {
  if (files.length < 1) {
    throw new BadRequestException(['No files provided']);
  }

  const modifiedFiles = await Promise.all(
    files.map(async (file) => await this.modifyFile(file)),
  );

  try {
    const res = await this.utApi.uploadFiles(modifiedFiles);

    return res.map((file) => file.data.url);
  } catch (error) {
    console.error('Upload failed:', error);
    throw new InternalServerErrorException();
  }
}
```

Рисунок 2.8. Завантаження файлів у сховище

Функція отримує масив зображень. Якщо ж масив порожній, повертається помилка, в іншому випадку зображення обробляються бібліотекою sharp – змінюється розмір відповідно до встановлених обмежень, зберігаючи при цьому пропорції і зменшує навантаження на клієнт. Кожен файл отримує унікальну назву, що унеможливорює конфлікти та дублікати. І лише після цього файли зберігаються у сховищі. При успішному збереженні сервіс повертає масив рядків з посиланням на зображення, при невдачі – повертається помилка. Сервіс також підтримує видалення файлів за ключами, отриманими з URL, що дозволяє підтримувати чистоту сховища при оновленні чи видаленні публікацій.

Ще одним надважливим модулем є модуль авторизації, що забезпечує класичну автентифікацію за електронною поштою та паролем, а також інтеграцію з Google через OAuth. Для реєстрації нового користувача передбачено перевірку відповідності пароля і його підтвердження, а також захист пароля шляхом

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				34
Змн.	Арк.	№ докум.	Підпис	Дата		

хешування через bcrypt. Після успішного входу або реєстрації генерується JWT-токен, який клієнт використовує для аутентифікованих запитів. Для обмеження доступу до певних ендпоїнтів використовується один з кількох захисних декораторів. На рисунку 2.9 наведений приклад використання декоратора, що вимагає обов'язкову авторизацію, на рисунку 2.10 – лише отримує дані авторизованого користувача, якщо такий існує.

```
@HttpCode(HttpStatus.CREATED)
@UseInterceptors(FileInterceptor('picture'))
@UseGuards(JwtAuthGuard)
@Post()
createComment(
```

Рисунок 2.9. Повний захист ендпоїнта від неавторизованих користувачів

```
💡 @HttpCode(HttpStatus.OK)
@UseGuards(OptionalJwtAuthGuard)
@Get()
getPublications(
```

Рисунок 2.10. Частковий захист ендпоїнта від неавторизованих користувачів

У випадку входу через Google, у разі відсутності користувача, створюється новий запис із мінімальним набором обов'язкових полів: Google email, ім'я користувача, що є скороченою версією електронної пошти, а пароль порожній, що дозволяє користувачу здійснювати автентифікацію лише за допомогою Google (див. рис. 2.11).

```
if (error.status == 404) {
  user = await this.userService.createOne({
    email,
    username: email.split('@')[0],
    fullName: profile.displayName,
    role: 'user',
    password: null,
  });
}
```

Рисунок 2.11. Створення нового користувача при Google авторизації, якщо його не існувало в БД

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				35
Змн.	Арк.	№ докум.	Підпис	Дата		

В результаті, серверна частина успішно імплементує логіку роботи з даними користувачів та їх публікацій. Виконує роботу з їх збереженням у базу даних та віддаленні сховища.

Центральним інструментом управління серверною взаємодією виступає TanStack Query. Ця бібліотека реалізує патерн кешованих запитів (рис. 2.12), що дозволяє зменшити кількість повторних HTTP-запитів, забезпечити миттєвий ререндер інтерфейсу після змін на бекенді та синхронізувати дані у реальному часі. Дані фетчаться через функції, прив'язані до ключів запитів, що дозволяє гнучко контролювати інвалідовані записи, повторні отримання даних та оновлення кешу після мутацій. У разі помилки передбачена централізована обробка через onError-гілки, що дозволяє відображати користувачу відповідні повідомлення або повторювати запити автоматично. Це також спрощує логіку infinite scroll, що є важливою складовою відображення величезної кількості публікацій та коментарів до них.

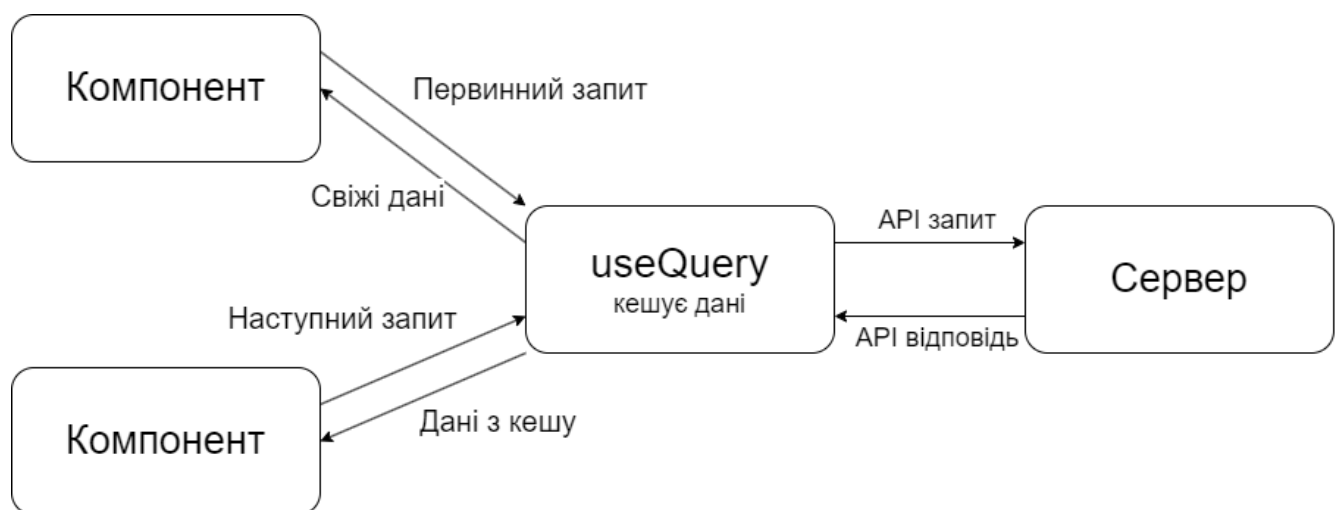


Рисунок 2.12. Візуалізація роботи TanStack Query

Для побудови інтерфейсу застосовується UI-бібліотека shadcn, яка поєднує переваги Tailwind CSS, Radix UI та модульного підходу до компонентів [20]. Інтерфейс побудований за принципом Atomic Design, де складні компоненти формуються з базових атомів. Наприклад, форма публікації складається з валідаційних полів, кнопок, модальних вікон та попереджень, реалізованих як окремі компоненти бібліотеки. Завдяки Tailwind інтерфейс зберігає адаптивність,

швидкість стилізації та високий рівень кастомізації зовнішнього вигляду. Основна перевага цієї бібліотеки – повний контроль над стилями кожного компоненту, так як кожний компонент завантажується в проєкт, що дозволяє повністю змінювати код компонентів. На рисунку 2.13 можна переглянути деякі завантажені компоненти з цієї бібліотеки.

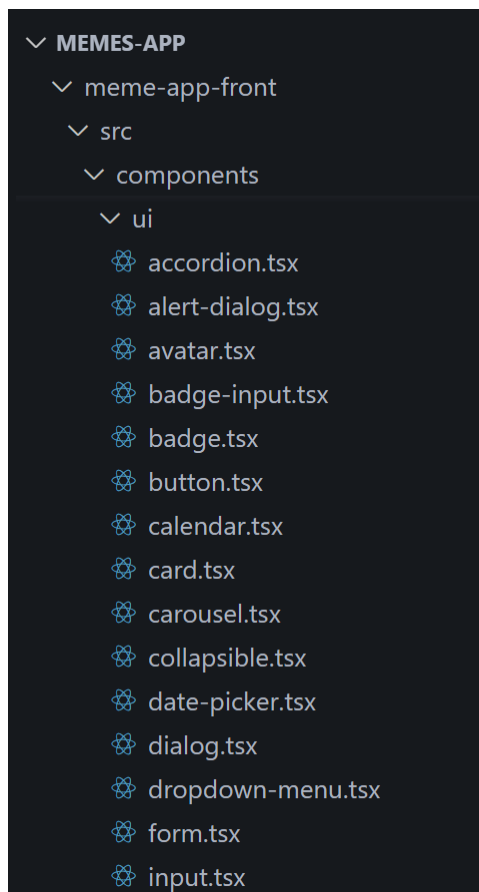


Рисунок 2.13. Частина компонентів shadcn

Ключовим інструментом взаємодії з медіаконтентом є Konva – бібліотека для побудови інтерактивного редактора зображень на основі HTML5 Canvas. Вона дозволяє реалізовувати масштабування, обертання та групування об’єктів на полотні. Під час редагування користувач може додавати текст, смайли та зображення, які зберігаються у вигляді JSON-представлення, що полегшує подальше збереження, передачу та повторний рендер. Для підвищення продуктивності використовуються кешування шарів, оптимізована відмальовка лише змінених частин, а також throttle-події для масштабування та переміщення. Для імплементації редактору були створені елементи редактора, що дозволяють

користувачу взаємодіяти з елементами полотна, панелі, що надають певний функціонал для кожного елемента, та самі елементи, що зберігають дані про свої розміри та положення на полотні (рис. 2.14).

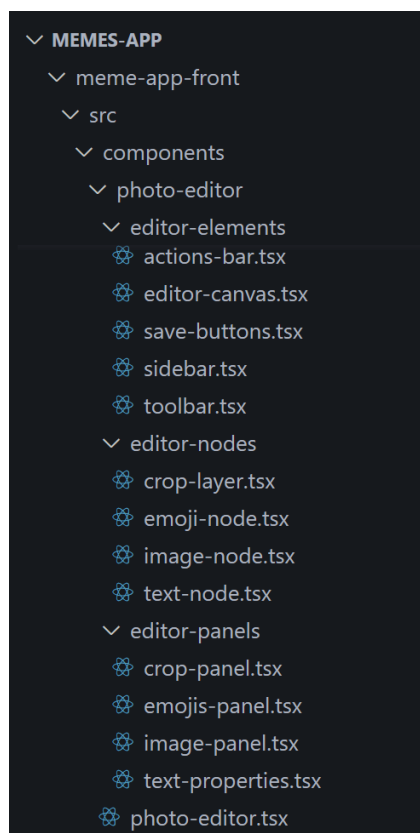


Рисунок 2.14. Компоненти редактора зображень

Клієнтська логіка інтегрується через глобальний менеджер стану Zustand, що забезпечує синхронізацію дій між редактора та менеджмент зображень при створенні публікації (див. додаток А). Це створює єдину узгоджену модель взаємодії, в якій користувач бачить миттєві результати своїх дій, а також має можливість повернутись до попередніх станів без втрати прогресу.

Таким чином, клієнтська частина платформи забезпечує високий рівень UX, оптимізовану роботу з даними та медіа, а також модульність та масштабованість, що відповідає сучасним вимогам до вебзастосунків.

Останній важливий компонент платформи реалізований за допомогою простого Express застосунку, що дозволяє швидко та успішно спілкуватися з Gemini API для генерації зображень за описом користувача. Запит на генерацію зображення з промптом надсилається з клієнта. Після цього сервер отримує цей

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

промпт та надсилає запит на Gemini API використовуючи модель «gemini-2.0-flash-preview-image-generation», відповідь від Gemini приходить у чанках, що сервер обрабляє та повертає на клієнт правильно відформатоване зображення. Весь код цього запиту можна переглянути у додатку Б.

Повернення валідного зображення обробляється TanStack Query (див. рис.2.15).

