Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

ЗАСТОСУВАННЯ МОВНИХ МОДЕЛЕЙ ДЛЯ РЕДАГУВАННЯ ЗОБРАЖЕНЬ І ГЕНЕРАЦІЇ ВІДЕО
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-2
Мартиненко Є. В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Гороховатський В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Мартиненку Євгенію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Застосування мовних моделей для редагування зображень і генерації відео

затверджена наказом університету від 26 травня 2026 року № 435Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 27 червня 2026 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали міжнародних конференцій, офіційна документація, дані інтернет-мережі, відкриті API-інтерфейси та моделі генеративного штучного інтелекту платформи Hugging Face, зокрема моделі сімейств Stable Diffusion, Veo, Sora та AnimateDiff, а також інструментальні засоби розробки вебзастосунків, а саме Python-фреймворк FastAPI, JavaScript-бібліотека React, реляційна база даних SQLite, інструменти Docker Compose та засоби JWT-автентифікації.

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз засад застосування мовних і мультимодальних моделей для створення та редагування медіаконтенту.

2. Обґрунтування архітектури вебзастосунку MediaGenerator та вибору інструментальних засобів розробки, зокрема FastAPI, React та SQLite.

3. Дослідження принципів взаємодії з генеративними сервісами через API та розробка підсистеми постачальників сервісів.

4. Програмна реалізація клієнтської та серверної частин вебзастосунку для обробки природномовних інструкцій користувача.

5. Тестування функціональних можливостей розробленого продукту щодо генерації зображень, створення розкадрувань та формування відео.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми обробки зображень та генерації відео, постановка задачі, взаємодія мовної моделі з генеративними сервісами, еволюція підходів до генерації зображень, відмінності між генерацією і редагуванням, сценарії генерації відео за текстовим описом та на основі початкового зображення, функціональні напрями, загальна архітектура вебзастосунку MediaGenerator, структура репозиторію та запуск застосунку, робочий процес мовної моделі, алгоритм виконання відеозадачі через зовнішній сервіс, ілюстрація роботи застосунку, основні положення та результати роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-16.04.26	
3	Аналіз літератури з проблеми, що вирішується	17.04.26-19.04.26	
4	Аналіз існуючих методів розпізнавання	20.04.26-22.04.26	
5	Формування кроків вирішення проблеми	23.04.26-05.05.26	
6	Програмна реалізація	26.04.26-20.05.26	
7	Аналіз отриманих результатів	21.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Гороховатський В. О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 85 с., 8 табл., 16 рис., 48 джерел.

ВЕБЗАСТОСУНОК, ГЕНЕРАТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, ГЕНЕРАЦІЯ ВІДЕО, ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ, ІНЖЕНЕРІЯ ЗАПИТІВ, МОВНА МОДЕЛЬ, РЕДАГУВАННЯ ЗОБРАЖЕНЬ, РОЗКАДРУВАННЯ, REACT.

Об’єктом роботи є вебзастосунок для створення, редагування та підготовки відеомедіа з використанням мовних моделей і генеративних сервісів.

Метою роботи є розроблення MediaGenerator для роботи із зображеннями й відео через природномовні інструкції користувача.

Застосунок підтримує генерацію та редагування зображень, аналіз зображень, розкадрування, створення відео за текстом або зображенням, автентифікацію і локальний режим Test.

У роботі проаналізовано засади застосування мовних і мультимодальних моделей, обґрунтовано архітектуру React, FastAPI, SQLite, JWT, ряд постачальників сервісів, Hugging Face, Stability AI та Test. Результатом є вебзастосунок, у якому текстова інструкція використовується для створення або редагування зображення, підготовки розкадрування і запуску відеозадачі.

GENERATIVE ARTIFICIAL INTELLIGENCE, IMAGE EDITING, IMAGE GENERATION, LANGUAGE MODEL, PROMPT ENGINEERING, REACT, STORYBOARDING, VIDEO GENERATION, WEB APPLICATION.

The object of the work is a web application for creating, editing, and preparing video media using language models and generative services.

The aim of the work is to develop MediaGenerator for working with images and videos through natural language user instructions.

The application supports image generation and editing, image analysis, storyboarding, text-to-video or image-to-video creation, authentication, and a local Test mode.

The work analyzes the principles of applying language and multimodal models, and substantiates the architecture based on React, FastAPI, SQLite, JWT, a number of service providers, Hugging Face, Stability AI, and Test. The result is a web application in which a text instruction is used to create or edit an image, prepare a storyboard, and launch a video task.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Постачальник сервісу – програмний адаптер, через який застосунок звертається до зовнішньої чи локальної моделі

Розкадрування – структурований опис майбутнього відео у вигляді сцен, руху камери та коротких підказок

Текстовий запит – інструкція користувача до мовної або генеративної моделі

ШІ – штучний інтелект

API – Application Programming Interface (інтерфейс прикладного програмування)

Docker Compose – інструмент для локального запуску взаємопов’язаних сервісів

FastAPI – Python-фреймворк для створення HTTP API

Image-to-image – створення або зміна зображення на основі початкового зображення та інструкції

Image-to-video – створення відео на основі початкового зображення та опису руху

JWT – JSON Web Token (формат токена для передавання даних про автентифікованого користувача)

LLM – Large Language Model (велика мовна модель для обробки природномовних інструкцій)

React – бібліотека для побудови клієнтського інтерфейсу

SQLite – файлова реляційна база даних для користувачів і журналу операцій

Text-to-image – створення зображення за текстовим описом

Text-to-video – створення відео за текстовим описом без початкового кадру

VLM – Vision-Language Model (візуально-мовна модель, що зіставляє зображення і текст)

ВСТУП

Генеративні засоби роботи із зображеннями та відео дедалі частіше використовуються не як окремі редактори, а як вебзастосунки, у яких основною формою керування є природномовний опис. Користувач формулює задум звичайним текстом, а програмна система перетворює його на уточнений запит, параметри моделі, звернення до зовнішнього сервісу або послідовність проміжних дій. Такий підхід змінює роль інтерфейсу: замість ручного вибору великої кількості інструментів людина описує бажаний результат, уточнює його та оцінює отриманий медіаматеріал [1–5].

Актуальність теми зумовлена тим, що сучасні медіасервіси мають поєднувати кілька різних класів моделей. Для створення зображення потрібна генеративна модель, для редагування – механізм зміни вже наявного файлу, для аналізу – візуально-мовна інтерпретація, для відео – сервіс, здатний підтримувати рух і часову узгодженість. Мовна модель у такій системі не замінює всі інші компоненти, але допомагає зв’язати їх у зрозумілий робочий процес: уточнити запит, сформувати план редагування, описати розкадрування або підготувати інструкцію руху для відеозадачі [6–8].

Практична складність полягає в тому, що користувацькі інструкції часто є короткими та неоднозначними. Наприклад, прохання «зробити сцену кінематографічною» не визначає ракурс, освітлення, колірну гаму, композицію та обмеження щодо збереження головного об’єкта. Для зображення таку інструкцію потрібно деталізувати, для редагування – пов’язати з конкретним вхідним файлом, а для відео – доповнити рухом камери, тривалістю, співвідношенням сторін і можливостями обраного постачальника сервісу. Через це важливою стає не лише наявність генеративної моделі, а й програмна організація всього сценарію роботи.

Робота присвячена розробленню вебзастосунку MediaGenerator. Це прикладний програмний артефакт із клієнтською частиною на React і серверною частиною на FastAPI. Застосунок підтримує реєстрацію та вхід користувачів,

ролі звичайного користувача й адміністратора, генерацію зображень, редагування завантажених зображень, покращення текстових запитів, аналіз і оцінювання зображення, побудову плану редагування, формування розкадрування, генерацію відео за текстовим описом або на основі початкового зображення, опитування статусу відеозадачі, локальну історію дій і журнал операцій у SQLite [9–12].

Окреме місце в роботі займає вибір постачальника сервісу. MediaGenerator працює не з однією жорстко прив’язаною моделлю: для зображень можуть використовуватися Stability AI, Hugging Face Inference Providers або локальний режим Test, а для відео – Runway, Fal, Kinovi, Hugging Face або Test. Така організація потрібна через залежність зовнішніх сервісів від токенів, квот, доступності моделей і мережевих умов. Локальний режим Test не замінює реальну генерацію, але дає змогу перевірити власну логіку застосунку без витрачання зовнішніх кредитів.

У першому розділі розглянуто предметну область, теоретичні засади генерації зображень, природномовного редагування й відеогенерації та сформульовано постановку задачі. У другому розділі обґрунтовано архітектуру та технологічні рішення MediaGenerator. У третьому розділі описано практичну реалізацію застосунку, основні сценарії роботи, перевірку працездатності та результати.

1 ЗАСТОСУВАННЯ МОВНИХ МОДЕЛЕЙ ДЛЯ РЕДАГУВАННЯ ЗОБРАЖЕНЬ І ГЕНЕРАЦІЇ ВІДЕО

1.1 Мовні моделі та взаємодія на основі текстових запитів

Робота із цифровими зображеннями тривалий час спиралася переважно на класичні методи комп'ютерного зору: виділення ознак, опис контурів, пошук ключових точок, порівняння дескрипторів, класифікацію і статистичний аналіз візуальних даних [1–3]. Такі підходи залишаються важливими, оскільки пояснюють, як зображення можна подати у вигляді набору характеристик і як ці характеристики використовуються для розпізнавання або порівняння [4–5]. Проте сучасні генеративні системи змістили акцент із пасивного аналізу готового матеріалу на створення й зміну медіаконтенту за описом користувача.

У традиційному графічному редакторі користувач безпосередньо керує інструментами: вибирає шар, маску, пензель, фільтр, корекцію кольору, режим трансформації або параметри виділення. Такий підхід дає високий контроль, але потребує від людини знання інструментів і послідовності дій. У генеративній медіасистемі значна частина керування переноситься в текстовий опис. Користувач формулює задум природною мовою, а програмна система перетворює цей опис на структурований запит до моделі або постачальника сервісу.

Природномовне керування не означає, що система «розуміє» зображення так само, як людина. Модель опрацьовує текст і візуальні дані на основі статистичних зв'язків, засвоєних під час навчання. Однак для прикладної системи важливим є не філософське тлумачення цього процесу, а практичний результат: користувач може описати сцену, стиль, об'єкти, світло, композицію або бажаний рух без ручного налаштування десятків параметрів. Саме це робить мовні моделі корисними у вебзастосунках для генерації та редагування медіа.

Текстовий запит у таких системах виконує роль короткої специфікації. У ньому можуть поєднуватися предметний рівень, наприклад головний об'єкт або

середовище; композиційний рівень, наприклад ракурс, план, освітлення; стилістичний рівень, наприклад фотореалізм, ілюстративність або рекламна подача; технічний рівень, наприклад співвідношення сторін, обмеження щодо небажаних елементів або вимога зберегти основний об’єкт. Якщо ці вимоги подані надто стисло, генеративна модель може надати перевагу найтиповішим асоціаціям і втратити важливі деталі.

Мовна модель у генеративній медіасистемі доцільна саме як проміжний шар між задумом користувача і спеціалізованою моделлю створення медіа. Вона може уточнити запит, виділити ключові обмеження, переформулювати інструкцію редагування, підготувати опис руху для відео або розкласти складну ідею на послідовність сцен. У цьому полягає її відмінність від моделі, яка безпосередньо синтезує пікселі чи кадри: мовна модель переважно працює зі змістом інструкції, а не з фінальним візуальним результатом.

Узагальнену роль мовної моделі як проміжного шару між задумом користувача і спеціалізованими сервісами генеративних медіа показано на рисунку 1.1.

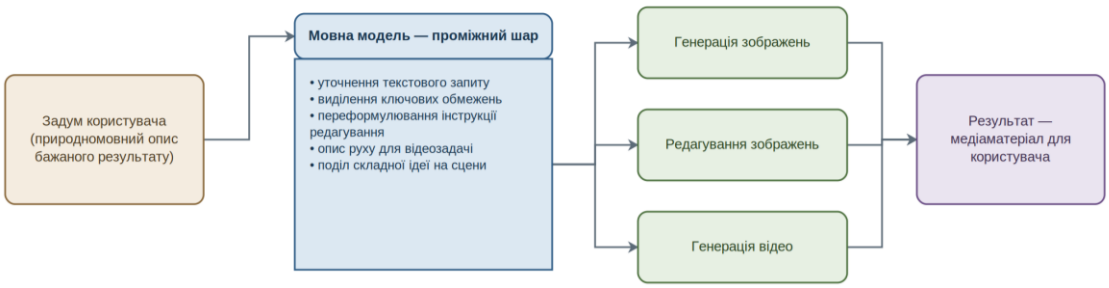

Рисунок 1.1 – Мовна модель як проміжний шар між задумом користувача і спеціалізованими генеративними сервісами

Розвиток генеративних моделей проходив кілька помітних етапів. Генеративно-змагальні мережі показали, що зображення можна синтезувати через взаємодію генератора і дискримінатора [6]. Архітектура Transformer стала основою для ефективної роботи з послідовностями та контекстом [7]. Великі мовні моделі продемонстрували здатність опрацьовувати інструкції, уточнювати

умови задачі й виконувати різні типи текстових перетворень без окремого навчання під кожну вузьку задачу [8]. У медіасистемах ці напрями поєднуються: текстова інструкція задає намір, а спеціалізована генеративна модель створює або змінює зображення.

Одним із математичних механізмів, що пояснює роботу Transformer-архітектури з контекстом, є масштабована скалярна увага. У загальному вигляді її можна подати формулою:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (1.1)$$

де Q – матриця запитів;

K – матриця ключів;

V – матриця значень;

d_k – розмірність векторів ключів, причому $d_k > 0$, $\sqrt{d_k}$ потрібний для зменшення масштабу скалярних добутків перед застосуванням softmax і стабілізації значень;

softmax – функція нормалізації, що перетворює оцінки на ваги.

У медіасистемах такого типу особливо важливим є поєднання мовної моделі з візуальними й відеогенеративними сервісами. Мовна модель допомагає сформулювати точний опис, а візуальна модель або зовнішній сервіс виконує перетворення. Такий поділ ролей відповідає практиці розроблення прикладних вебзастосунків: система інтегрує доступні сервіси, забезпечує для них зручний інтерфейс, перевірку параметрів, обробку помилок і зрозумілий результат для користувача.

Для роботи з такими системами суттєвим стає не тільки зміст запиту, а й спосіб його подання. У короткому запиті часто змішуються мета, стиль, технічні параметри й очікуваний результат. Користувач може написати «зробити якісніше», але для системи цього недостатньо: потрібно зрозуміти, чи йдеться про деталізацію об’єкта, корекцію освітлення, зміну фону, усунення артефактів

або підготовку зображення до відео. Через це мовна модель має працювати як засіб нормалізації наміру. Вона перетворює загальну ідею на текст, який краще відповідає очікуванням генеративного сервісу.

Перевагою природномовного підходу є його доступність. Користувачу не потрібно знати, як саме працює дифузійна модель або які параметри має конкретний сервіс. Він описує бажаний результат, а система допомагає зробити цей опис придатним для генерації. Водночас природномовний інтерфейс має обмеження. Одна й та сама фраза може допускати кілька трактувань, а надмірно загальні слова, наприклад «красиво», «реалістично» або «кінематографічно», не задають точних умов. Тому в якісній системі текстовий запит бажано не просто передавати далі, а уточнювати й структурувати.

У межах розроблення вебзастосунку це означає, що система повинна підтримувати не одну кнопку генерації, а пов’язаний робочий процес. Користувач вводить опис, за потреби покращує його, обирає сервіс, отримує результат, оцінює його, редагує або використовує як основу для відео. Такий сценарій відрізняється від простого виклику окремої моделі тим, що програмна частина відповідає за організацію дій, а не лише за передачу одного запиту.

1.2 Основні підходи до генерації і редагування зображень

Генерація зображень за текстовим описом є одним із найпоширеніших напрямів застосування генеративного штучного інтелекту. Її сутність полягає в тому, що користувач подає текстову інструкцію, а модель створює нове зображення, яке має відповідати змісту опису. На відміну від класичного пошуку за зображеннями, система не вибирає готовий файл із набору даних, а формує новий візуальний артефакт.

Ранні генеративні підходи значною мірою спиралися на генеративно-змагальні мережі. Вони дали змогу отримувати переконливі зображення, але складно керувалися докладним текстовим описом і могли бути нестабільними

під час навчання. Подальший розвиток пов’язаний із моделями, які краще поєднують текстові та візуальні представлення. Важливим кроком стала поява підходів, у яких текст і зображення навчаються у спільному просторі ознак, як у CLIP [9]. Така ідея дала змогу точніше пов’язувати словесний опис із візуальним результатом.

Сучасні системи генерації зображень часто ґрунтуються на дифузійних моделях [10, 11]. Їхня загальна ідея полягає у поступовому перетворенні шуму на зображення під керуванням текстового опису та інших умов. Для користувача цей процес прихований, але його наслідки помітні: добре сформульований опис може впливати на об’єкт, стиль, освітлення, кольори й композицію. Латентні дифузійні моделі, зокрема підхід Latent Diffusion, дали змогу виконувати генерацію ефективніше, оскільки значна частина обчислень відбувається у стиснутому латентному просторі [12].

Послідовність основних етапів розвитку підходів до генерації зображень показано на рисунку 1.2.

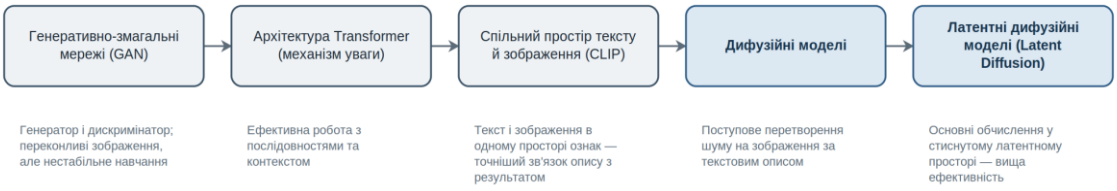

Рисунок 1.2 – Еволюція підходів до генерації зображень

Прямий процес у дифузійній моделі описує поступове додавання гаусового шуму до зображення. Один крок такого процесу записується так:

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right), \tag{1.2}$$

де $q(x_t|x_{t-1})$ – умовний розподіл переходу між двома сусідніми станами;

x_t – зображення після додавання шуму на кроці t;

x_{t-1} – зображення на попередньому кроці;

β_t – коефіцієнт шуму на кроці t, причому $0 < \beta_t < 1$;

I – одинична матриця;

$\mathcal{N}$ – нормальний розподіл.

На практиці користувач бачить не математичний процес, а інтерфейс із полем введення, вибором моделі або сервісу та результатом у вигляді зображення. Саме тому для вебзастосунку важливо не тільки підключити генеративний сервіс, а й правильно організувати вхідні дані. Текстовий опис має бути достатньо конкретним, а параметри на кшталт співвідношення сторін повинні передаватися так, щоб користувач отримав результат у потрібному форматі.

Прикладами систем і моделей, пов’язаних із генерацією зображень, є DALL·E, GLIDE, Imagen, Stable Diffusion, FLUX, сервіси Stability AI та окремі моделі, доступні через Hugging Face [13–15]. Для вебзастосунку важливим є спільний принцип їх використання: текстовий опис стає умовою генерації, а програмна система має перетворити цей опис на коректний запит до обраного сервісу.

Особливістю генерації зображень є те, що результат часто має творчий характер. Навіть за однакової інструкції модель може сформувати різні варіанти, а якість залежить від деталізації запиту, моделі, параметрів і обмежень сервісу. Через це у прикладній системі доцільно передбачити можливість уточнення запиту. Мовна модель може перетворити коротке речення на повніший опис: додати стиль, уточнити освітлення, композицію, фон, матеріали або небажані елементи.

У задачах генерації зображень важливо розділяти художній опис і керувальні параметри. Художній опис відповідає за зміст майбутнього зображення: що має бути в кадрі, у якому середовищі розташований об’єкт, який настрій або стиль потрібно передати. Керувальні параметри визначають формат результату, сервіс, модель і додаткові обмеження. Якщо ці рівні змішати, користувачеві складніше зрозуміти, чому результат відрізняється від

очікуваного: через нечіткий опис, невдалу модель, неправильний формат кадру або технічне обмеження сервісу. Тому інтерфейс генерації має відокремлювати поле текстового опису від полів вибору сервісу й параметрів.