```
onSuccess: (data) => {
  queryClient.invalidateQueries({ queryKey: queryKeys.generateAiImage() });
  const file = base64ToFile(data.base64, uuid(), 'image/png');

  addPhotos([
    {
      id: uuid(),
      file,
      preview: URL.createObjectURL(file),
      name: file.name,
      width: 0,
      height: 0,
      edited: false,
      ai: true,
    },
  ]);
},
```

Рисунок 2.15. Отримання згенерованого зображення

При успішному виконанні запиту зображення конвертується у тип File та зберігається у Zustand сховищі для подальшої роботи Як і для решти зображень, воно доступне для редагування чи видалення.

Висновки до другого розділу

У другому розділі було детально окреслено архітектурне та функціональне проектування платформи, визначаючи ключові компоненти та взаємодії системи. Було розроблено варіанти використання системи, де чітко окреслено ролі користувача та гостя, а також типові сценарії їхньої взаємодії з платформою.

Класова діаграма та структура бази даних демонструють об'єктно-орієнтований підхід до моделювання системи. Вона включає гнучку і масштабовану модель даних, що забезпечує збереження й обробку основної інформації. Такий підхід дозволяє ефективно обслуговувати основні функції соціальної взаємодії.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдяки побудові діаграм активності були окреслені ключові алгоритми взаємодії користувача із системою, зокрема процеси реєстрації, авторизації, , публікації мемів, їх редагування та генерації зображень за допомогою ШІ. Це забезпечує чітке розуміння логіки платформи як для користувачів, так і для розробників.

Загалом, реалізований функціонал платформи демонструє цілісну архітектуру із чітким розмежуванням обов'язків між сервісами. Такий підхід дозволяє не лише підтримувати чистоту коду, а й легко розширювати платформу в майбутньому.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ПЛАТФОРМОЮ

3.1. Розгортання платформи

Під час розробки платформи всі компоненти розгортаються локально, для глобального користування платформу було розгорнуто на серверах за допомогою селфхост сервісів Coolify [21] та Dokploy [22]. Діаграму розгортання платформи можна переглянути на рисунку 3.1.

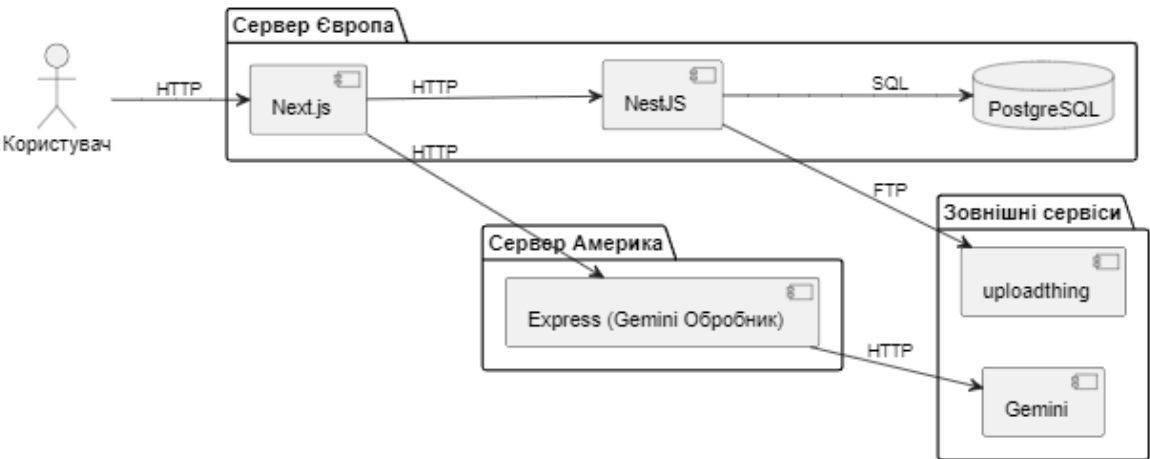


Рисунок 3.1. Діаграма розгортання платформи