Для прикладної програмної розробки важливо розрізняти створення власної моделі й створення власного застосунку. Навчання сучасної моделі генерації зображень потребує значних обчислювальних ресурсів, наборів даних і часу. Натомість розроблення вебзастосунку дає змогу реалізувати інший важливий аспект – інтеграцію мовної моделі, генеративного сервісу, інтерфейсу користувача, автентифікації та сценаріїв перевірки. У такому разі науково-практична цінність полягає не в навчанні нової нейронної мережі [16, 17], а в побудові керованого програмного середовища для роботи з генеративними медіа.

У генерації зображень суттєву роль відіграє співвідношення сторін. Квадратне зображення зручне для аватарів або окремих ілюстрацій, горизонтальне – для заставок і презентацій, вертикальне – для мобільного контенту. Якщо застосунок не враховує цей параметр, користувач може отримувати технічно правильне, але непридатне за форматом зображення. Тому вибір розміру або співвідношення сторін є не другорядною деталлю, а частиною користувацького сценарію.

Ще одним важливим питанням є обробка помилок зовнішніх сервісів. Генеративний сервіс може бути недоступним, вимагати ключ доступу, мати квоти, обмеження моделі або змінювати формат відповіді. Для користувача ці технічні обставини не повинні виглядати як незрозумілий збій. Вебзастосунок має пояснювати помилку, дозволяти вибрати інший сервіс або використати тестовий режим. Такий підхід забезпечує відтворюваність основних сценаріїв навіть тоді, коли зовнішні сервіси тимчасово недоступні.

Отже, генерація зображень за текстовим описом у межах цієї роботи розглядається як поєднання трьох складових: текстового наміру користувача, мовного уточнення цього наміру та виклику спеціалізованого сервісу створення

зображень. Саме на цьому поєднанні ґрунтується подальше проєктування вебзастосунку.

Окремим напрямом, пов’язаним із генерацією зображень, є природномовне редагування готового візуального матеріалу.

Редагування зображень відрізняється від генерації тим, що система працює не з порожнім візуальним простором, а з уже наявним файлом. Користувач очікує, що частина зображення залишиться незмінною, а зміниться лише фон, об’єкт, стиль, колір, освітлення або масштаб. Через це задача редагування складніша для керування: модель має врахувати як текстову інструкцію, так і структуру початкового зображення.

Принципову відмінність між генерацією та редагуванням зображень показано на рисунку 1.3.

Рисунок 1.3 – Відмінність між генерацією та редагуванням зображень

У класичних редакторах для цього застосовуються маски, виділення областей, шари, ручне ретушування, корекція кольорів і геометричні перетворення. У природномовному редагуванні користувач описує бажану зміну словами. Наприклад, інструкція може вимагати зробити фон студійним, прибрати небажаний об’єкт, замінити елемент, змінити освітлення або збільшити

зображення. Система повинна перетворити таку інструкцію на зрозумілий для моделі або сервісу запит.

Відомими напрямами у цій сфері є редагування через керування увагою в запиті, інструкційне редагування зображень, використання масок, умовне керування за допомогою структурних підказок і перетворення зображення на зображення [14, 18–21]. Наприклад, підходи на кшталт Prompt-to-Prompt показують, що зміна окремих частин текстового опису може впливати на відповідні області зображення [18]. InstructPix2Pix демонструє можливість редагування на основі природномовної інструкції [19]. ControlNet розширює керування генерацією за рахунок додаткових умов, наприклад контурів або карти глибини [21].

Для прикладного вебзастосунку важливо не намагатися підтримати всі відомі методи одразу, а виділити основні типи дій, зрозумілі користувачу. До таких дій належать заміна або уточнення об’єкта, видалення фону, масштабування, загальне стилістичне перетворення і створення плану редагування. Частина таких операцій може виконуватися зовнішньою генеративною моделлю, частина – локально звичайними алгоритмами обробки зображень. Наприклад, масштабування або видалення фону не завжди потребує великої мовної моделі, але природномовний інтерфейс може допомогти користувачу обрати потрібний режим.

Мовна модель у редагуванні корисна не тільки для покращення запиту. Вона може виконувати роль консультанта: проаналізувати завантажене зображення, описати його зміст, запропонувати план змін [22, 23], сформувати інструкцію для генеративного сервісу й попередити про можливі небажані зміни. Для користувача це важливо, бо редагування має зберігати ідентичність основного об’єкта. Якщо потрібно змінити фон, модель не повинна випадково змінити обличчя, форму товару або інший ключовий елемент.

Проблема збереження змісту особливо помітна у задачах, де редагується реальне фото. Якщо користувач завантажує власне зображення, результат має бути впізнаваним. Надмірно вільна генерація може створити красивий, але вже

інший об’єкт. Тому в системі редагування доцільно розділяти опис того, що треба змінити, і опис того, що потрібно зберегти. Таке розділення можна подати користувачу простими засобами інтерфейсу, а всередині системи – використати під час формування уточненого запиту.

Редагування також відрізняється від генерації відповідальністю за початковий матеріал. Якщо нове зображення не відповідає запиту, його можна створити повторно з іншим описом. Якщо ж невдале редагування змінює важливу частину завантаженого файлу, користувач втрачає довіру до інструменту. Тому доцільно підтримувати проміжний етап планування редагування: система може спочатку сформувати короткий план, вказати, які елементи потрібно змінити, які залишити, а вже потім виконувати генеративне перетворення. Такий підхід робить редагування прозорішим і краще відповідає задачам, де потрібно зберегти об’єкт, а не створити повністю нову сцену.

Редагування зображень також потребує оцінювання результату. Користувач може не відразу побачити всі помилки: деформацію дрібних деталей, зміну освітлення, появу артефактів [24–26], втрату реалістичності або невідповідність початковій інструкції. Візуально-мовна модель може допомогти з попередньою оцінкою, порівнявши запит і результат. Така оцінка не замінює людського рішення, але дає корисний додатковий опис.

У межах розроблюваного застосунку природномовне редагування потрібно розглядати як окремий сценарій, а не як різновид звичайної генерації. Генерація починається з опису майбутнього зображення, тоді як редагування починається з наявного файлу. Отже, інтерфейс повинен приймати зображення, інструкцію, тип редагування і, за потреби, додаткові параметри. Така структура робить роботу користувача зрозумілішою і зменшує ризик випадкових дій.

Важливою перевагою природномовного редагування є доступність для користувачів без професійних навичок графічного дизайну. Людина може сформулювати бажаний результат звичайними словами, а система допомагає перетворити цей намір на технічно придатну дію. Водночас якісний результат залежить від того, наскільки правильно вебзастосунок організовує взаємодію

між користувачем, мовною моделлю, вхідним зображенням і сервісом редагування.

1.3 Генерація відео як складніший напрям

Генерація відео є складнішим напрямом, ніж генерація окремого зображення. Відео складається з послідовності кадрів, тому модель повинна враховувати не лише вигляд об’єктів, а й часову узгодженість, рух камери, зміни фону, стабільність персонажів або предметів, тривалість сцени та плавність переходів. Якщо ці умови порушуються, користувач бачить нестабільний результат: об’єкти змінюють форму, рух виглядає неприродним, а кадри сприймаються як набір окремих зображень.

Формально відео можна подати як впорядковану послідовність кадрів:

$$V = \{F_1, F_2, \dots, F_T\}, \quad F_t \in \mathbb{R}^{H \times W \times C}, \tag{1.3}$$

де V – відео як упорядкована послідовність кадрів;

F_t – кадр із номером t;

T – кількість кадрів у відео;

H – висота кадру;

W – ширина кадру;

C – кількість каналів кольору, причому $H, W, C > 0$.

Таке подання підкреслює, що відеогенерація має працювати не тільки з якістю окремого кадру, а й з узгодженістю всієї послідовності.

У сучасних системах можна виділити два основні сценарії: перетворення текстового опису на відео та перетворення зображення на відео. У першому випадку користувач задає лише словесний опис сцени, а модель формує як перший кадр, так і подальший рух. У другому випадку користувач подає початкове зображення, яке стає опорною точкою для ролика. Для прикладного

застосунку обидва сценарії важливі, оскільки не кожна ідея починається з готового зображення, але наявний стартовий кадр часто дає більше контролю над композицією.

Обидва сценарії формування відео та їх перехід до асинхронної відеозадачі показано на рисунку 1.4.

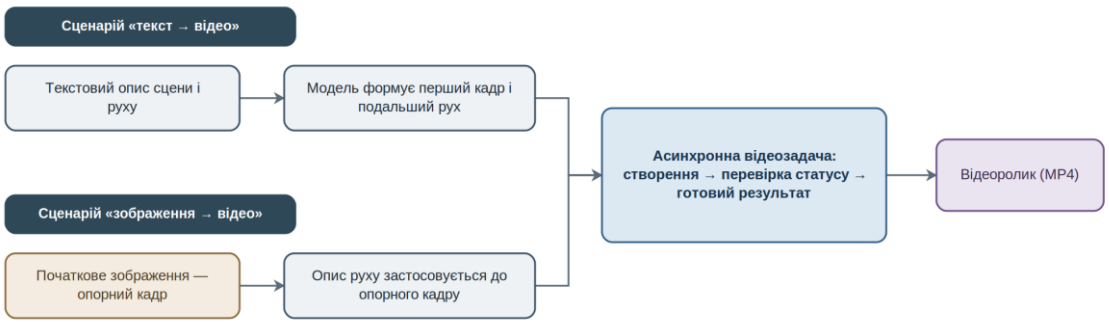

Рисунок 1.4 – Два сценарії генерації відео: за текстовим описом і на основі зображення

Відеогенеративні моделі розвиваються на основі тих самих загальних тенденцій, що й генерація зображень: дифузійні підходи, латентні представлення, текстове керування і спеціалізовані модулі для часової узгодженості [27–33]. Прикладами відомих напрямів є Video Diffusion Models, Make-A-Video, Imagen Video, Veo, AnimateDiff, Stable Video Diffusion та інші сервіси, пов’язані з перетворенням тексту або зображення на відео. Для програмного застосунку ключовими є не назви окремих архітектур, а вимоги, які вони формують до вхідних даних, статусів задачі, тривалості, формату відео та показу результату.

На відміну від зображення, відео зазвичай генерується довше. Багато сервісів не повертають готовий файл одразу, а створюють задачу, яку потрібно перевіряти до завершення. Тому у вебзастосунку необхідно підтримувати асинхронний сценарій: створення задачі, отримання її ідентифікатора, відображення статусу, повторну перевірку й показ готового відео. Якщо цей

процес не організований, користувач не розуміє, чи система працює, чи запит було втрачено.

Текстовий опис руху має іншу природу, ніж опис статичного зображення. Для зображення зазвичай описують об’єкт, фон і стиль. Для відео потрібно додати дію, темп, напрям руху, поведінку камери, бажану тривалість і характер зміни сцени. Наприклад, фраза «кінематографічний кадр міста» може працювати для зображення, але для відео варто уточнити, чи камера наближається, рухається вбік, піднімається вгору, чи має бути плавний паралакс, чи змінюється освітлення.

Для відео суттєвою є і тривалість. П’ятисекундний ролик зазвичай має містити одну просту дію, тоді як довший фрагмент може включати розвиток сцени або зміну ракурсу. Якщо інтерфейс дає лише мінімальний вибір тривалості, користувач змушений підлаштовувати задум під обмеження форми, а не навпаки. Тому в прикладному застосунку бажано передбачити ширший набір тривалостей і водночас не приховувати, що конкретний зовнішній сервіс може підтримувати тільки частину з них. Система має передавати обраний параметр тоді, коли це можливо, і повертати зрозуміле повідомлення, якщо модель має власне обмеження.

За сталої частоти кадрів кількість кадрів у відео визначається добутком тривалості на частоту кадрів:

$$N = T \cdot f, \tag{1.4}$$

де N – кількість кадрів у відео;

T – тривалість відео в секундах, причому $T > 0$;

f – частота кадрів за секунду, причому $f > 0$.

Отже, збільшення тривалості або частоти кадрів безпосередньо збільшує обсяг даних, який має бути згенерований або оброблений.

Саме тому мовна модель у відеосценарії корисна як засіб доповнення запиту. Вона може перетворити коротку ідею на опис руху, додати вказівки щодо

камери, уникнути суперечностей і зробити інструкцію придатнішою для відеомоделі. У цьому випадку кнопка покращення запиту фактично не «покращує рух» сама по собі, а доповнює текстовий опис руху. Така точність формулювання важлива для інтерфейсу, для кращого розуміння користувача роботи системи.

Окремим проміжним сценарієм є розкадрування. Воно не створює відео безпосередньо, але допомагає підготувати ідею: поділити її на сцени, описати візуальну частину, рух камери, тривалість, текст озвучення або підказки для майбутньої генерації. Для коротких роликів це особливо корисно, бо користувач може спочатку побачити структуру відео в текстовому вигляді, а вже потім запускати генерацію.

Під час відеогенерації важливими є параметри тривалості та співвідношення сторін. Короткі ролики зручні для швидкої перевірки і менше залежать від квот сервісу, довші – краще підходять для повноцінного фрагмента, але потребують більше часу. Горизонтальний формат використовується для презентацій і широких екранів, вертикальний – для мобільного перегляду, квадратний – для універсального контенту. Тому застосунок має давати змогу вибирати ці параметри там, де це підтримує обраний сервіс.

Разом із тим різні постачальники сервісів мають різні обмеження. Один сервіс може підтримувати лише кілька значень тривалості, інший – ширший вибір. Один працює тільки із зображенням як початковим кадром, інший підтримує перетворення текстового опису на відео без фото. Універсальний інтерфейс повинен враховувати ці відмінності, щоб користувач не запускав сценарій, який обраний сервіс не може виконати.

Таким чином, генерація відео вимагає від вебзастосунку більшої організації, ніж генерація зображень. Потрібно підтримати вибір джерела, параметри тривалості й формату, асинхронний статус, показ відео та зрозумілу поведінку в тестовому режимі. Ці вимоги безпосередньо впливають на архітектуру MediaGenerator.

1.4 Функціональні вимоги до прикладного вебзастосунку

Під час розроблення прикладної системи для генерації та редагування медіа можна обрати кілька загальних підходів. Локальний запуск відкритих моделей дає високий контроль над обчисленнями й параметрами, але потребує потужного обладнання, налаштування середовища, значного обсягу пам’яті та постійного оновлення залежностей. Для вебзастосунку, орієнтованого на користувацький сценарій, такий шлях надмірно зміщує увагу на інфраструктуру.

Другий підхід полягає у створенні власного вебзастосунку, який використовує зовнішні сервіси як виконавчі модулі. У цьому випадку застосунок відповідає за сценарій користувача, мінімальні дані, авторизацію, вибір сервісу, підготовку запиту, показ результату, обробку помилок і тестування. Зовнішні сервіси виконують ресурсомістку генерацію, але не визначають логіку всієї системи.

Для MediaGenerator важливим є те, що робота із зображеннями й відео має бути подана як єдиний процес. Користувач може почати з текстової ідеї, покращити її, створити зображення, відредагувати його, оцінити результат, підготувати розкадрування або перейти до відео. Якщо кожна дія існує окремо, система перетворюється на набір не пов’язаних кнопок. Якщо ж між діями є спільна логіка, користувач сприймає застосунок як творчу студію.

Важливою частиною такого застосунку є розмежування творчої та службової інформації. До творчої належать текстовий задум, інструкція редагування, опис сцени, стиль, рух камери й очікуваний результат. До службової належать назва постачальника сервісу, модель, статус задачі, ідентифікатор задачі, повідомлення про помилку й дані користувача. У зручному інтерфейсі ці типи інформації не змішуються.

Для роботи із зовнішніми сервісами потрібна гнучкість. Один сервіс може бути придатним для генерації зображень, інший – для редагування, третій – для відео. Крім того, користувач може мати ключ доступу тільки до частини сервісів. Тому застосунок має дозволяти вибір сервісу там, де це передбачено сценарієм,

і мати контрольований тестовий режим. Тестовий режим не є заміною реальної генерації, але він підтверджує працездатність власної логіки застосунку.

Для користувача така система має поєднувати простоту і прозорість. Простота означає, що користувач не повинен вручну складати складний технічний запит. Прозорість означає, що він розуміє, який сервіс обрано, який тип джерела використовується для відео, яка тривалість задана, чи триває обробка і де з’явиться результат. Саме ця рівновага між творчим описом і технічною керованістю визначає якість вебзастосунку.

Систематизовані функціональні напрями вебзастосунку MediaGenerator показано на рисунку 1.5.

Рисунок 1.5 – Функціональні напрями вебзастосунку MediaGenerator

На основі проведеного аналізу можна визначити основні функціональні напрями MediaGenerator: генерація зображень за текстовим описом, редагування завантажених зображень природною мовою, покращення запитів за допомогою мовної моделі, аналіз і оцінювання зображень, побудова розкадрування, генерація відео з тексту або зображення, відстеження статусу відеозадачі, автентифікація користувачів, адміністративний перегляд і локальний тестовий режим.

1.5 Постановка задачі

Застосування мовних моделей для редагування зображень і генерації відео є актуальним завданням у контексті розвитку мультимодальних вебсистем. Сучасний користувач очікує не лише окремої генерації файлу, а цілісного робочого процесу, у якому текстовий опис можна уточнити, використати для створення зображення, застосувати до редагування, перетворити на розкадрування або використати для відео. Тому ставиться завдання створити вебзастосунок MediaGenerator, який забезпечує роботу з генеративними медіа через природномовні інструкції та підтримує інтеграцію з кількома зовнішніми сервісами.

Об’єктом роботи є вебзастосунок для створення, редагування та підготовки відеомедіа з використанням мовних моделей і генеративних сервісів.

Метою роботи є розроблення MediaGenerator для роботи із зображеннями й відео через природномовні інструкції користувача.

Завданням роботи є розроблення вебзастосунку MediaGenerator, здатного підтримувати покращення текстових запитів за допомогою мовної моделі, генерацію та редагування зображень через обрані сервіси, аналіз візуального результату, підготовку розкадрування, створення відеозадач і локальну перевірку роботи без зовнішніх кредитів.

Для досягнення мети необхідно вирішити такі завдання:

– зробити аналіз сучасних підходів до застосування мовних моделей у генеративних медіасистемах;

– розглянути особливості генерації зображень за текстовим описом і природномовного редагування зображень;

– визначити вимоги до сценаріїв перетворення текстового опису на відео та зображення на відео;

– обґрунтувати архітектуру вебзастосунку, який поєднує клієнтську частину, серверний прикладний інтерфейс, автентифікацію, базу даних і шари взаємодії із сервісами генерації;

– реалізувати генерацію зображень із вибором сервісу та співвідношення сторін;

– реалізувати редагування завантажених зображень, зокрема режим зміни масштабу і тестовий режим;

– реалізувати сценарії покращення запиту, аналізу зображення, формування плану редагування і побудови розкадрування;

– реалізувати генерацію відео з тексту або зображення, вибір тривалості, співвідношення сторін, постачальника сервісу і перевірку статусу задачі;

– забезпечити локальний тестовий режим для зображень і відео, який дає змогу перевірити роботу застосунку без зовнішніх токенів;

– перевірити працездатність основних сценаріїв застосунку та описати отримані результати.

2 ПРОЄКТУВАННЯ ТА ТЕХНОЛОГІЧНЕ ОБҐРУНТУВАННЯ СИСТЕМИ MEDIAGENERATOR

2.1 Загальна архітектура системи

MediaGenerator спроєктовано як вебзастосунок із поділом на клієнтську, серверну, сервісну та інтеграційну частини. Клієнтська частина в frontend/src відповідає за робочу студію, авторизаційні екрани, вибір режиму, введення текстового запиту, завантаження зображення, вибір API-сервісу, показ результатів, локальну історію і адміністративну панель. Серверна частина в backend/app відповідає за маршрути FastAPI, JWT-автентифікацію, роботу з SQLite [34], журналювання типів операцій, взаємодію з мовною моделлю, генерацію та редагування зображень, а також запуск і перевірку відеозадач.

У системі немає окремого навчання власної генеративної моделі. Такий вибір відповідає прикладному характеру роботи: основна складність полягає в тому, щоб організувати керований робочий процес між користувачем, мовною моделлю, сервісами зображень і відеосервісами. Обчислювальне навантаження передається зовнішнім API або контрольованому локальному режиму Test, а власний код відповідає за підготовку запиту, перевірку параметрів, нормалізацію відповіді, показ результату та обробку помилок.