Ця діаграма демонструє, як користувач взаємодіє з розподіленою інфраструктурою вебзастосунку, що охоплює сервери в різних регіонах та підключення до зовнішніх сервісів. У центрі всієї взаємодії – користувач, який надсилає HTTP-запити до інтерфейсу, створеного за допомогою Next.js. Цей фреймворк розміщено на європейському сервері разом із NestJS, що відповідає за обробку логіки на бекенді.

NestJS отримує запити від Next.js, опрацьовує їх та взаємодіє з базою даних PostgreSQL за допомогою SQL-запитів. У разі потреби збереження чи отримання зображень, NestJS звертається до зовнішнього сервісу UploadThing через FTP або відповідний API, залежно від реалізації.

Окрім цього, Next.js також має канал взаємодії з американським сервером, де розміщено Express-додаток, що виступає посередником для запитів до штучного інтелекту Gemini. Express, у свою чергу, надсилає HTTP-запити до самого сервісу Gemini, обробляючи відповіді й повертаючи результат клієнту через Next.js. Причина розміщення цього сервісу так далеко полягає в політиці Google

використання Gemini. Адже для більшості європейських країн не дозволено розгортати обрану модель генерації зображень.

Таким чином, уся система є взаємопов'язаною мережею, де Next.js виступає центральною точкою координації між користувачем, бекендом, зовнішнім AI-сервісом та сховищем файлів, а фізичне розміщення компонентів у різних регіонах дозволяє балансувати навантаження та оптимізувати час відповіді.

Деплой усіх частин платформи налаштовується за допомогою розгортання Docker контейнерів на серверах. В додатку В можна переглянути Docker-файли написані для кожної частини платформи. В цих файлах описані інструкції для встановлення залежностей, збирання та запуску проєкту.

В результаті були розгорнуті три частини платформи: клієнт [23], сервер [24] та ШІ обробник [25].

3.2. Структура інтерфейсу та користувацька взаємодія

При першому візиті сайту неавторизований користувач бачить головну сторінку з найсвіжішими публікаціями (рис. 3.2).

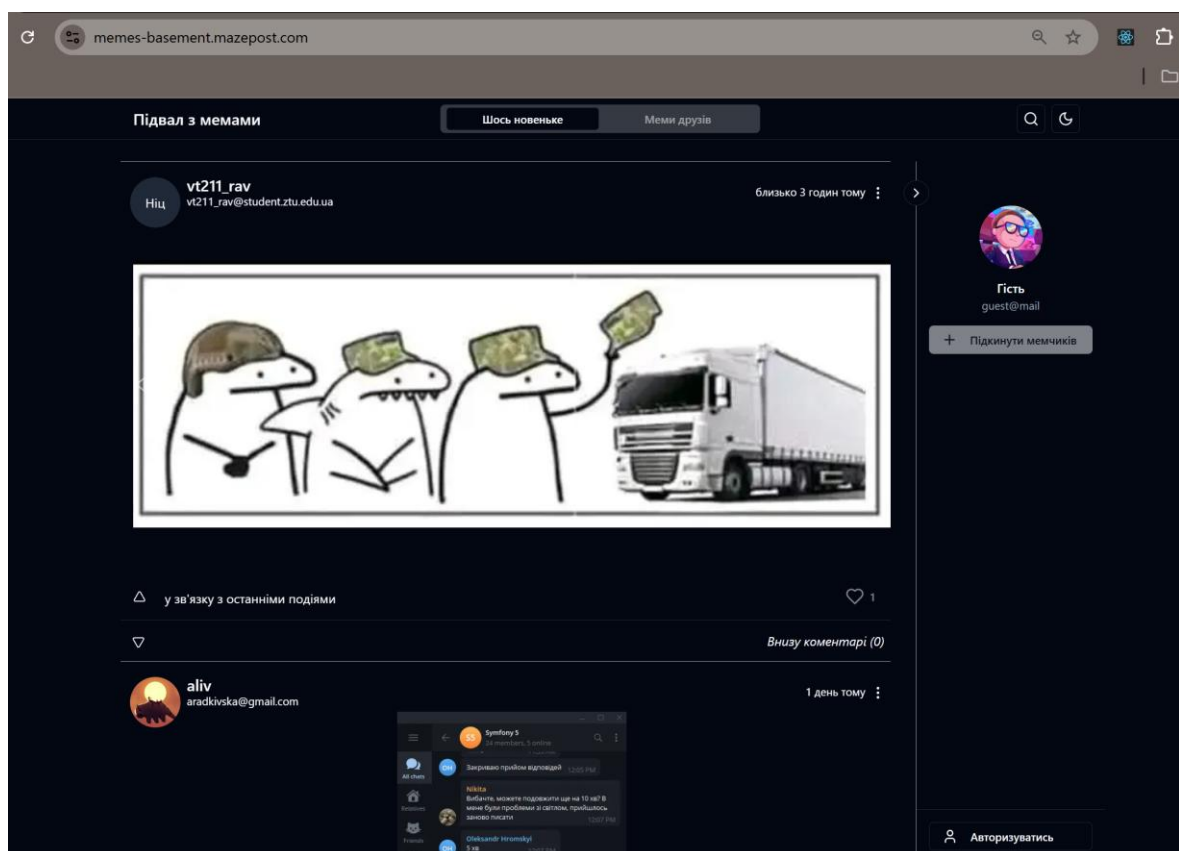


Рисунок 3.2. Головна сторінка платформи

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

Також вебсайт доступний для зручного перегляду з мобільного пристрою, що можна переглянути на рисунку 3.3.

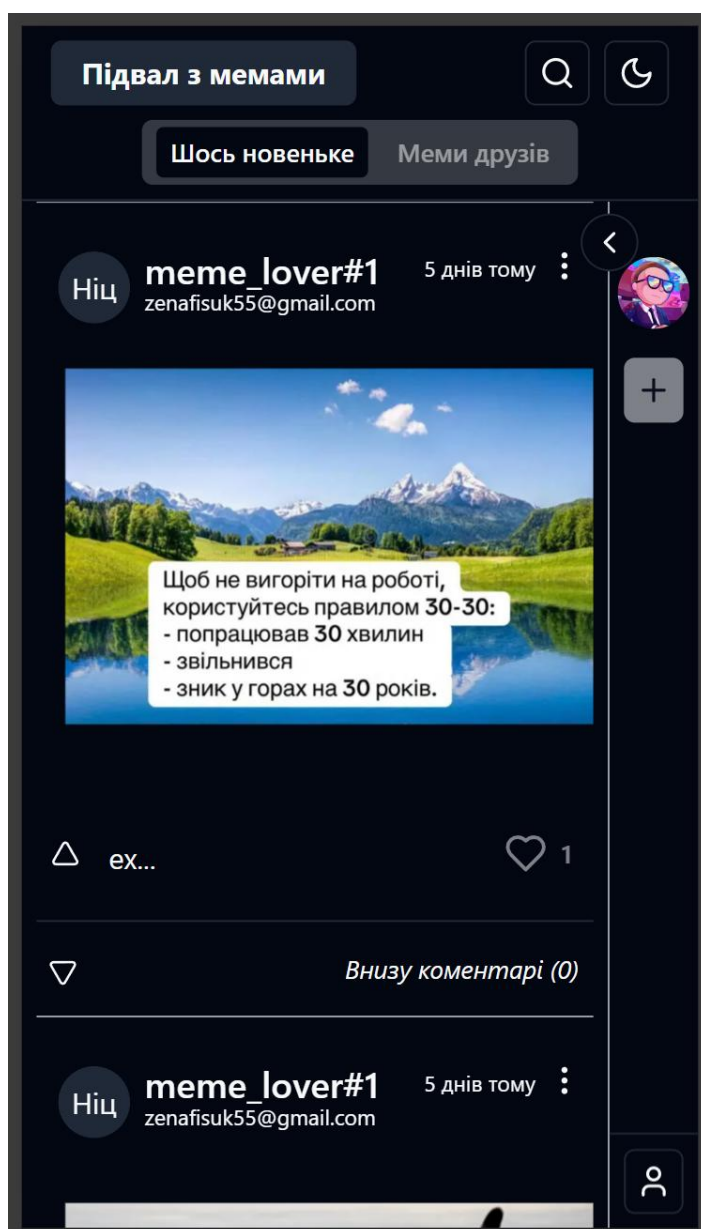


Рисунок 3.3. Головна сторінка платформи на мобільному пристрої

Взаємодія з публікаціями йому недоступна, тому що для цього спочатку потрібно авторизуватися. Проте вже на цьому етапі є змога шукати потрібні мемі (рис. 3.4) та переглядати профілі користувачів (рис.3.5).

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

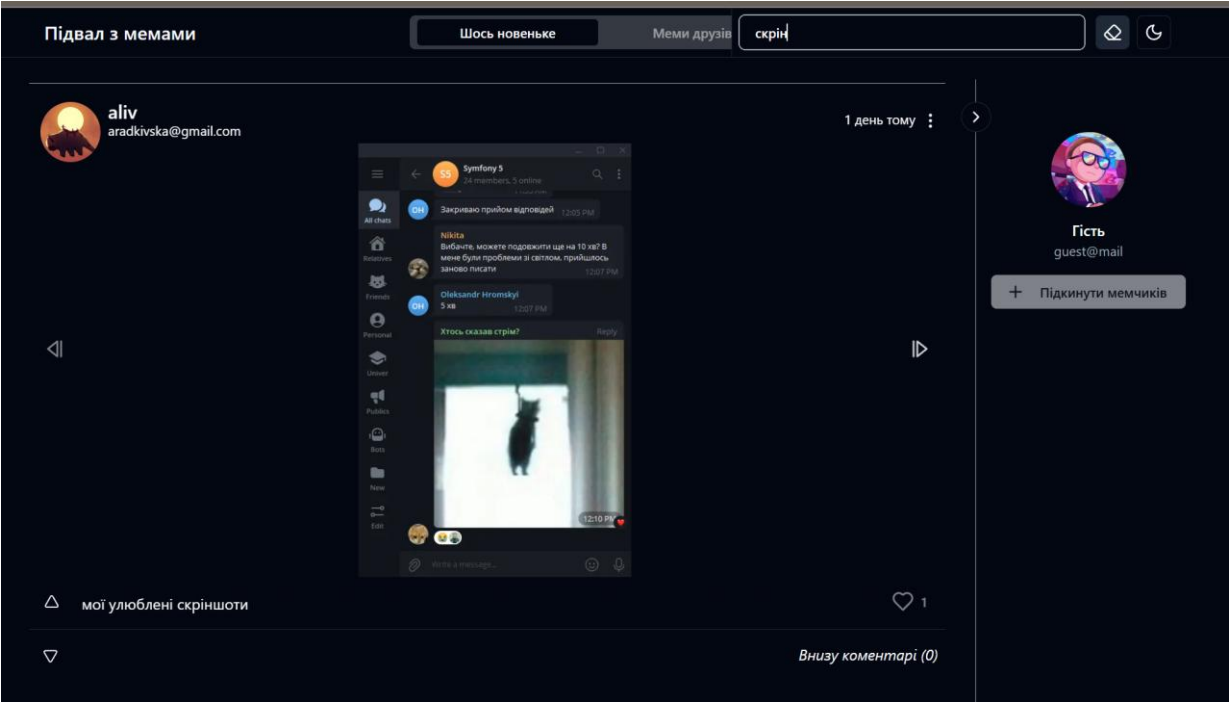


Рисунок 3.4. Пошук публікацій

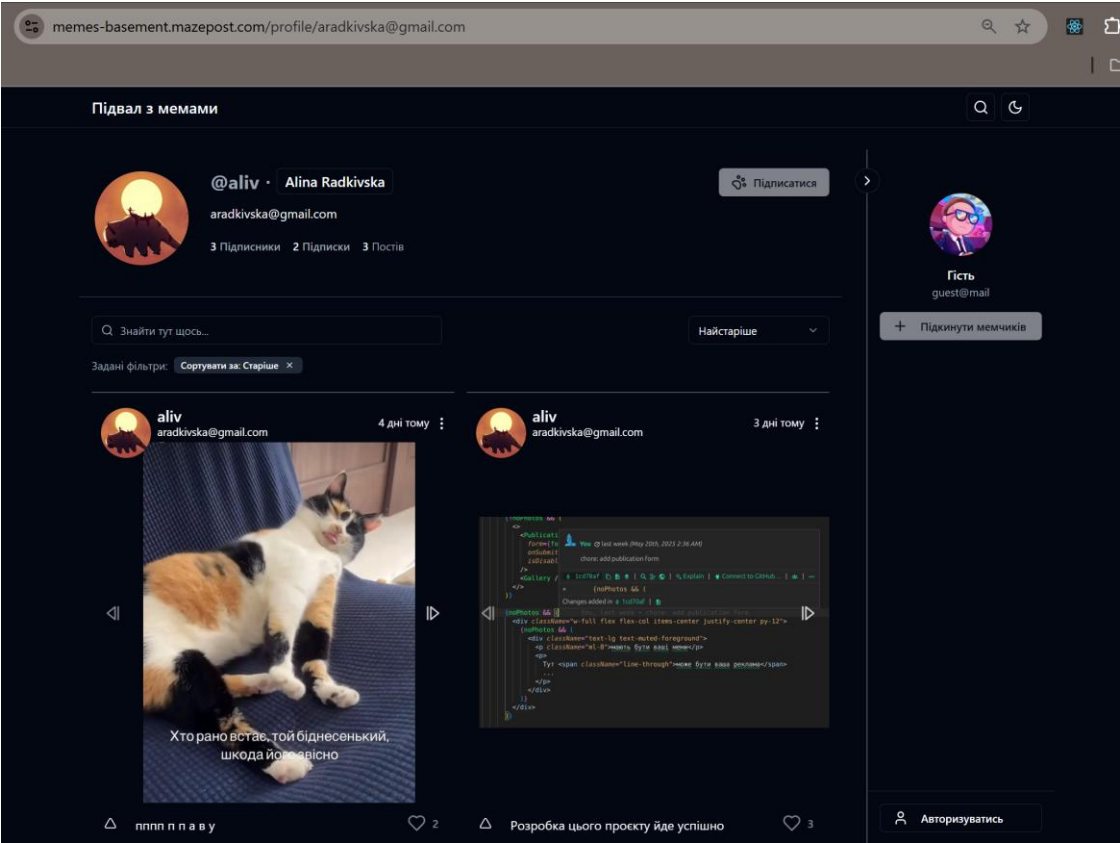


Рисунок 3.5. Профіль іншого авторизованого користувача платформи

Також будь-яким користувачам дозволено завантажувати мему, що зображено на рисунку 3.6.

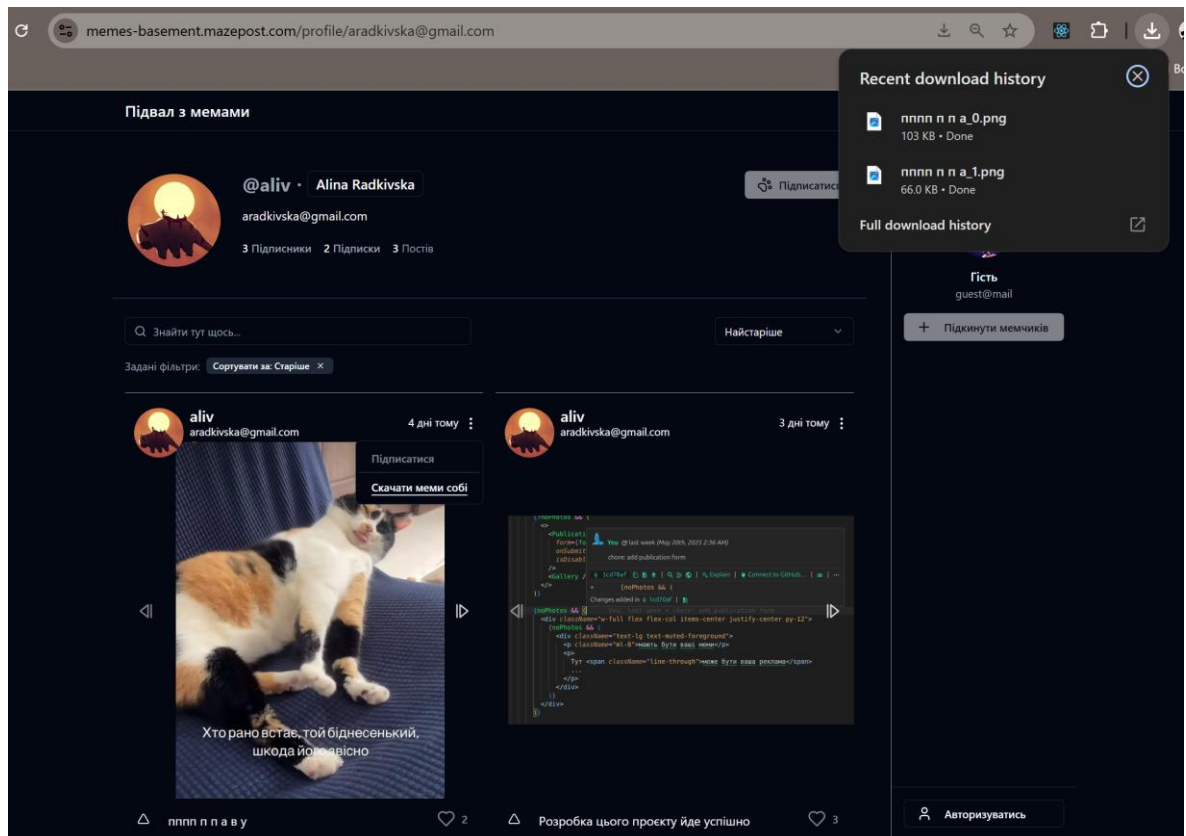


Рисунок 3.6. Завантаження мемів на пристрій користувача

Сторінка авторизації має форму для логіну і кнопку для Google швидкої авторизації без паролю (рис. 3.7). На тій же сторінці є можливість зареєструватись за допомогою форми або знову ж таки – за допомогою акаунту Google (рис. 3.8)

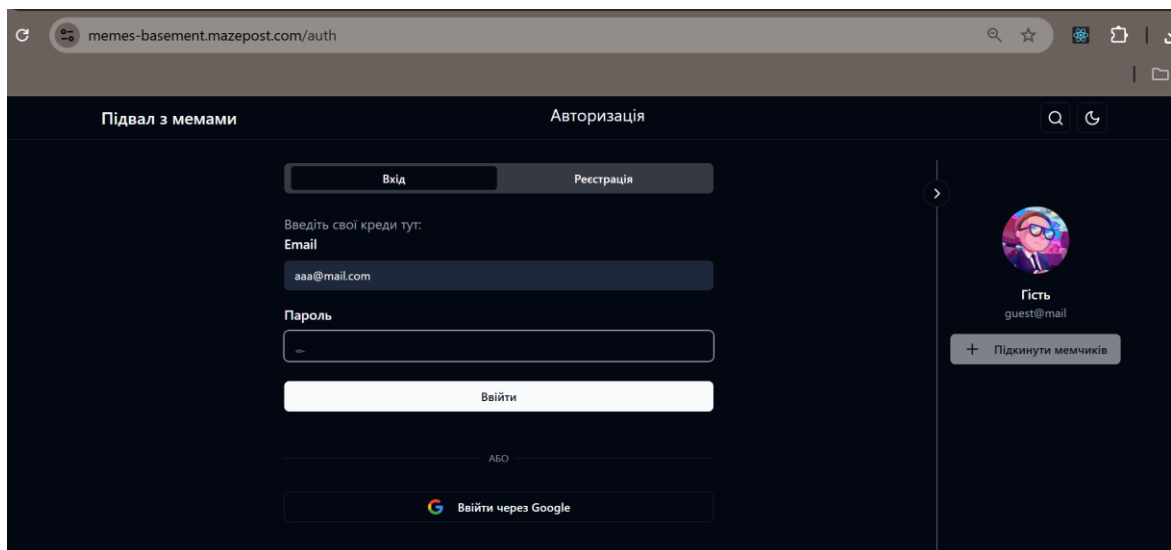


Рисунок 3.7. Сторінка авторизації

		Радківка А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

Підвал з мемами

Авторизація

Вхід

Реєстрація

Заповніть інфу про себе:

Аватарка (не обов'язково)

Email

new@mail.com

Нікнейм

new_user

Повне ім'я (не обов'язково)

Newbie

Якась дуже важлива інформація, що буде в профілі (не обов'язково)

Пафосна цитата

Дата народження (не обов'язково)

Оберіть дату

Пароль

.....

Повторіть пароль

.....

Зареєструватися

Гість

guest@mail

+ Підкинути мемчиків

Авторизуватись

Рисунок 3.8. Сторінка реєстрації

Авторизований користувач може редагувати дані свого профілю за потреби, що зображено на рисунку 3.9.

Підвал з мемами

new_user

Нікнейм

new_user

Повне ім'я

Newbie

Важлива інформація для профілю

Дуже пафосна і відома кожному псу цитата

Дата народження

11-е лютого 2004 р.

червень 2025

пн вт ср чт пт сб нд

1

2 3 4 5 6 7 8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

23 24 25 26 27 28 29

30

Зберегти зміни

new_user

new@mail.com

0 Підписники 0 Підписки

Змінити інфо

+ Підкинути мемчиків

Вийти з акаунту

Рисунок 3.9. Сторінка редагування даних користувача

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

Після авторизації користувач може взаємодіяти з контентом: ставити вподобайки та залишати коментарі (рис. 3.10).

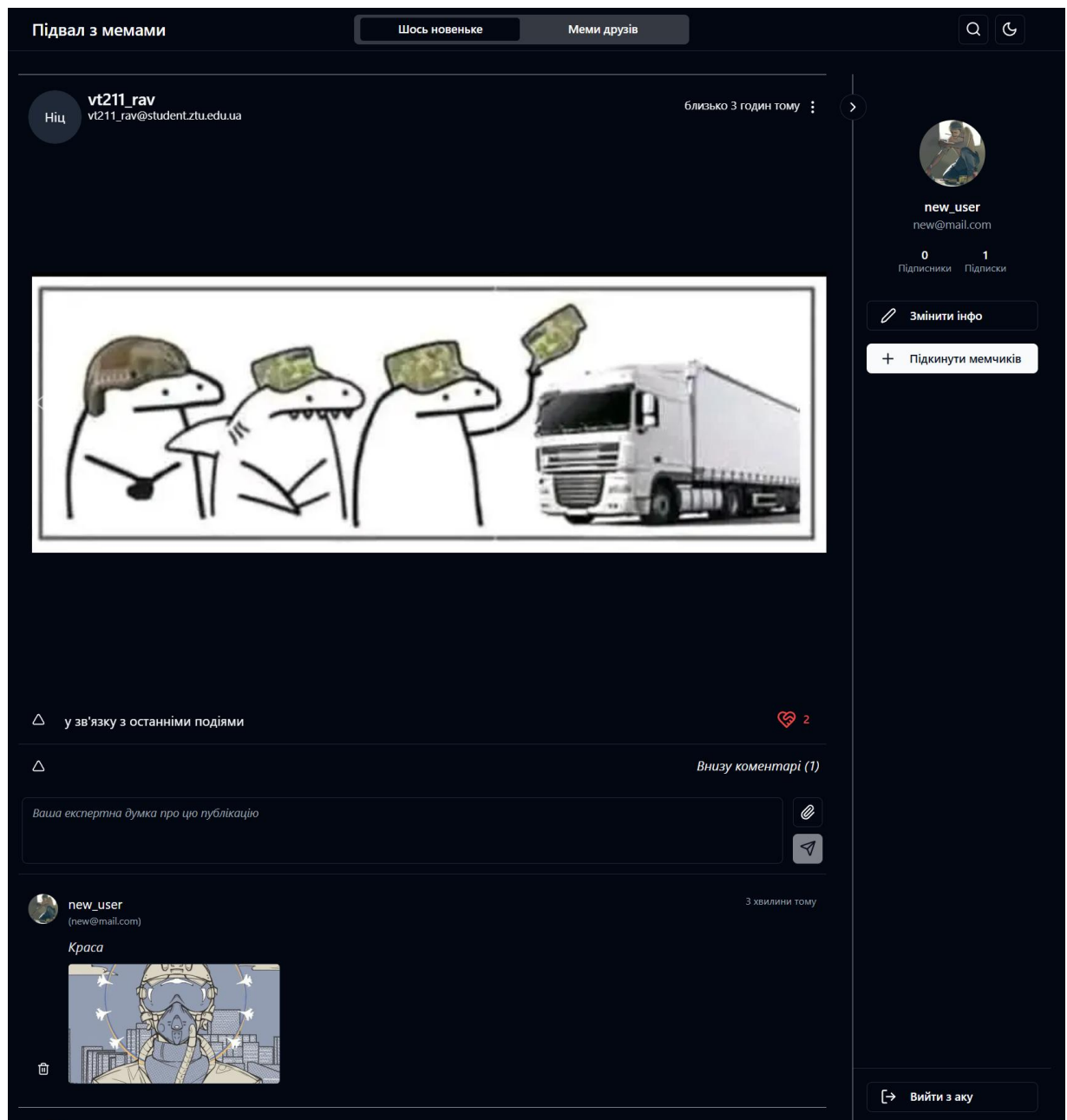


Рисунок 3.10. Вподобана та прокоментована публікація