На рисунку 2.1 подано загальну архітектуру MediaGenerator. Користувач працює з React-інтерфейсом, клієнтська частина надсилає запити до FastAPI, сервер перевіряє JWT [35], обирає потрібний сервісний шар і повертає уніфікований результат. База даних не зберігає медіафайли, але фіксує користувачів і типи виконаних операцій.

Архітектура MediaGenerator побудована за принципом явних меж відповідальності. Клієнтська частина організовує дії користувача і не містить логіки формування запиту до Hugging Face або Runway.

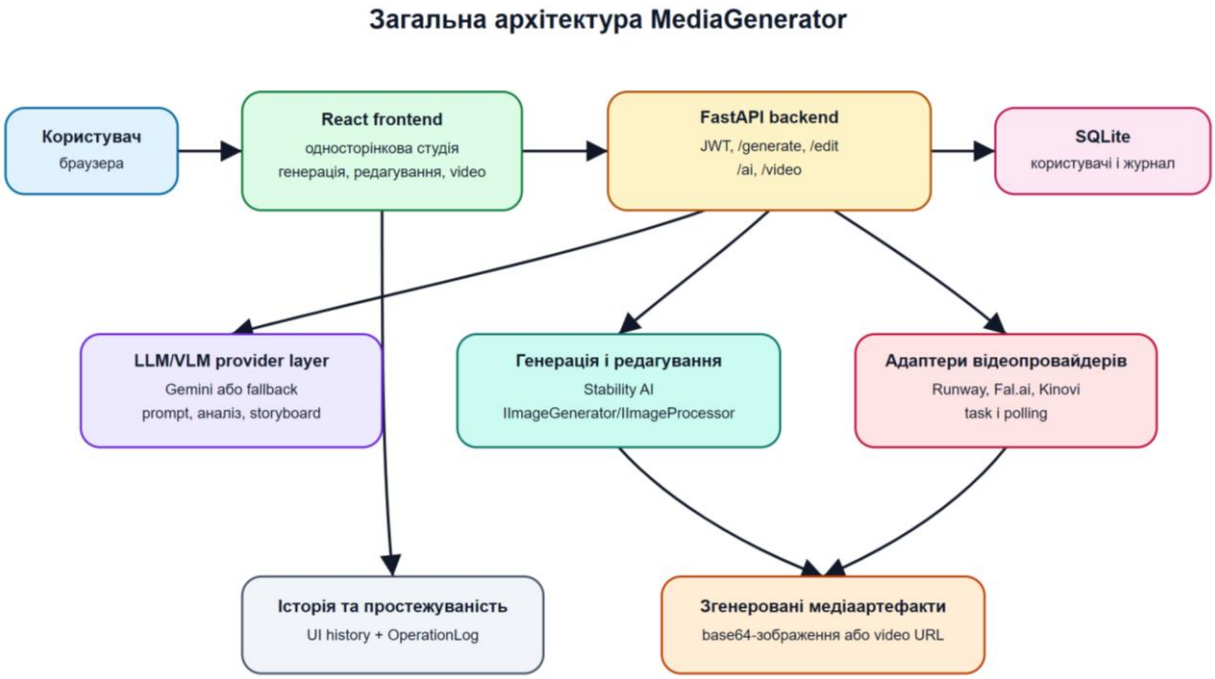

Рисунок 2.1 – Загальна архітектура MediaGenerator

Серверна частина контролює доступ, перевіряє формат даних і не відповідає за візуальне розташування елементів у студії. Шар постачальників сервісів взаємодіє із зовнішніми моделями й не працює з ролями користувачів. Завдяки такому поділу кожна частина системи має власну роль: інтерфейс організовує робочий процес, API контролює дані, сервісний шар виконує інтеграцію, а база даних фіксує мінімальну інформацію для простежуваності.

Такий поділ відповідає проблемам, визначеним у першому розділі. Медіасистема з керуванням через текстові запити має кілька джерел помилок: нечіткий запит, обмеження моделі, недоступність сервісу, невдалий результат (зокрема, нерозпізнавання об'єктів в умовах неповної інформації [22]) або технічний збій. У проєкті ці ризики перетворюються на архітектурні рішення: реєстр сервісів, режим Test, очищені повідомлення про помилки, локальна історія, розділення синхронних і асинхронних сценаріїв. Отже, архітектура не є випадковою структурою файлів, а відповіддю на вимоги, отримані з аналізу предметної області.

У таблиці 2.1 наведено основні архітектурні частини системи. Вона узагальнює склад клієнтської частини, серверного прикладного інтерфейсу,

сервісного шару, бази даних і зовнішніх інтеграцій, тому допомагає перейти від загальної схеми до пояснення ролі кожного компонента.

Таблиця 2.1 – Основні частини MediaGenerator

Частина системи	**Розташування**	**Основна відповідальність**
Клієнтська частина	frontend/src	режими студії, форми, локальний стан, результат, історія
Серверне API	backend/app/routers	маршрути автентифікації, генерації, редагування, мовної моделі, відео й перевірки стану
Сервісний шар	backend/app/services	мовна модель, сервіси зображень, відеосервіси, резервна логіка
Дані	backend/app/database.py, models.py	користувачі, ролі, журнал типів операцій
Конфігурація	.env, .env.example, config.py	ключі постачальників сервісів, моделі, тестові режими

2.2 Проєктування клієнтської частини

Клієнтську частину реалізовано як односторінкову React-студію [36]. Центральним координатором є App.jsx, де зберігаються активний режим, текстовий запит, завантажений файл, результат генерації, стан відеозадачі, повідомлення про помилки, поточний сервіс зображень, вибрана відеомодель,

локальна історія та дані користувача. Такий варіант обрано тому, що творчий процес не є лінійним. Користувач може спочатку згенерувати зображення, потім перейти до редагування, повернутися до текстового запиту, сформувати розкадрування або перевірити відеозадачу.

Інтерфейс складається з кількох режимів: генерація, редагування, розкадрування, відео та історія. У режимах генерації й редагування передбачено вибір API-сервісу. Для зображень користувач може обрати Stability, кілька маршрутів Hugging Face або локальний Test. У режимі відео користувач обирає модель із переліку Runway, Fal.ai, Kinovi, Hugging Face або Test. Такий інтерфейс не приховує різні джерела генерації, але подає їх як один зрозумілий список сервісів.

Окреме рішення стосується відображення тестового відео. Зовнішні постачальники відеосервісів повертають URL до готового ролика, тоді як Test повертає службовий test://-URL. Компонент ResultDisplay.jsx перетворює його на маршрут /api/video/artifacts/test/{task_id}.mp4 і показує результат у звичайному HTML-плеєрі <video>. Завдяки цьому тестовий сценарій поводиться для користувача як повноцінне відео, а не як JSON-відповідь або статичне зображення.

Стан клієнтської частини організовано окремо для кожного режиму. Текстовий запит, завантажений файл, результат, помилка, допоміжні панелі мовної моделі та відеозадача зберігаються в modeStates, тому результат генерації не відображається в режимі редагування, а відеорезультат не переноситься на інші вкладки. Після повернення до того самого режиму користувач бачить попередній контекст цієї вкладки. Водночас локальна історія зникає після перезавантаження сторінки, а медіарезультати не переносяться в серверне сховище. Основний акцент зроблено на робочому процесі та інтеграції сервісів, а не на побудові повноцінної медіабібліотеки.

Вибір сервісу й моделі у клієнтській частині реалізовано через службовий ключ, який поєднує назву постачальника сервісу і модель. Це дає змогу зберігати в одному полі вибору як локальні варіанти Test, так і повні ідентифікатори

моделей Hugging Face. Перед викликом API цей ключ розкладається на provider і model. Клієнтська частина не повинна знати внутрішню логіку кожного сервісу, вона лише передає вибір користувача на сервер.

Компонентна структура клієнтської частини залишається простою. LoginPage відповідає за вхід і реєстрацію, ModeSelector – за перемикання режимів, FileUpload – за завантаження зображення, EditModeSelector – за локальні типи редагування, ResultDisplay – за показ зображення або відео, AdminPanel – за перегляд користувачів адміністратором. Текст інтерфейсу винесено в i18n.js, що дає змогу підтримувати українську й англійську мови без серверного сервісу перекладу.

Для користувача важливо, щоб режим роботи був зрозумілим без читання технічної документації. Тому в інтерфейсі генерація, редагування, розкадрування і відео подані як окремі вкладки робочої студії. Кожен режим має власний набір керувань: текстовий запит для генерації, файл і режим редагування, параметри розкадрування для планування відео, службові поля вибору сервісу, моделі, тривалості й співвідношення сторін та перемикач відео за текстом або відео на основі зображення. Це зменшує змішування понять і відповідає логіці розділу 1: різні класи задач мають різні вхідні дані та різні ризики.

Проєктне рішення не використовувати складний глобальний механізм керування станом є свідомим. Для наявного обсягу застосунку стан React у App.jsx є достатнім, бо кількість екранів обмежена, а основні дії зосереджені навколо однієї студії. Додавання Redux або іншого станового фреймворку збільшило б технічну складність без пропорційної користі. Водночас код організовано так, щоб у майбутньому винести історію, відеозадачі або налаштування сервісів в окремий контекст.

Окрему роль у клієнтській частині відіграє перевірка вхідних даних до надсилання запиту. Для генерації зображень достатньо текстового опису, співвідношення сторін і вибору сервісу. Для редагування потрібний файл, інструкція і сумісний режим редагування. Для відео набір вхідних даних

залежить від типу сценарію: під час створення відео за текстом файл не потрібний, а під час створення відео на основі зображення початковий кадр є обов’язковим. Такий поділ зменшує кількість помилкових запитів і робить поведінку інтерфейсу передбачуваною.

Керування співвідношенням сторін і тривалістю також розміщено на рівні робочої студії, оскільки ці параметри належать до задуму користувача. Для зображень співвідношення сторін впливає на композицію майбутнього результату. Для відео воно визначає формат кадру, а тривалість впливає на очікуваний обсяг руху. Якщо ці параметри приховати в серверній частині, користувач не зможе заздалегідь визначити формат результату. Тому інтерфейс передає їх явно, а серверна частина перевіряє сумісність із обраним сервісом.

Панелі допоміжних відповідей мовної моделі розташовано в межах відповідного режиму. Покращений запит для генерації, план редагування, аналіз зображення або стан відеозадачі не повинні залишатися видимими після переходу до іншої вкладки. Таке рішення важливе не лише з погляду зручності, а й з погляду змістовної точності: відповідь мовної моделі стосується конкретного сценарію і може бути помилково сприйнята як результат іншої дії, якщо її не ізолювати.