Також з'являється можливість не лише переглядати профілі інших користувачів, а й підписуватись на них, що можна переглянути на рисунку 3.11. Також для авторизованих користувачів з'являється можливість переглядати лише мему користувачів з підписок на головній сторінці (рис.3.12).

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

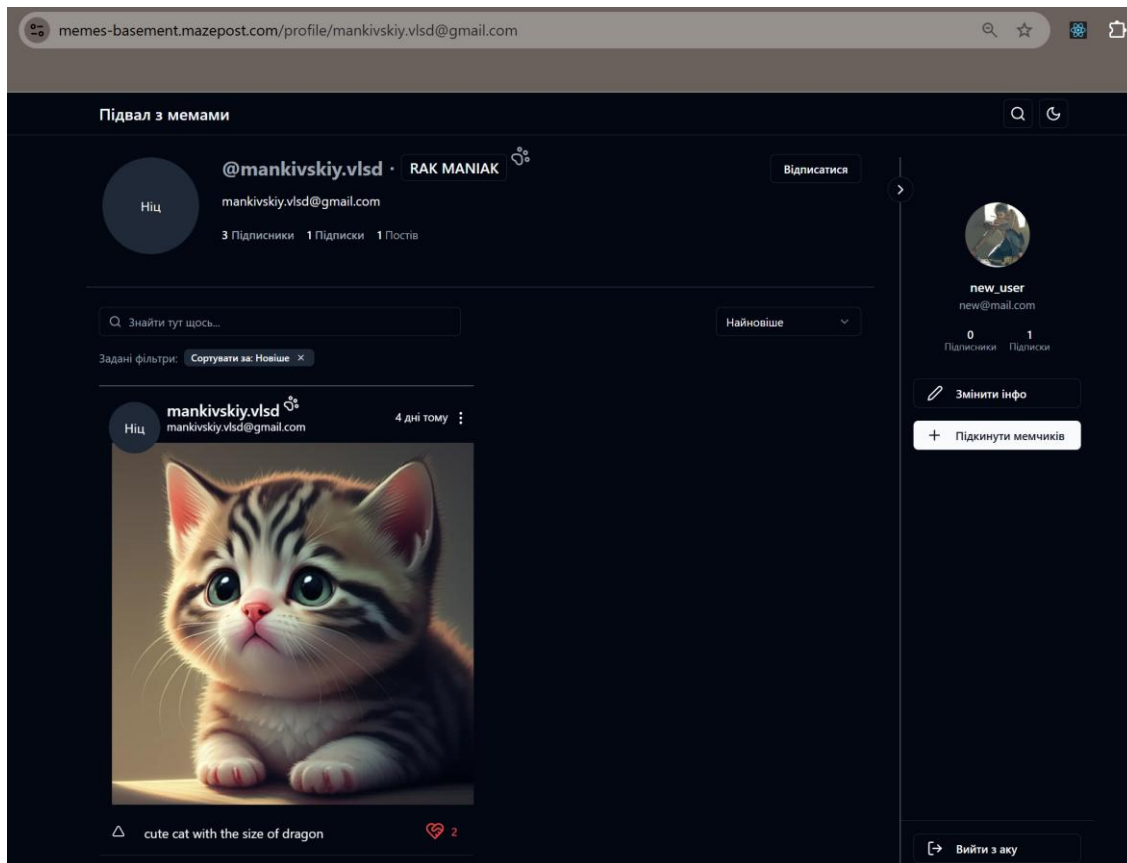


Рисунок 3.11. Профіль іншого користувача, що в підписах поточного

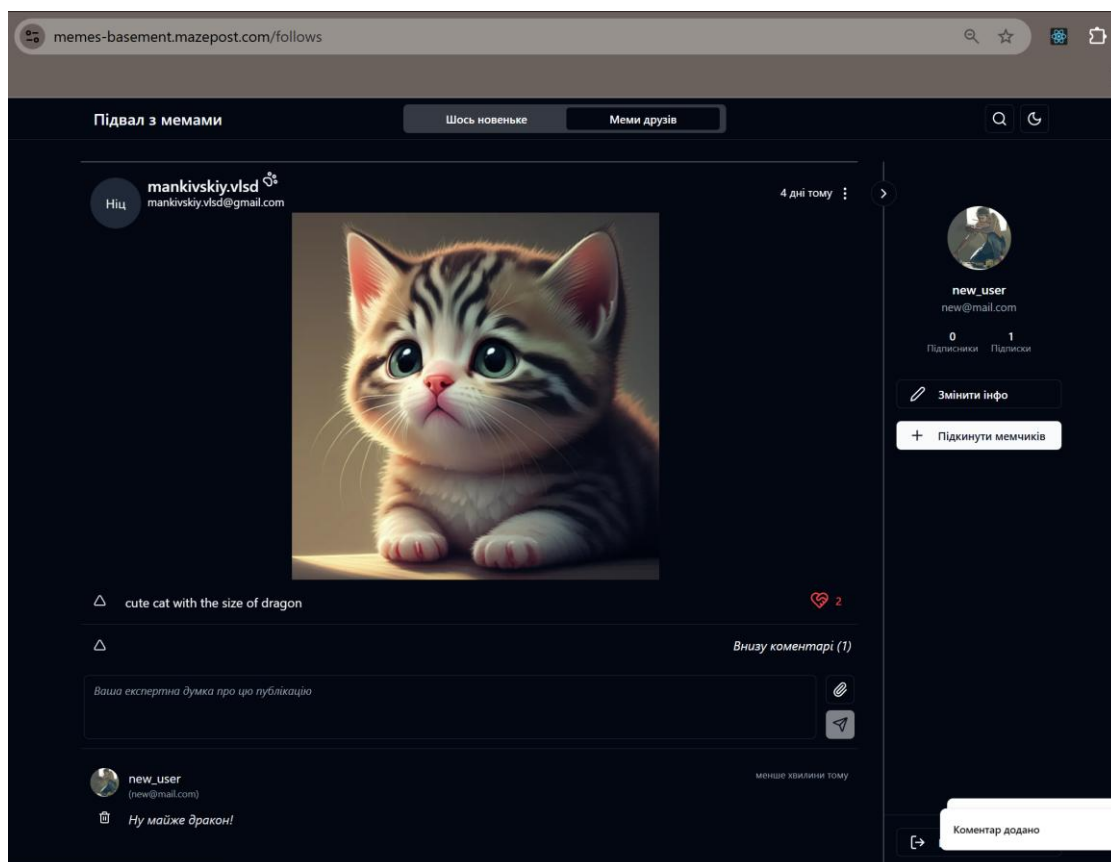


Рисунок 3.12. Головна сторінка з публікаціями користувачів з підписок

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

Після авторизації також з'являється можливість завантажувати зображення та користуватися редактором та генерувати зображення (рис.3.13).

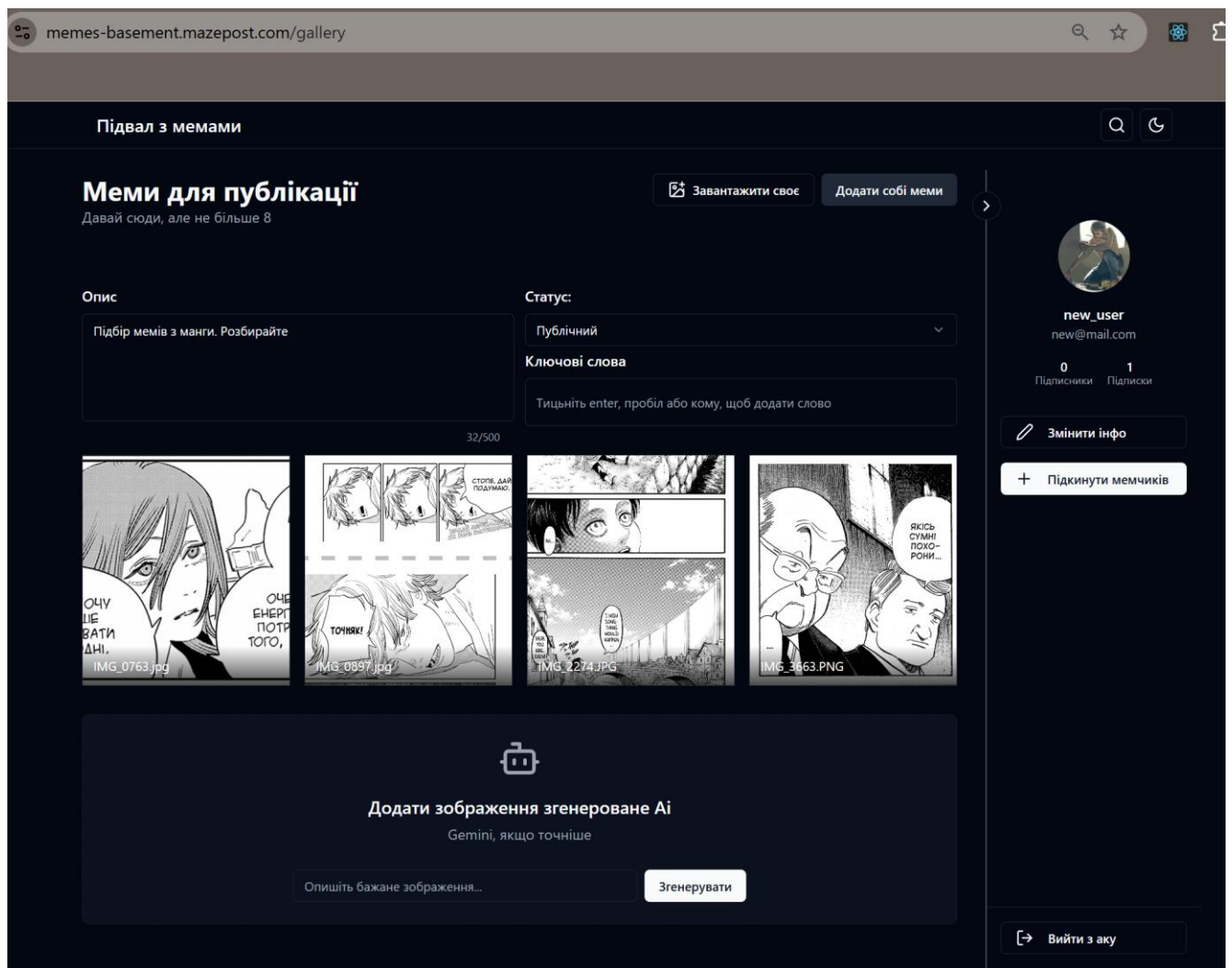


Рисунок 3.13. Сторінка створення публікації

Поглянемо ближче на редагування зображення у редакторі. На рисунку 3.14 зображено додавання тексту, на рисунку 3.15 – додавання ще одного зображення, на рисунку 3.16 – додавання емоджі.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

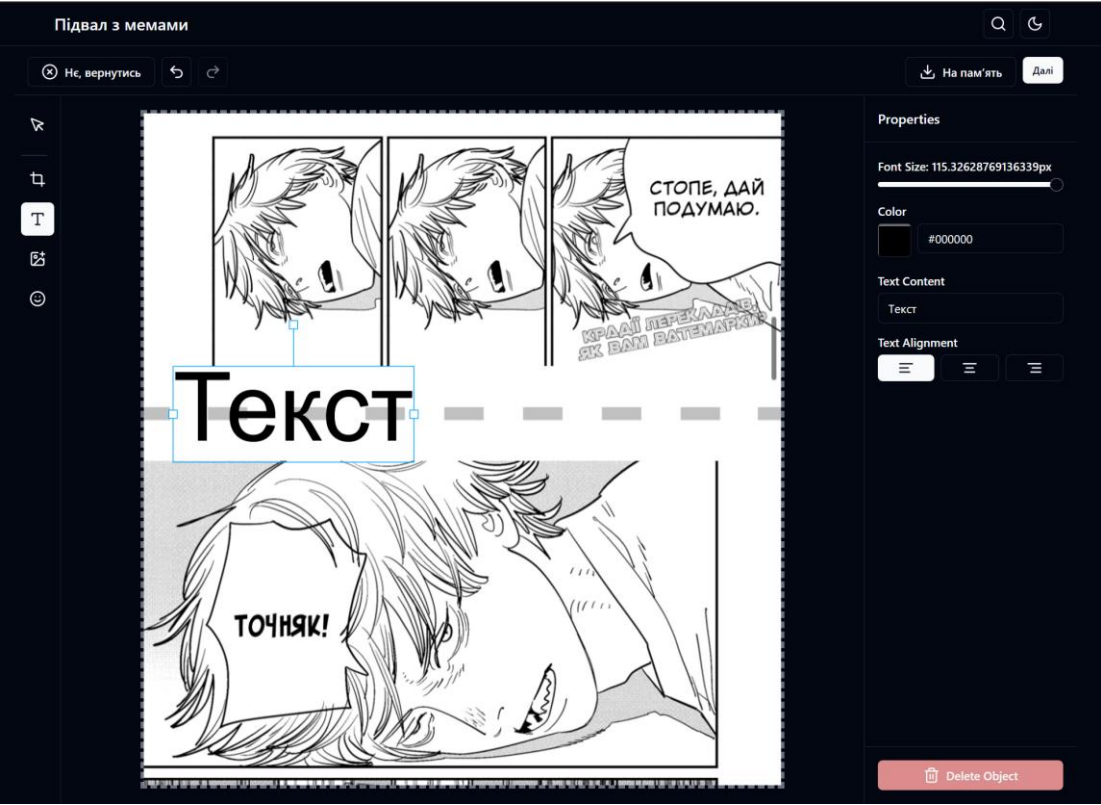


Рисунок 3.14. Додавання тексту

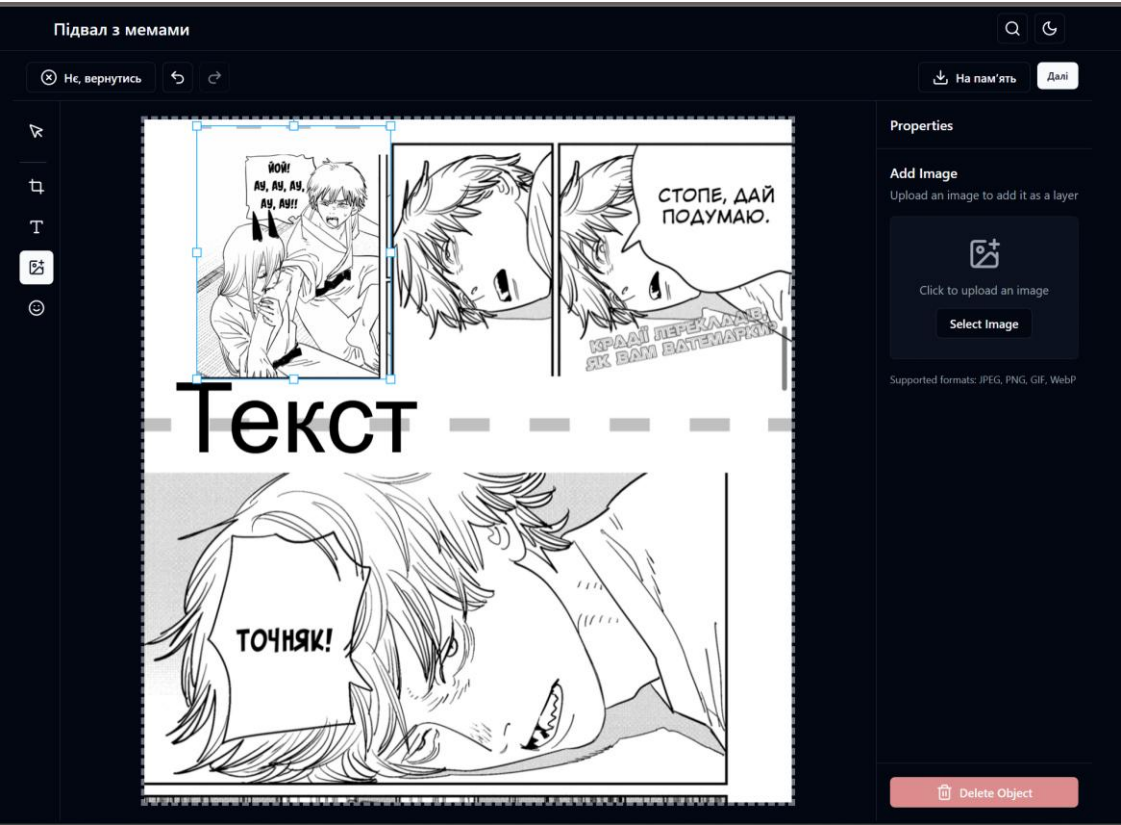


Рисунок 3.15. Додавання зображення

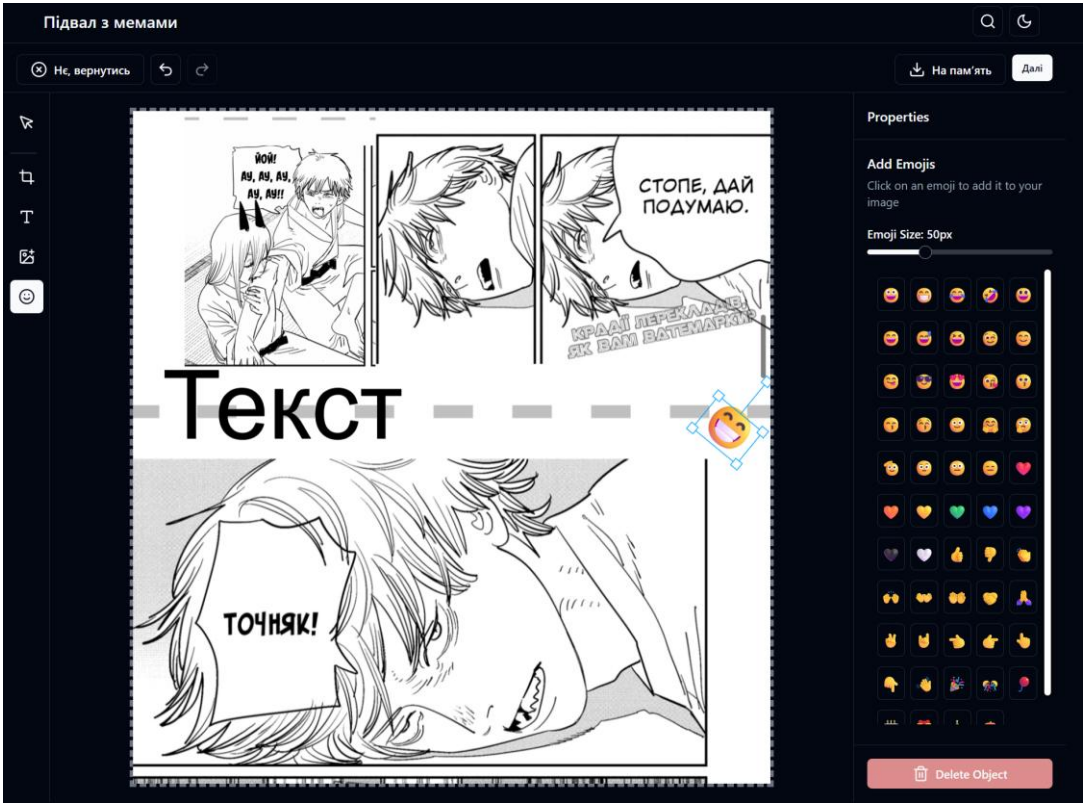


Рисунок 3.16. Додавання емоджі

На рисунку 3.17 можна побачити результат генерування зображення за допомогою штучного інтелекту.

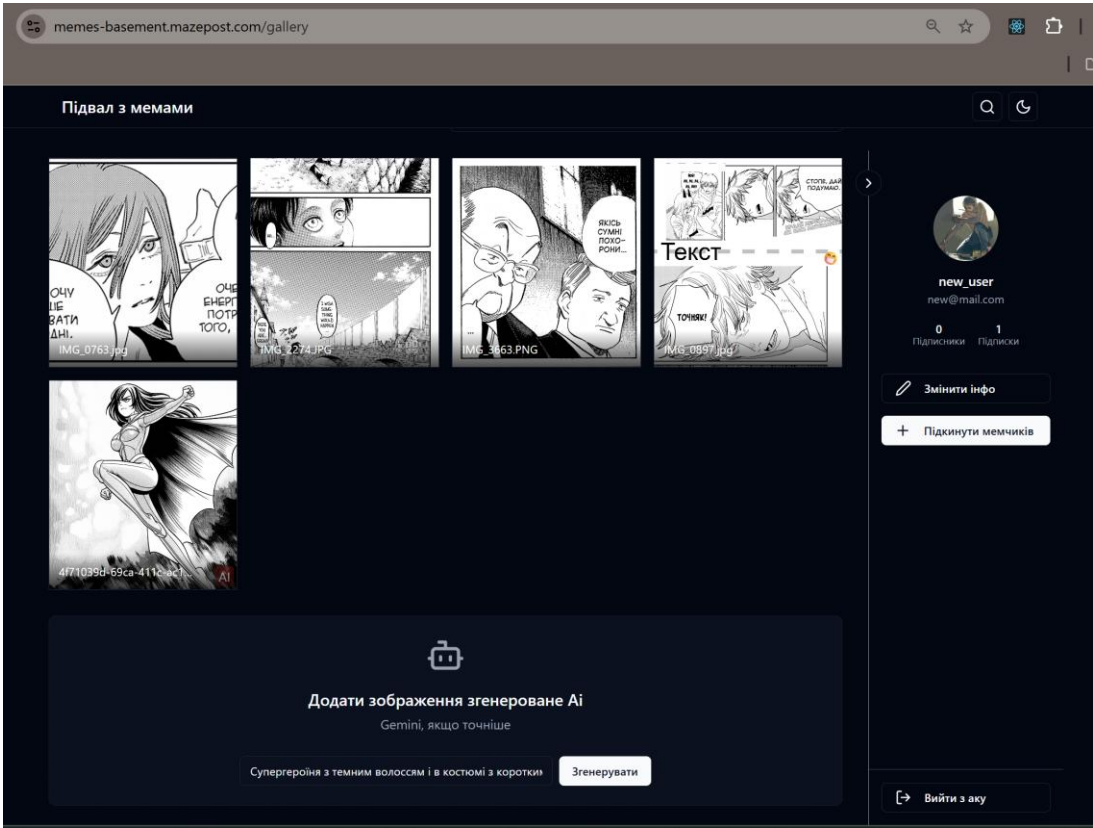


Рисунок 3.17. Зображення згенероване штучним інтелектом

Після заповнення опису та ключових слів користувач може опублікувати свої меми. Результат публікації можна побачити на рисунку 3.18.

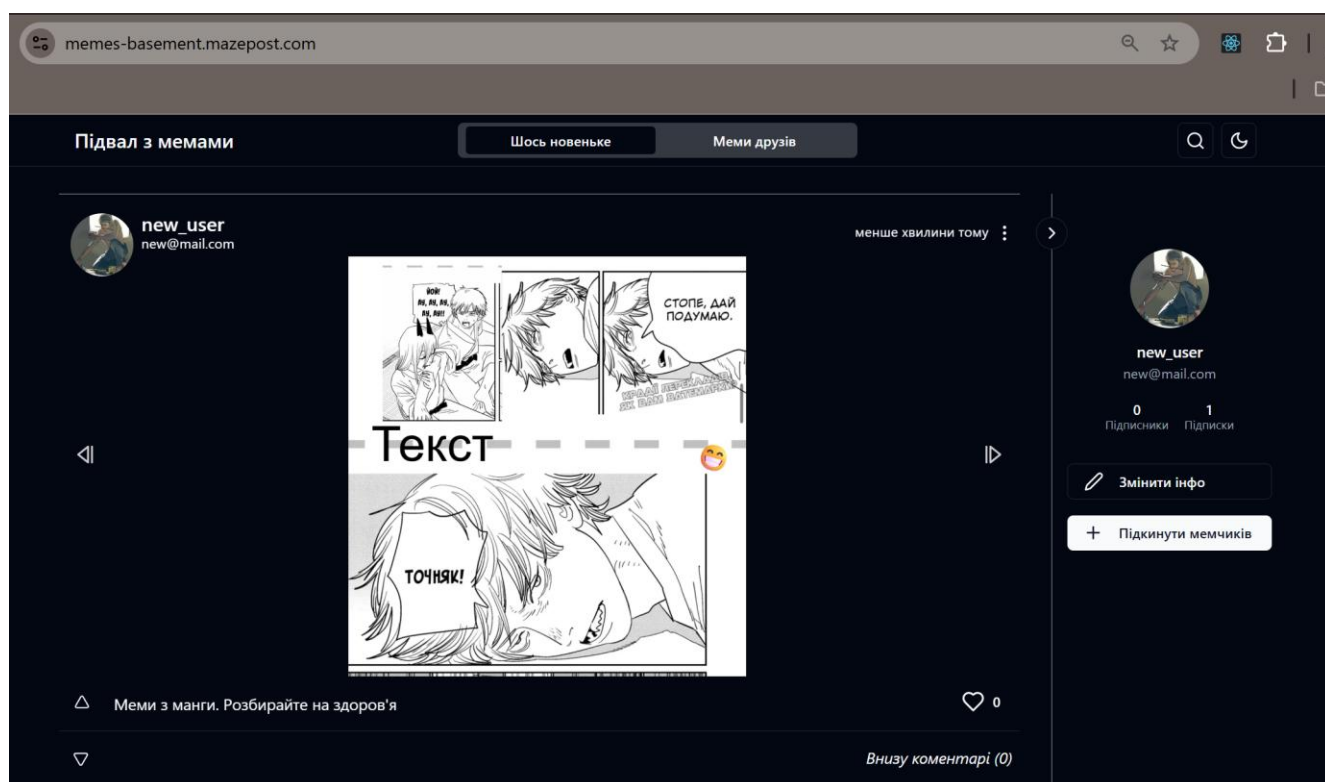


Рисунок 3.18. Опубліковані меми

3.3. Тестування роботи програмного продукту

Роботу платформи було протестовано за допомогою різних методів тестування, що дало змогу переконатися в її стабільності, коректності обробки запитів та відповідей, а також надійності взаємодії між клієнтською та серверною частинами. Серверну частину було доцільно перевіряти за допомогою менеджера запитів Postman, який дозволив надсилати запити до окремих ендпоінтів і переглядати відповіді у зручному форматі, аналізуючи правильність обробки введених даних.

Для спрощення та автоматизації роботи з захищеними ендпоінтами також був написаний спеціальний скрипт для авторизації. Він здійснює вхід користувача, зберігає отриманий токен в пам'яті Postman і автоматично підставляє його в заголовки наступних запитів (рис. 3.19).

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				52
Змн.	Арк.	№ докум.	Підпис	Дата		

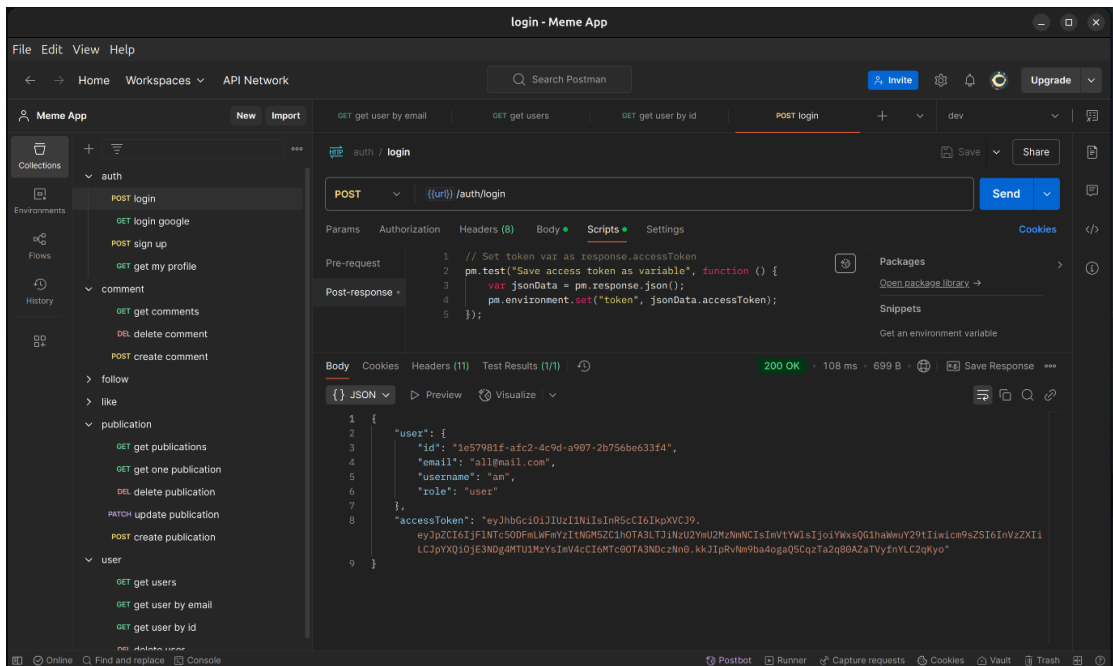


Рисунок 3.19. Скрипт для авторизації у Postman

Також були створені unit-тести за допомогою бібліотеки Jest для перевірки роботи модулів бекенду [26]. Результат роботи тестів можна побачити на рисунку 3.20.

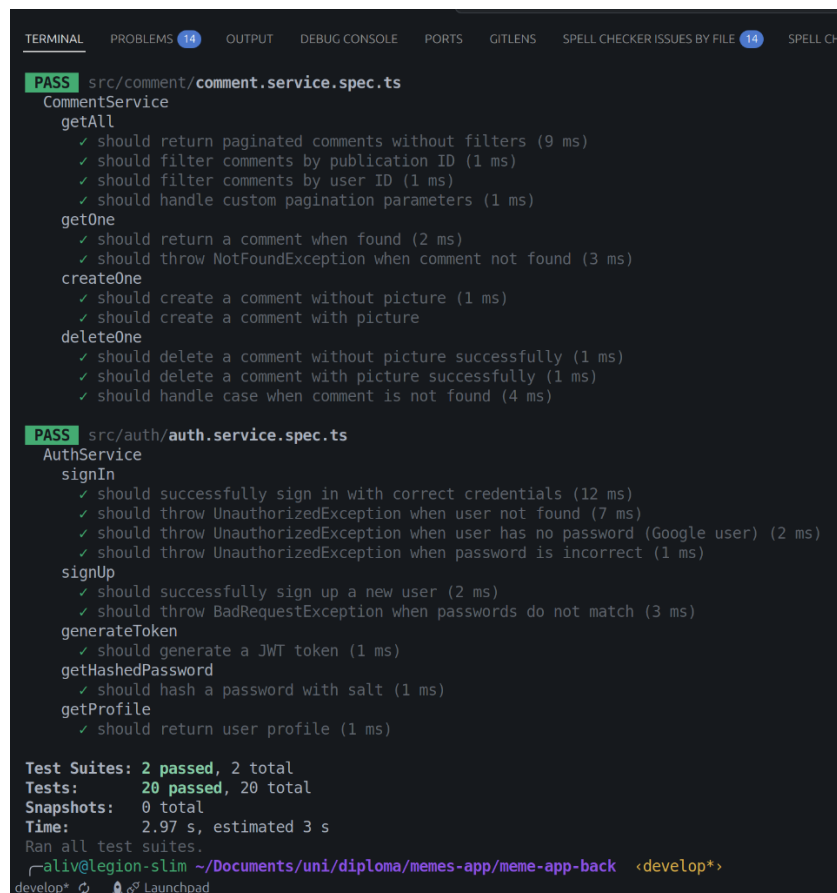


Рисунок 3.20. Результат запуску юніт тестів

		Радківка А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

Юніт-тести для CommentService перевіряють, як працює отримання, створення та видалення коментарів. Вони переконуються, що метод getAll повертає список коментарів із фільтрами та пагінацією, getOne шукає конкретний коментар, і якщо його немає – викидає виняток. Метод createOne перевіряється як для випадку без картинки, так і з переданим файлом, що передається у файл-сервіс. Метод deleteOne тестується для випадків, коли коментар є (з картинкою і без) і коли він відсутній.

У юніт-тестах AuthService перевіряється логіка авторизації. Вони тестують, як сервіс працює з введеними обліковими даними, порівнює паролі через bcrypt, створює користувача, генерує JWT-токен і обробляє помилки при невірних даних. Усі залежності, включно з UserService, JwtService та bcrypt, підміняються моками, щоб перевірити лише поведінку методу без справжньої логіки шифрування чи збереження.

Для перевірки клієнтської частини платформи було використане мануальне тестування. Воно включало в себе перевірки усіх основних флоу роботи вебсайту. Зокрема: авторизація, публікація мемів, редагування зображень, генерація мемів за допомогою ШІ, перегляд та пошук публікацій, коментування та вподобання публікації.

В результаті цього дослідження були виявлені та оброблені помилки, що виникали, щоб користувач міг побачити, що саме пішло не так.

Для усіх форм вебсайту були написані zod-схеми, що ефективно валідує усі поля та збирає помилки для їх відображення [27]. Приклад показу цих помилок при невдачі валідації можна побачити на сторінці реєстрації на рисунку 3.21.

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Рисунок 3.21. Помилки при заповненні форми реєстрації

Також були використані тости для показу серверних помилок, що обробляються у хуках TanStack Query. Розглянемо це на прикладі на рисунку 3.22.