Односторінкова студія обрана тому, що робота з генеративними медіа часто передбачає повернення до попереднього кроку. Користувач може сформувати запит, отримати зображення, перейти до редагування, а потім знову змінити початковий опис. Повна сторінкова навігація між окремими екранами зробила б цей процес повільнішим. Натомість вкладки дають змогу зберігати відчуття єдиного робочого середовища.

2.3 Проєктування серверної частини

Серверну частину побудовано на FastAPI [37, 38]. У main.py створюється застосунок, підключається CORS, ініціалізується база даних, реєструються компоненти й модулі маршрутів auth, generate, ai та video. Такий поділ дає змогу відокремити автентифікацію, базові операції із зображеннями, допоміжні сценарії мовної моделі та відеозадачі.

Маршрути /api/generate і /api/edit приймають не лише текстовий запит або файл, а й параметри provider та model. Це дає змогу одному маршруту працювати з різними сервісами. Якщо provider дорівнює test, запит виконує локальний режим Test. Якщо provider належить до Hugging Face, запит проходить через HuggingFaceImageProvider. Для Stability використовується відповідний сервісний механізм генерації на основі латентних дифузійних моделей [39] . У відповіді сервер повертає зображення у base64 і службові метадані сервісу та моделі, які клієнтська частина може зберегти в історії.

Відеошар реалізовано через уніфіковані функції створення задачі та перевірки статусу. Для зовнішніх постачальників сервісів це означає передавання запиту в їхній API та подальше періодичне опитування стану. Для Test створюється локальний ідентифікатор задачі, після короткої затримки статус переходить у SUCCEEDED, а відповідь містить test://mediagenerator/{task_id}. Далі клієнтська частина отримує відтворюваний MP4 через маршрут артефактів, тому інтерфейс і серверну частину можна перевіряти без витрачання зовнішніх токенів.

Серверна частина також відповідає за контрольовані помилки. Якщо ключ Stability або Hugging Face відсутній, маршрут не повинен завершуватися неконтрольованим технічним збоєм. Він має повернути зрозуміле повідомлення, щоб інтерфейс показав користувачу пояснення. Це важливо для стабільної роботи застосунку, бо зовнішні сервіси штучного інтелекту залежать від квот, доступності моделей і мережевих умов.

Маршрути серверної частини спроєктовано з урахуванням різної тривалості операцій. Під час генерації й редагування зображень результат повертається синхронно: користувач натискає кнопку, інтерфейс показує стан завантаження, а сервер повертає зображення або повідомлення про помилку. Відео працює інакше: перший запит створює задачу, а готовність перевіряється окремим маршрутом. Завдяки цьому інтерфейс не блокується під час довготривалого рендерингу, а користувач бачить проміжний статус.

Для сценаріїв мовної моделі також важливий передбачуваний формат відповіді. Покращення запиту, план редагування, розкадрування, аналіз зображення й оцінювання результату повертають JSON-структури з очікуваними полями. Якщо зовнішня модель повертає неповний або некоректний JSON, сервісний шар має сформувати резервну структуру, яку інтерфейс усе одно може показати. Таке рішення не робить резервну відповідь рівноцінною реальному виведенню мовної моделі, але зберігає стабільність інтерфейсу й дає змогу перевіряти його без обов’язкового ключа.

Контракт API спроєктовано так, щоб клієнтська частина отримувала однакові категорії відповіді незалежно від постачальника сервісу. Для зображень це base64-поле image і метадані provider та model. Для відео це task_id, provider, status, model і, після завершення, video_url. Для сценаріїв мовної моделі це структуровані об’єкти, які можна показати у Director Notes. Усі ці контракти не є повною абстракцією зовнішніх сервісів, але вони підтримують стабільний інтерфейс і подальше розширення.

Важливим є і напрям залежностей. Модулі маршрутів викликають сервісний шар, але сервісний шар не залежить від React-компонентів або конкретного вигляду сторінки. Компоненти клієнтської частини викликають API-функції, але не імпортують серверний код і не мають доступу до секретних ключів. Такий напрям залежностей спрощує перевірку: можна окремо перевірити серверні маршрути через TestClient, окремо зібрати клієнтську частину через Vite і окремо виконати браузерний сценарій.

Серверні маршрути мають виконувати не тільки передавання даних, а й їхню нормалізацію. Запит від клієнтської частини може містити службовий ключ вибраного сервісу, модель, файл, співвідношення сторін, тривалість або тип відеовходу. Сервер повинен перетворити ці дані на формат, який очікує конкретний постачальник сервісу, і після відповіді привести результат до спільного вигляду. Саме тому маршрути не повинні містити всю логіку зовнішніх API, а передають виконання відповідному сервісному шару.

Для відео особливо важливо розділити створення задачі й отримання результату. Якщо сервер намагатиметься тримати один HTTP-запит відкритим до завершення рендерингу, інтерфейс може зависати, а користувач не бачитиме проміжного стану. Натомість створення задачі повертає ідентифікатор, а перевірка статусу виконується окремо. Такий контракт ближчий до реальної поведінки зовнішніх відеосервісів і дає змогу однаково обробляти PENDING, PROCESSING, SUCCEEDED і FAILED.

Помилки зовнішніх сервісів не можна без змін передавати в інтерфейс. Вони можуть містити зайві службові деталі, незрозумілі користувачу коди або нестандартну структуру. Тому серверна частина очищує повідомлення й повертає коротке пояснення. Для користувача важливі три речі: що операцію не виконано, чому це могло статися і що можна спробувати далі, наприклад вибрати Test або інший сервіс.

Ще одним елементом серверного проєктування є перевірка ролей. Маршрути генерації та редагування доступні автентифікованому користувачу, але адміністративний список користувачів доступний тільки ролі admin. Такий поділ не ускладнює основні медіасценарії, але показує, що застосунок має базову модель доступу, а не один відкритий маршрут генерації.

2.4 Модель даних і журналювання

Модель даних у MediaGenerator свідомо обмежена. Сутність User містить ім’я користувача, хеш пароля, роль, ознаку активності та дату створення. Сутність OperationLog містить посилання на користувача, тип операції й час виконання. Система не зберігає повні текстові запити, зображення або відео як окремі сутності. Основним завданням моделі даних є підтримка автентифікації, ролей і базової простежуваності дій, а не побудова повноцінного медіасховища.

Ролі мають два рівні. Звичайний користувач може працювати з генерацією, редагуванням, LLM-сценаріями, відео та історією. Адміністратор додатково переглядає список користувачів. Публічна реєстрація не створює адміністратора, навіть якщо клієнт вручну надішле роль admin. Це просте, але важливе обмеження, яке не дозволяє отримати адміністративні права через форму реєстрації.

Простежуваність реалізовано на двох рівнях. Серверний рівень фіксує типи виконаних операцій у OperationLog. Клієнтський рівень зберігає локальну історію сесії з фрагментами відповідей. Серверний журнал потрібний для базового аудиту, а локальна історія – для прозорості творчого процесу в інтерфейсі. Ці механізми не дублюють один одного: один працює як мінімальна база даних, інший – як користувацький журнал дій.

Модель даних відокремлює облікові записи й журнал виконаних дій від медіафайлів. Зображення та відео показуються користувачу в межах сеансу роботи, а в базі даних фіксується тип виконаної операції, час і користувач. Такий підхід зменшує обсяг службових даних, не перевантажує базу великими файлами й залишає основний акцент на сценаріях генерації, редагування та перевірки результатів.

За потреби систему можна розширити окремими сутностями для медіафайлів, версій запитів, відеозадач і збережених результатів. Однак ці сутності мають проєктуватися разом із правилами доступу, обмеженнями розміру, очищенням файлів і захистом приватних матеріалів користувача. У

розробленому застосунку мінімальна модель даних підтримує автентифікацію, ролі й простежуваність основних дій.

Такий мінімалізм моделі даних узгоджується зі сценаріями MediaGenerator. Користувач отримує результат одразу в інтерфейсі, а система фіксує сам факт виконання операції. Для генерації зображення важливими є текстовий запит, сервіс і зображення, але зберігання кожного такого результату потребувало б окремого сховища, правил видалення і контролю доступу. Для відео питання ще складніше, оскільки файли мають більший розмір і можуть створюватися довше. Тому на рівні бази даних залишено тільки ті сутності, які потрібні для входу, ролей і базового журналу.

Роль OperationLog полягає не в тому, щоб замінити повну історію медіа, а в тому, щоб показати зв’язок між користувачем і типом виконаної дії. Наприклад, система може зафіксувати факт генерації, редагування або створення відеозадачі. Локальна історія в інтерфейсі, навпаки, показує користувачу останні результати поточної сесії. Разом ці два рівні утворюють достатню простежуваність для розробленого обсягу застосунку.

Проєктування ролей також впливає на дані. Публічна реєстрація створює звичайного користувача, а не адміністратора. Тому поле ролі в базі даних не повинно довіряти значенню, довільно переданому з клієнтської частини. Серверна логіка має самостійно визначати роль нового облікового запису і перевіряти її під час доступу до адміністративних маршрутів.

2.5 Компонентний шар мовної моделі та візуального аналізу

Шар мовної моделі [40] реалізовано в services/llm.py. Його роль не зводиться до генерації красивого тексту. Він підтримує кілька конкретних сценаріїв: покращення текстового запиту, побудову плану редагування, аналіз зображення, оцінювання результату й генерацію розкадрування. Для кожного сценарію очікується структурована відповідь, тому клієнтська частина може

показувати результат у панелі Director Notes, а не просто виводити довільний текст.

На рисунку 2.2 показано робочий процес мовної моделі в MediaGenerator. Користувач задає природномовну інструкцію або завантажує зображення. Серверна частина передає дані до шару мовної або візуально-мовної моделі [41], отримує структуровану відповідь і повертає її інтерфейсу. Далі ця відповідь може бути використана як покращений текстовий запит, уточнена інструкція для редагування, розкадрування або оцінка якості.