```

onError: (err) => {
  console.error('API Error:', err);
  const errorMessage =
    err.response?.data.message.join(' ') || 'Щось пішло не так...';
  toast(errorMessage);
},

```

Рисунок 3.22. Обробка серверної помилки при невдачі публікації

Клієнт отримує серверну помилку і показує її повідомлення за допомогою тоста, якщо ж повідомлення немає, платформа лише повідомляє користувача, що сталася невідома помилка.

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

Окрім цього була здійснена перевірка сайту на продуктивність та швидкість роботи за допомогою Lighthouse. Перевірка головної сторінки зображена на рисунку 3.23, перевірка сторінки профілю – на рисунку 3.24.

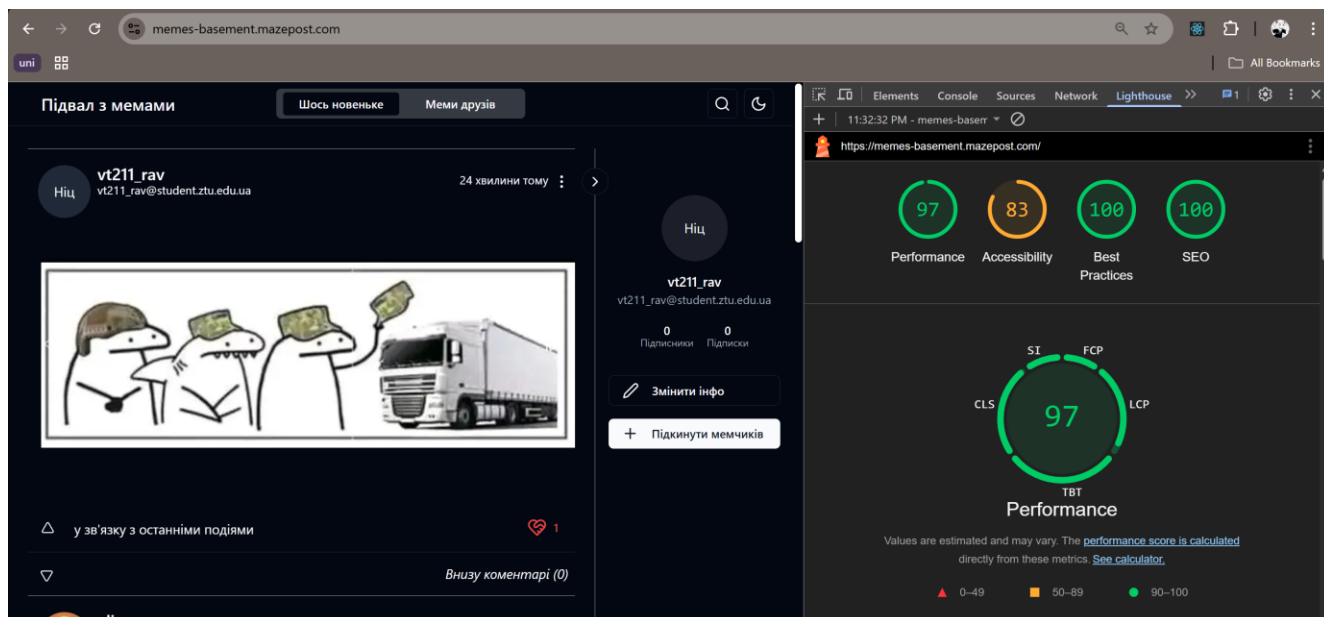


Рисунок 3.23. Перевірка головної сторінки за допомогою Lighthouse

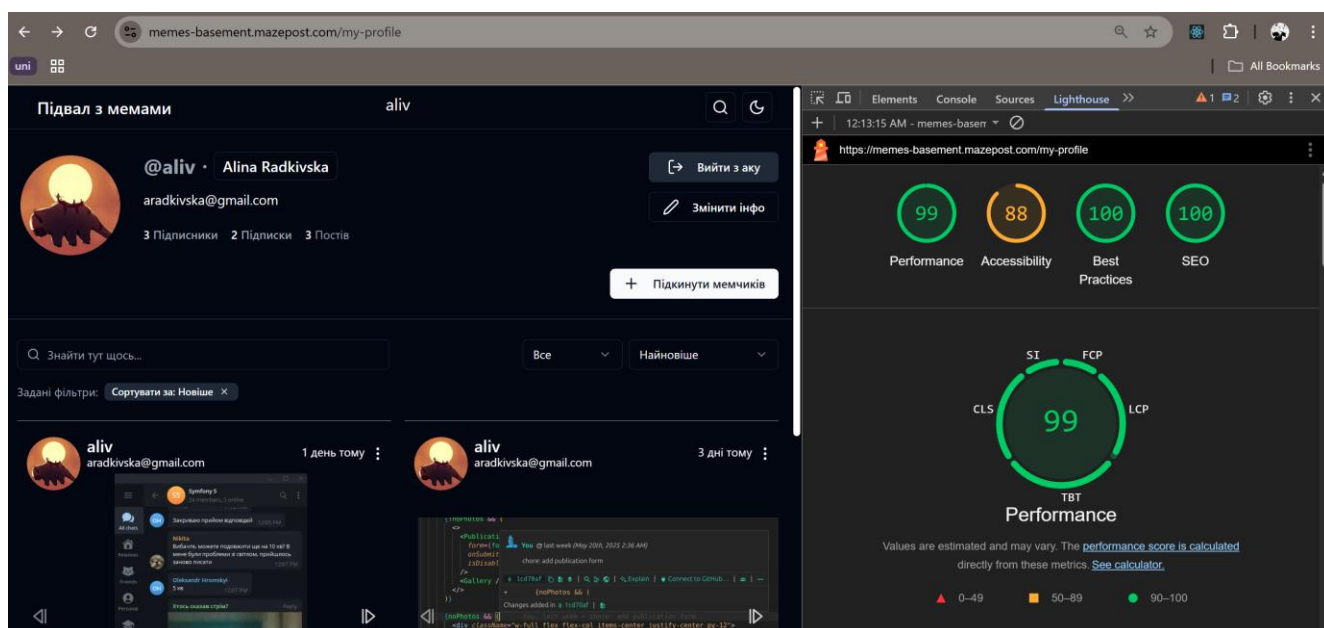


Рисунок 3.24. Помилки при заповненні форми реєстрації

Висновки до третього розділу

У третьому розділі було представлено повний цикл роботи платформи – від розгортання інфраструктури до тестування функціоналу. Розгортання здійснювалося за допомогою контейнеризації з використанням Docker, що

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк. 56
		Савицький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволило забезпечити стабільну та масштабовану роботу системи. Взаємодія між клієнтською частиною (Next.js), серверною логікою (NestJS), зовнішнім ШІ-сервісом та файловим сховищем реалізована через чітко структуровану мережу, що охоплює декілька регіонів розміщення серверів, оптимізуючи швидкість та надійність платформи.

Інтерфейс платформи продуманий для забезпечення інтуїтивної взаємодії з користувачем – від перегляду контенту до його створення, редагування й генерації за допомогою ШІ. Реалізовані механізми реєстрації, авторизації та соціальної взаємодії (вподобання, коментарі, підписки) створюють умови для динамічного користувацького досвіду.

Окрему увагу приділено тестуванню: використано як автоматизовані юніт-тести для серверної частини, так і мануальне тестування клієнтського інтерфейсу. Це дозволило виявити й усунути критичні помилки, а також перевірити продуктивність платформи за допомогою Lighthouse. Таким чином, реалізований підхід забезпечив надійність, зручність та високу якість взаємодії з платформою на всіх рівнях.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

В результаті написання випускної кваліфікаційної роботи бакалавра було виконано комплекс робіт, що охоплюють усі основні етапи створення вебплатформи для генерації, редагування та публікації мемів із використанням інструментів для розробки ПЗ. Кожен розділ дипломної роботи мав чітко визначені цілі, в результаті чого було реалізовано повноцінну клієнт-серверну систему з надійною архітектурою, інтуїтивним інтерфейсом та високим рівнем продуктивності.

У першому розділі було проведено всебічний аналіз аналогічних платформ, що дало змогу виокремити ключові функціональні вимоги до майбутньої системи. Особливу увагу було приділено можливості інтеграції ШІ-інструментів для генерації зображень та реалізації зручного візуального редактора мемів. На основі порівняльного аналізу технологій було обґрунтовано вибір клієнт-серверної архітектури, а також стеку технологій: Next.js – для фронтенду, NestJS – для бекенду, PostgreSQL – як основної системи управління базами даних. У цьому ж розділі сформовано архітектурну концепцію проєкту, визначено модульну організацію системи, засоби впровадження залежностей та взаємодії між окремими компонентами. Обґрунтовано вибір інструментальних засобів з урахуванням масштабованості, продуктивності та безпеки.

У другому розділі було зосереджено увагу на архітектурному та функціональному проєктуванні платформи. Розроблено UML-діаграми, які демонструють логіку роботи системи: діаграми випадків використання окреслюють типові сценарії взаємодії з платформою як для авторизованих користувачів, так і для гостей; класова діаграма відображає структуру об'єктної моделі, яка забезпечує гнучкість і масштабованість бази даних; діаграми активності дозволили формалізувати основні алгоритми платформи – реєстрацію, авторизацію, публікацію мемів, їх редагування, а також генерацію зображень за допомогою ШІ. У цьому розділі детально визначено функціональні обов'язки

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

окремих компонентів системи, забезпечено чітке логічне розмежування між сервісами та сформовано структуру, придатну до подальшого розширення.

У третьому розділі представлено реалізацію та впровадження платформи. Платформа була успішно розгорнута за допомогою контейнеризації з використанням Docker, що забезпечило незалежність від середовища розгортання, стабільність та масштабованість. Налагоджено взаємодію між фронтендом і бекендом, зовнішнім ШІ-сервісом для генерації зображень, а також файловим сховищем. Реалізовано повноцінний інтерфейс із можливістю реєстрації та авторизації користувачів, створення, редагування й перегляду мемів, залишення вподобань, коментарів і підписок, що формує елементи соціальної взаємодії. Здійснено тестування як серверної логіки (автоматизовані юніт-тести), так і клієнтської частини (мануальне тестування та Lighthouse-аналіз), що дозволило виявити та усунути критичні помилки, перевірити продуктивність та відповідність очікуванням користувачів.

Загалом, виконана робота демонструє системний підхід до розробки складної вебплатформи, що поєднує високий рівень технічної реалізації з орієнтацією на потреби кінцевого користувача. Платформа є технічно зрілим продуктом, що поєднує гнучкість, стабільність і зручність використання і відповідає всім поставленим вимогам на початку її розробки.

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				59
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Теза на тему «Підбір рекомендацій публікацій з використанням content-based та collaborative-based фільтрації» [електронний ресурс] – Режим доступу до ресурсу: <https://zcit.kubg.edu.ua/index.php/journal/issue/view/13/23>
2. Instagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.instagram.com/>
3. Reddit [Електронний ресурс] – Режим доступу до ресурсу: <https://www.reddit.com/>
4. 9GAG [Електронний ресурс] – Режим доступу до ресурсу: <https://9gag.com/>
5. Документація фреймворку Next.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs>
6. Документація фреймворку NestJS [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.nestjs.com/>
7. Предметно-Орієнтоване Проектування / Кондратюк С.В. Науковий керівник – Рудюк Л.В. [Електронний ресурс] – Режим доступу до ресурсу: <https://conf.ztu.edu.ua/wp-content/uploads/2019/02/47-1.pdf>
8. What Is a REST API? Examples, Uses, and Challenges / Postman Blog [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.postman.com/rest-api-examples/>
9. Документація до мови програмування TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>
10. Документація до бібліотеки TanStack Query v5 [Електронний ресурс] – Режим доступу до ресурсу: <https://tanstack.com/query/v5/docs/framework/react/overview>
11. Документація до бібліотеки Zustand [Електронний ресурс] – Режим доступу до ресурсу: <https://zustand.docs.pmnd.rs/getting-started/introduction>
12. Документація до бібліотеки Konva [Електронний ресурс] – Режим доступу до ресурсу: <https://konvajs.org/docs/index.html>

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

13. Документація до Gemini API [Електронний ресурс] – Режим доступу до ресурсу: <https://ai.google.dev/gemini-api/docs>
14. Документація до Postman [Електронний ресурс] – Режим доступу до ресурсу: <https://learning.postman.com/docs/introduction/overview/>
15. Документація до Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/desktop/>
16. Документація бази даних PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
17. Документація до ORM TypeORM [Електронний ресурс] – Режим доступу до ресурсу: <https://typeorm.io/>
18. Документація для використання Google OAuth[Електронний ресурс] – Режим доступу до ресурсу:
<https://developers.google.com/identity/protocols/oauth2>
19. Документація до сховища uploadthing [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.uploadthing.com/>
20. Документація до UI бібліотеки shadcn [Електронний ресурс] – Режим доступу до ресурсу: <https://ui.shadcn.com/docs/components>
21. Документація до хмари Coolify [Електронний ресурс] – Режим доступу до ресурсу: <https://coolify.io/docs/get-started/installation>
22. Документація до хмари Dokploy[Електронний ресурс] – Режим доступу до ресурсу: <https://docs.dokploy.com/docs/core>
23. Memes basement [Електронний ресурс] – Режим доступу до ресурсу: <https://memes-basement.mazepost.com/>
24. Memes basement API [Електронний ресурс] – Режим доступу до ресурсу: <https://memes-basement-api.mazepost.com/>
25. Memes basement AI Service [Електронний ресурс] – Режим доступу до ресурсу: <https://memes-basement-ai.mazepost.com/>
26. Документація до бібліотеки для тестування Jest [Електронний ресурс] – Режим доступу до ресурсу: <https://jestjs.io/uk/docs/getting-started>

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

27. Документація до бібліотеки zod [Електронний ресурс] – Режим доступу до ресурсу: <https://zod.dev/packages/zod>

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

ДОДАТКИ

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

Лістинг коду файлу stores/editor-store.tsx:

```
import { create } from 'zustand';

export type EditorMode =
  | 'select'
  | 'crop'
  | 'text'
  | 'image'
  | 'emoji'
  | 'draw'
  | 'filter';

export type TextAlign = 'left' | 'center' | 'right';
export type ObjectType = 'image' | 'text' | 'emoji' | 'drawing';

export interface EditorObject {
  id: string;
  type: ObjectType;
  x: number;
  y: number;
  width?: number;
  height?: number;
  draggable?: boolean;
  crop?: {
    x: number;
    y: number;
    width: number;
    height: number;
  };
};
[key: string]: number | string | boolean | undefined | object;
}

export interface CropConfig {
  x: number;
  y: number;
  width: number;
  height: number;
}

export interface TextOptions {
  fontSize: number;
  fontFamily: string;
  fill: string;
  align: TextAlign;
}

export interface EditorState {
  // Mode and selection
  mode: EditorMode;
  setMode: (mode: EditorMode) => void;

  // Objects management
  objects: EditorObject[];
  addObject: (object: EditorObject) => void;
  updateObject: (id: string, changes: Partial<EditorObject>) => void;
  removeObject: (id: string) => void;
}
```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

// Selection management
selectedObjectId: string | null;
setSelectedObjectId: (id: string | null) => void;

// Text options
textOptions: TextOptions;
setTextOptions: (options: Partial<TextOptions>) => void;

// Crop functionality
isCropping: boolean;
setIsCropping: (cropping: boolean) => void;
cropConfig: CropConfig | null;
setCropConfig: (config: CropConfig | null) => void;

// History management
// eslint-disable-next-line @typescript-eslint/no-explicit-any
history: any;
currentHistoryIndex: number;
addHistory: () => void;
undo: () => void;
redo: () => void;

// Canvas settings
canvasWidth: number;
canvasHeight: number;
setCanvasSize: (width?: number, height?: number) => void;

// Clear all objects
clearObjects: () => void;
}

export const useEditorStore = create<EditorState>((set, get) => ({
  // Mode and selection
  mode: 'select',
  setMode: (mode) => set({ mode }),

  // Objects management
  objects: [],
  addObject: (object) => {
    set((state) => ({
      objects: [...state.objects, object],
      selectedObjectId: object.id,
    }));
    get().addHistory();
  },
  updateObject: (id, changes) => {
    set((state) => ({
      objects: state.objects.map((obj) =>
        obj.id === id ? { ...obj, ...changes } : obj
      ),
    }));
    get().addHistory();
  },
  removeObject: (id) => {
    set((state) => ({
      objects: state.objects.filter((obj) => obj.id !== id),
      selectedObjectId:

```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        state.selectedObjectId === id ? null : state.selectedObjectId,
    ));
    get().addHistory();
},

// Selection management
selectedObjectId: null,
setSelectedObjectId: (id) => set({ selectedObjectId: id }),

// Text options
textOptions: {
    fontSize: 24,
    fontFamily: 'Arial',
    fill: '#000000',
    align: 'left' as TextAlign,
},
setTextOptions: (options) =>
    set((state) => ({
        textOptions: {
            ...state.textOptions,
            ...options,
        },
    }))),

// Crop functionality
isCropping: false,
setIsCropping: (cropping) => set({ isCropping: cropping }),
cropConfig: null,
setCropConfig: (config) => set({ cropConfig: config }),

// History management
history: [],
currentHistoryIndex: -1,
addHistory: () => {
    const state = get();
    const historyCopy = [...state.history].slice(
        0,
        state.currentHistoryIndex + 1
    );
    const newHistory = [
        ...historyCopy,
        { objects: JSON.parse(JSON.stringify(state.objects)) },
    ];
    set({
        history: newHistory,
        currentHistoryIndex: newHistory.length - 1,
    });
},
undo: () => {
    const { currentHistoryIndex, history } = get();
    if (currentHistoryIndex > 0) {
        const newIndex = currentHistoryIndex - 1;
        const historyState = history[newIndex];
        set({
            currentHistoryIndex: newIndex,
            objects: historyState.objects,
        });
    }
}

```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    },
    redo: () => {
      const { currentHistoryIndex, history } = get();
      if (currentHistoryIndex < history.length - 1) {
        const newIndex = currentHistoryIndex + 1;
        const historyState = history[newIndex];
        set({
          currentHistoryIndex: newIndex,
          objects: historyState.objects,
        });
      }
    },
  },

  // Canvas settings
  canvasWidth: 0,
  canvasHeight: 0,
  setCanvasSize: (width, height) =>
    set({
      canvasWidth: width ? width : 0,
      canvasHeight: height ? height : 0,
    }),

  // Clear all objects
  clearObjects: () => {
    set({ objects: [], canvasHeight: 0, canvasWidth: 0 });
    get().addHistory();
  },
}));

```

Лістинг коду фійлу stores/photo-store.tsx:

```

import { create } from 'zustand';

export interface Photo {
  id: string;
  file: File;
  preview: string;
  name: string;
  width: number;
  height: number;
  edited: boolean;
  ai: boolean;
}

interface PhotoState {
  photos: Photo[];
  addPhotos: (photos: Photo[]) => void;
  updatePhoto: (id: string, file: File, preview: string) => void;
  removePhoto: (id: string) => void;
  getPhoto: (id: string) => Photo | undefined;
}

const getPhotoMeta = async (
  url: string
): Promise<{ naturalHeight: number; naturalWidth: number }> =>
  new Promise((resolve, reject) => {
    const img = new Image();

```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    img.onload = () => resolve(img);
    img.onerror = (err) => reject(err);
    img.src = url;
  });

export const usePhotoStore = create<PhotoState>((set, get) => ({
  photos: [],
  addPhotos: async (newPhotos) => {
    const updatedPhotos = await Promise.all(
      newPhotos.map(async (photo) => {
        const res = await getPhotoMeta(photo.preview);
        return {
          ...photo,
          height: res.naturalHeight,
          width: res.naturalWidth,
        };
      })
    );

    set((state) => ({
      photos: [...state.photos, ...updatedPhotos],
    })));
  },
  updatePhoto: async (id, file, preview) => {
    const res = await getPhotoMeta(preview);

    set((state) => {
      const updatedPhotos = state.photos.map((photo) =>
        photo.id === id
        ? {
            ...photo,
            file,
            preview,
            height: res.naturalHeight,
            width: res.naturalWidth,
          }
        : photo
      );

      return { photos: [...updatedPhotos] };
    });
  },
  removePhoto: (id) =>
    set((state) => {
      const toRemove = state.photos.find((p) => p.id === id);
      if (toRemove) {
        URL.revokeObjectURL(toRemove.preview);
      }
      return {
        photos: state.photos.filter((photo) => photo.id !== id),
      };
    }),
  getPhoto: (id) => get().photos.find((photo) => photo.id === id),
})));

```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				68
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг коду з файлу server.ts:

```
import express from 'express';
import cors from 'cors';
import { GoogleGenAI } from '@google/genai';

const app = express();
const port = process.env.PORT || 3000;

const allowedOrigins = process.env.CORS_ALLOWED_ORIGINS
  ? process.env.CORS_ALLOWED_ORIGINS.split(',')
  : ['http://localhost:3000', 'http://localhost:3001'];

app.use(
  cors({
    origin: allowedOrigins,
  })
);

app.use(express.json());

const ai = new GoogleGenAI({
  apiKey: process.env.GEMINI_API_KEY,
});

interface AiImageResponse {
  mimeType: string;
  base64: string;
}

app.post('/api/ai-image', async (req, res) => {
  try {
    const { prompt }: { prompt: string } = req.body || {};

    if (!prompt) {
      res.status(400).json({ error: 'Prompt is required' });
      return;
    }

    const config = {
      responseModalities: ['IMAGE', 'TEXT'],
      responseMimeType: 'text/plain',
    };

    const model = 'gemini-2.0-flash-preview-image-generation';
    const contents = [
      {
        role: 'user',
        parts: [
          {
            data: '',
            text: `Generate an image according to this description: ${prompt}`,
          },
        ],
      },
    ];
  }
});
```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const response = await ai.models.generateContentStream({
  model,
  config,
  contents,
});

for await (const chunk of response) {
  const part = chunk?.candidates?.[0]?.content?.parts?.[0];
  const inlineData = part?.inlineData;

  if (inlineData?.data && inlineData?.mimeType) {
    res.json({
      mimeType: inlineData.mimeType,
      base64: inlineData.data,
    } as AiImageResponse);
    return;
  }
}

res.status(500).json({ error: 'No image data returned' });
return;
} catch (error) {
  console.error(error);
  res.status(500).json({ error: 'Image generation failed' });
  return;
}
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});

```

Додаток В

Лістинг коду з файлу ai-server/Dockerfile:

```

FROM node:22.14.0-alpine AS base

# Install dependencies only when needed
FROM base AS deps
# Check https://github.com/nodejs/docker-
node/tree/b4117f9333da4138b03a546ec926ef50a31506c3#nodealpine to understand why libc6-
compat might be needed.
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Install dependencies based on the preferred package manager
COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* .npmrc* ./
RUN \
  if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
  elif [ -f package-lock.json ]; then npm ci; \
  elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm && pnpm i --frozen-lockfile; \
  else echo "Lockfile not found." && exit 1; \
  fi

# Rebuild the source code only when needed

```

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

```

FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

RUN \
    if [ -f yarn.lock ]; then yarn run build; \
    elif [ -f package-lock.json ]; then npm run build; \
    elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm && pnpm run build; \
    else echo "Lockfile not found." && exit 1; \
    fi

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV=production

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 server-ai

COPY --from=builder --chown=server-ai:nodejs /app/dist ./
COPY --from=builder --chown=server-ai:nodejs /app/node_modules ./node_modules

USER server-ai

EXPOSE 3000

ENV PORT=3000

ENV HOSTNAME="0.0.0.0"
CMD ["node", "server.js"]

```

Лістинг коду з файлу meme-app-back/Dockerfile:

```

FROM node:22.14.0-alpine AS base

# Install dependencies only when needed
FROM base AS deps
# Check https://github.com/nodejs/docker-
# node/tree/b4117f9333da4138b03a546ec926ef50a31506c3#nodealpine to understand why libc6-
# compat might be needed.
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Install dependencies based on the preferred package manager
COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* .npmrc* ./
RUN \
    if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
    elif [ -f package-lock.json ]; then npm ci; \
    elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm && pnpm i --frozen-lockfile; \
    else echo "Lockfile not found." && exit 1; \
    fi

# Rebuild the source code only when needed

```

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

```

FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

RUN \
    if [ -f yarn.lock ]; then yarn run build; \
    elif [ -f package-lock.json ]; then npm run build; \
    elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm && pnpm run build; \
    else echo "Lockfile not found." && exit 1; \
    fi

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV=production

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nestjs

COPY --from=builder --chown=nestjs:nodejs /app/dist ./
COPY --from=builder --chown=nestjs:nodejs /app/node_modules ./node_modules

USER nestjs

EXPOSE 3000

ENV PORT=3000

ENV HOSTNAME="0.0.0.0"
CMD ["node", "src/main.js"]

```

Лістинг коду з файлу meme-app-front/Dockerfile:

```

FROM node:22.14.0-alpine AS base

# Install dependencies only when needed
FROM base AS deps
# Check https://github.com/nodejs/docker-
# node/tree/b4117f9333da4138b03a546ec926ef50a31506c3#nodealpine to understand why libc6-
# compat might be needed.
# Install build dependencies for native modules like canvas
RUN apk add --no-cache \
    libc6-compat \
    python3 \
    py3-pip \
    make \
    g++ \
    cairo-dev \
    jpeg-dev \
    pango-dev \
    musl-dev \
    giflib-dev \
    pixman-dev \
    pangomm-dev \
    libjpeg-turbo-dev \

```

		Радківська А.В.			Житомирська політехніка. 25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    freetype-dev && \
    ln -sf python3 /usr/bin/python
WORKDIR /app

# Install dependencies based on the preferred package manager
COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* .npmrc* ./
RUN \
    if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
    elif [ -f package-lock.json ]; then npm ci; \
    elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm && pnpm i --frozen-lockfile; \
    else echo "Lockfile not found." && exit 1; \
    fi

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Next.js collects completely anonymous telemetry data about general usage.
# Learn more here: https://nextjs.org/telemetry
# Uncomment the following line in case you want to disable telemetry during the build.
ENV NEXT_TELEMETRY_DISABLED=1

RUN \
    if [ -f yarn.lock ]; then yarn run build; \
    elif [ -f package-lock.json ]; then npm run build; \
    elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm && pnpm run build; \
    else echo "Lockfile not found." && exit 1; \
    fi

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV=production
# Uncomment the following line in case you want to disable telemetry during runtime.
ENV NEXT_TELEMETRY_DISABLED=1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder --chown=nextjs:nodejs /app/package.json /app/package-lock.json ./
COPY --from=builder --chown=nextjs:nodejs /app/.next /app/.next
COPY --from=builder --chown=nextjs:nodejs /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/next.config.ts /app/
COPY --from=builder --chown=nextjs:nodejs /app/node_modules ./node_modules

USER nextjs

EXPOSE 3001

ENV PORT=3001

# server.js is created by next build from the standalone output
# https://nextjs.org/docs/pages/api-reference/config/next-config-js/output

```

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савицький Р.С.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

ENV HOSTNAME="0.0.0.0"
CMD ["npx", "next", "start"]

		Радківська А.В.			Житомирська політехніка.25.121.21.000 - ПЗ	Арк.
		Савіцький Р.С.				74
Змн.	Арк.	№ докум.	Підпис	Дата		