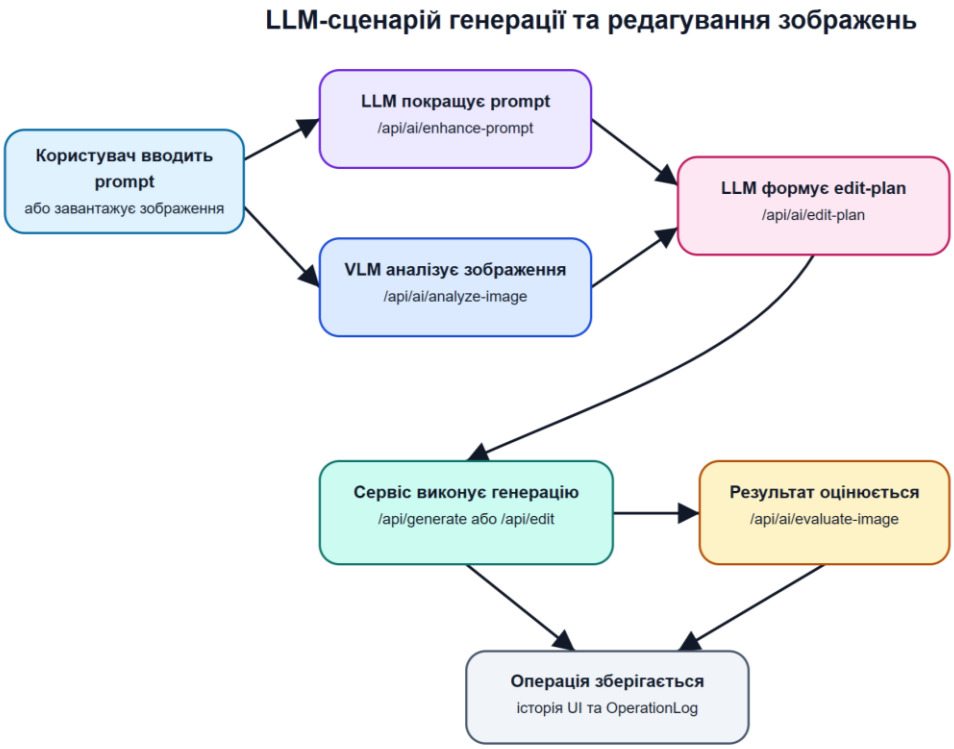

Рисунок 2.2 – Робочий процес мовної моделі в MediaGenerator

У цьому шарі важливо зберігати чесну межу між допоміжною логікою та генеративним синтезом. LLM може уточнити інструкцію, але вона не гарантує ідеальний результат у моделі зображень, що може призвести до деформації структурних ознак [1, 2]. VLM може описати завантажений файл, але може помилитися в деталях. Тому LLM-відповіді використовуються як підказки й проміжні структури, а не як абсолютна оцінка якості.

Ще одна причина винесення логіки мовної моделі в окремий сервіс полягає у змінності моделей. Застосунок може використовувати Gemini, резервну відповідь або інший постачальник мовної чи візуально-мовної моделі. Якби покращення запиту й розкадрування були розміщені безпосередньо в маршрутах або React-компонентах, заміна моделі вимагала б змін у багатьох місцях. Сервісний шар концентрує цю змінність і залишає зовнішній контракт стабільним для інтерфейсу.

Практичний сенс шару мовної моделі полягає в тому, що користувач не повинен самостійно думати мовою конкретного сервісу. Наприклад, інструкція «зробити фон студійним і не змінювати обличчя» для людини є зрозумілою, але для сервісу редагування зображень краще мати уточнений запит із цільовим об’єктом, областю збереження, змінами фону, освітленням і небажаними змінами. Шар мовної моделі виконує саме це перетворення. Він не усуває потребу в людській оцінці, але робить проміжний намір видимим.

Сценарій розкадрування також належить до шару мовної моделі. Він не створює відео безпосередньо, але готує майбутній опис руху: поділяє ідею на сцени, описує рух камери, пропонує короткий текст озвучення і запити для окремих кадрів. У контексті теми роботи це важливо, бо генерація відео рідко починається з одного випадкового речення. Якісніший робочий процес передбачає підготовку сцени, а потім запуск створення відео на основі зображення або текстового опису.

Для кожного сценарію мовної моделі потрібен окремий шаблон запиту. Покращення запиту має зберегти задум користувача, але зробити його конкретнішим для генеративної моделі. План редагування має відокремити об’єкт зміни від елементів, які потрібно зберегти. Аналіз зображення має описати вміст файлу, а оцінювання результату – зіставити його з початковим запитом [42, 43]. Якщо всі ці сценарії звести до одного універсального звернення, відповіді будуть менш передбачуваними і складнішими для відображення в інтерфейсі.

Структурований формат відповіді важливий ще й тому, що інтерфейс не повинен показувати користувачу довільний великий текст там, де очікується

короткий план або список сцен. Наприклад, розкадрування зручно подавати як послідовність кадрів, а не як суцільний абзац. План редагування доцільно розділяти на ціль зміни, об’єкти збереження й уточнений запит. Тому шар мовної моделі виконує не лише генерацію тексту, а й підготовку даних до подальшого використання в робочій студії.

Резервні відповіді також є частиною проєктного рішення. Якщо зовнішня мовна модель тимчасово недоступна, застосунок не повинен втрачати весь інтерфейс допоміжних сценаріїв. Резервна структура не замінює повноцінну відповідь моделі, але дає змогу показати очікуваний формат, перевірити панель результатів і не переривати роботу інших частин системи.

2.6 Сервісні інтеграції та контейнерний запуск

Шар сервісів зображень винесено в окремий набір модулів backend/app/services/image_providers. У ньому є базовий контракт, сервіс для Stability, сервіс для Hugging Face і локальний TestImageProvider. Це зменшує залежність маршрутів від конкретного API і робить вибір сервісу явною частиною запиту.

Hugging Face обрано як додатковий напрям інтеграції, тому що Inference Providers надають єдиний API для різних типів задач, зокрема генерації зображень, а бібліотека huggingface_hub містить InferenceClient для викликів із Python [44, 45]. У MediaGenerator цей варіант не подається як гарантовано безкоштовний у будь-яких умовах: офіційна документація Hugging Face зазначає наявність безоплатних кредитів і оплату після їх вичерпання залежно від постачальника сервісу та обчислень [46]. Саме тому поряд із Hugging Face використовується Test, який не залежить від зовнішніх кредитів.

Відеосервіси мають іншу природу, ніж генерація зображення. Зображення зазвичай повертається синхронно як base64 або файл. Відео є довготривалою задачею, тому сервер повертає task_id, а клієнтська частина періодично перевіряє

статус. Уніфікований контракт містить службові поля provider, model, task_id, status і video_url. Завдяки цьому один інтерфейс може працювати з Runway, Fal.ai, Kinovi, Hugging Face і Test.

Уніфікація постачальників сервісів не означає, що всі вони мають однакові можливості. Одні сервіси краще підходять для коротких роликів на основі зображення, інші – для генерації зображення за текстом або редагування зображення, частина потребує окремого model, частина має власні статуси задач. Тому сервер виконує роль адаптера: він не приховує назву сервісу, але приводить результат до спільного мінімального контракту. У клієнтській частині це зменшує кількість умовних гілок і дає змогу додавати нові сервіси без повної перебудови інтерфейсу.

Для зображень сервісний шар також полегшує розширення. Якщо в майбутньому буде додано сервіс, що підтримує маски, розширення меж зображення або опорне зображення, його можна реалізувати в окремому класі й підключити через реєстр. Маршрути вже приймають provider і model, тому основний контракт API не потрібно змінювати.

Порівняно з прямою інтеграцією API в маршрути, шар постачальників сервісів має три переваги. По-перше, маршрут не містить деталей конкретного HTTP-клієнта, заголовків авторизації або формату тіла запиту. По-друге, помилки можна очищувати в одному місці, не дублюючи правила в кожному кінцевому маршруті. По-третє, Test стає рівноправним учасником архітектури, а не тимчасовим «обхідним шляхом», тому контрольний сценарій перевіряє той самий контракт, що й реальний зовнішній сервіс.

Сервіси зображень і відео мають різну модель очікування результату. Для зображення користувач зазвичай очікує результат після одного запиту, тому інтерфейс показує індикатор очікування й одразу виводить зображення. Для відео результат може з’явитися через значний проміжок часу, тому користувач бачить активну задачу, її статус і потім відеоплеєр. Ця різниця впливає на архітектуру: один і той самий компонент результату має підтримувати як

статичне зображення, так і відеофайл, а серверна частина має повертати різні контракти для синхронних і асинхронних сценаріїв.

Підтримка відео за текстом і відео на основі зображення також потребує окремого проєктного рішення. Якщо сценарій починається з тексту, основним вхідним елементом є опис сцени, руху й стилю. Якщо сценарій починається із зображення, файл стає опорним кадром, а текст уточнює рух і зміну сцени. У клієнтській частині ці варіанти подано як типи входу, а серверна частина перевіряє, чи потрібний файл для обраного режиму.

Тривалість відео не є другорядним параметром. Короткий ролик потребує стислого руху, а довший – більш розгорнутої дії або плавнішого розвитку сцени. Тому в інтерфейсі передбачено ширший вибір тривалості, ніж лише два фіксовані значення. Сервер передає цей параметр постачальнику сервісу, якщо той його підтримує, або обробляє обмеження обраної моделі контрольованим повідомленням.

На рисунку 2.3 показано робочий процес відеозадачі через зовнішній сервіс. Користувач завантажує зображення, обирає сервіс і параметри, сервер створює задачу, а клієнтська частина перевіряє статус до появи готового результату.

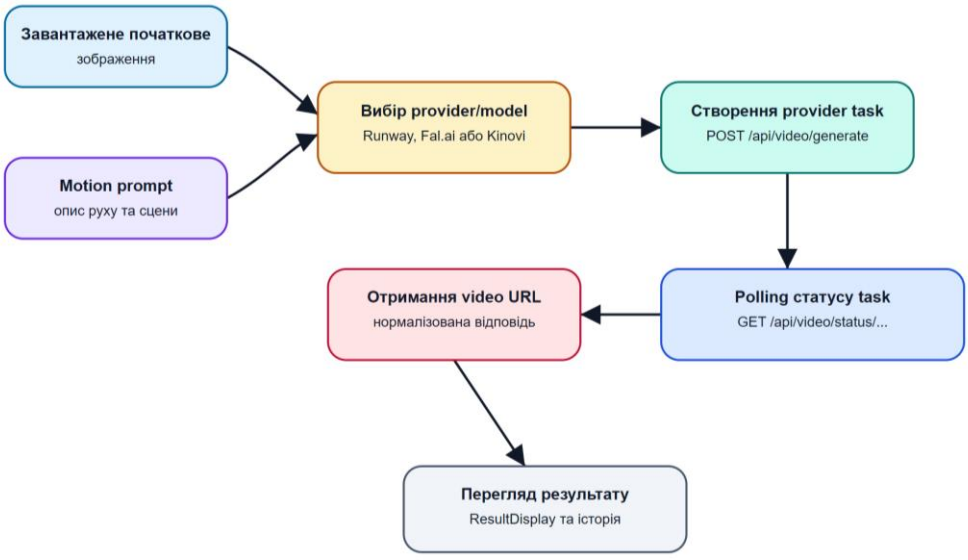

Рисунок 2.3 – Робочий процес відеозадачі через зовнішній сервіс

У таблиці 2.2 подано ролі постачальників сервісів у застосунку.

Таблиця 2.2 – Постачальники сервісів генерації та редагування в MediaGenerator

Рівень	**Постачальник сервісу або механізм**	**Роль у системі**
Мовна або візуально-мовна модель	Gemini або резервна відповідь	покращення запиту, план редагування, розкадрування, аналіз і оцінювання
Зображення	Stability	базова генерація та редагування зображень
Зображення	Hugging Face Inference Providers	альтернативний доступ до text-to-image/image-to-image моделей
Зображення	TestImageProvider	локальна Test-генерація й Test-редагування
Відео	Runway, Fal.ai, Kinovi, Hugging Face	асинхронні відеозадачі на основі зображення
Відео	TestVideoService	локальна перевірка задачі, статусу й відеорезультату

Локальний режим Test є окремим проєктним механізмом у межах сервісних інтеграцій.

Локальний режим Test призначений для контрольованої перевірки прикладної логіки MediaGenerator. Він не замінює генеративні моделі й не претендує на художню якість результатів. Його завдання інше: забезпечити стабільний шлях виконання запиту, коли потрібно перевірити авторизацію,

маршрути серверної частини, передавання файлів, оброблення відповіді та відображення результату в інтерфейсі.

З практичного боку такий режим допомагає розділити помилки на дві групи. Перша група пов’язана з власним застосунком: форма не передала дані, серверний маршрут не спрацював, результат не відобразився або статус не оновився. Друга група пов’язана із зовнішнім сервісом: відсутній ключ, завершилася квота, модель недоступна або формат відповіді не збігається з очікуваним. Test дозволяє перевірити першу групу незалежно від другої.

Для зображень TestImageProvider створює PNG із контрольованою композицією, назвою «MediaGenerator Test» і фрагментом користувацького запиту або інструкції редагування. Для редагування він приймає завантажене зображення й повертає результат у тому самому форматі, що й зовнішні сервіси. Для відео TestVideoService створює ідентифікатор задачі, переводить її стан від PENDING до SUCCEEDED і формує локальний MP4-артефакт. Завдяки цьому перевіряється не тільки створення задачі, а й фінальний етап – відтворення відео у плеєрі.

Наявність такого режиму допомагає відокремити помилки власного застосунку від проблем зовнішнього сервісу. Якщо локальний шлях Test працює, але зовнішній сервіс повертає помилку, першочергово аналізуються ключ доступу, квота, модель, формат запиту або мережеве з’єднання. Якщо помилка виникає і в режимі Test, причину потрібно шукати у клієнтській формі, серверному маршруті, обробленні файлу або логіці відображення результату.

Режим Test також корисний для перевірки сценаріїв, у яких результат має певний формат. Зображення повертається як рядок base64, відео – як MP4-файл, статус задачі – як структурована відповідь із фіксованими значеннями. Це дає змогу однаково перевіряти успішні й негативні сценарії, не змішуючи якість генерації з працездатністю програмної системи.

Ще одна перевага полягає в тому, що локальний режим зручний для перевірки інтерфейсних деталей. Саме в ньому видно, чи коректно поводяться вкладки, чи зберігається контекст режиму, чи не переноситься панель результату

в іншу частину студії, чи правильний формат має тестове відео. Для складних творчих застосунків ці ознаки не менш важливі, ніж сам факт отримання відповіді від зовнішнього API.

Безпека, конфігурація та робота з ключами доступу також належать до проєктування інтеграційного шару.

Ключі зовнішніх сервісів не повинні зберігатися у клієнтському коді або репозиторії. Для цього в MediaGenerator використовується файл середовища .env, а .env.example містить лише шаблонні назви змінних. Серверна частина читає конфігурацію через config.py, де передбачено параметри Stability, Hugging Face, відеосервісів, режиму Test і загальні службові налаштування застосунку.

JWT використовується для доступу до приватних маршрутів. Запит без токена не запускає генерацію, редагування або адміністративні дії. Окремо перевіряється роль користувача: звичайний обліковий запис працює з медіасценаріями, а адміністративний має доступ до списку користувачів. Такий поділ зменшує ризик випадкового або навмисного доступу до службових функцій.

Для зовнішніх сервісів важливо не повертати користувачу повні технічні відповіді, якщо вони можуть містити внутрішні адреси, ключі або службові деталі. Повідомлення про помилку має бути зрозумілим, але очищеним від приватної інформації. Це особливо важливо для сценаріїв, де відповідь формується не власною системою, а стороннім API.

Безпека вибору сервісу також не може спиратися лише на інтерфейс. Клієнтська частина показує передбачені варіанти моделі й сервісу, але серверна частина повторно перевіряє отримані значення. Користувацький рядок не повинен перетворюватися на довільний маршрут або виклик непередбаченого зовнішнього ресурсу.

Окремої уваги потребують завантажені файли. Навіть якщо інтерфейс приймає тільки зображення, файл користувача не можна вважати безпечним за замовчуванням. Для розробленого застосунку базовими засобами контролю є обмеження типу операції, перевірка сценарію оброблення, розділення маршрутів

і журналювання помилок. У разі подальшого розвитку доцільно додати перевірку MIME-типу на сервері, обмеження розміру файлу, очищення тимчасових артефактів і захист від надмірної кількості запитів.

Конфігурація має бути відокремлена від коду, тому один і той самий застосунок може працювати з різним набором зовнішніх сервісів. Якщо ключ певного сервісу не налаштований або квоту вичерпано, користувач не повинен втрачати доступ до всього інтерфейсу. У такому випадку залишається доступним локальний режим Test, а реальні генеративні сервіси підключаються через відповідні змінні середовища.

Це рішення має ще один практичний наслідок: один і той самий програмний артефакт можна відкривати в різних умовах без зміни коду. На одному запуску можуть бути ввімкнені всі доступні сервіси, на іншому – лише локальний Test, а на третьому – один зовнішній сервіс для зображень і один для відео. Для користувача це виглядає як однакова студія, хоча ззовні набір інтеграцій змінюється.

Крім того, сервісні ключі потрібно захищати від випадкового потрапляння до історії комітів або клієнтського пакета. Тому .env.example зберігає тільки назви змінних, а самі значення залишаються поза репозиторієм. Це не лише питання безпеки, а й питання відтворюваності: інший користувач може побачити, які саме змінні потрібні для запуску, і підставити власні значення без редагування вихідного коду.

У випадку відео та зображень безпечне поводження з ключами особливо важливе, оскільки ці запити можуть бути дорогими або обмеженими за квотою. Якщо ключ потрапить до клієнтського коду, він стає доступним у браузері. Якщо ключ потрапить до репозиторію, він може зберегтися у зовнішніх копіях. Тому серверна конфігурація, локальне зберігання значень і перевірка наявності ключів є частиною самої архітектури, а не другорядним технічним кроком.

Локальне розгортання MediaGenerator передбачає узгоджену роботу клієнтської і серверної частин.

Локальний запуск MediaGenerator передбачає серверну частину на порту 8000 і клієнтську частину через Vite або контейнерний вебсервер. Docker Compose описує два сервіси: серверний застосунок і клієнтський інтерфейс. Змінні середовища передаються через .env, тому зовнішні сервіси можна підключати або вимикати без зміни програмного коду.

Практично це означає, що запуск системи не залежить від одного сценарію розгортання. У середовищі розроблення зручним є локальний запуск через Vite і FastAPI. Для повторюваної перевірки підходить контейнерний запуск. Для перевірки інтерфейсу важливо лише те, що клієнтська частина бачить серверний API, а серверна частина коректно читає конфігурацію і може відповісти на контрольний запит.

Під час проєктування враховано стабільність клієнтської збірки. Tailwind CSS має знаходити frontend/tailwind.config.js незалежно від робочої папки запуску, тому PostCSS явно посилається на відповідний конфігураційний файл, а шляхи content нормалізовано у форматі з прямими слешами. Це рішення не змінює функціональних можливостей системи, але забезпечує коректне відображення інтерфейсу.

У таблиці 2.3 підсумовано ключові проєктні рішення та їхній вплив на систему.

Проєктне рішення має визначені обмеження. MediaGenerator не є промисловою системою керування медіаактивами, не зберігає медіафайли як бібліотеку, не виконує власне навчання моделей і не гарантує художню якість зовнішньої генерації. Його призначення полягає в іншому: об’єднати мовну модель, генерацію зображень, природномовне редагування, асинхронне створення відео, авторизацію, журналювання і контрольований тестовий режим в одному прикладному вебзастосунку.

Саме ці обмеження роблять рішення реалістичним. Застосунок не намагається замінити зовнішні сервіси, а працює як надбудова над ними.

Таблиця 2.3 – Проєктні рішення та наслідки для MediaGenerator

Рішення	**Причина вибору**	**Наслідок**
React-студія з локальним станом	творчий процес потребує швидкого перемикання режимів	менше навігації, чіткіший користувацький сценарій
Маршрути FastAPI	розділення автентифікації, зображень, мовної моделі й відео	зрозуміла структура серверної частини
Шар постачальників сервісів	різні API мають різні формати й квоти	один інтерфейс працює з кількома сервісами
Режим Test	зовнішній сервіс може бути недоступний	відтворювана перевірка без платних запитів
.env і .env.example	ключі не повинні бути в репозиторії	безпечніша конфігурація запуску

Користувач отримує єдиний інтерфейс, а система зберігає достатню гнучкість, щоб змінювати постачальників сервісів, режими або формати відповіді без повного перепроєктування студії. Такий баланс є корисним для прикладної роботи, де головним результатом стає не абстрактна теорія, а завершений і зрозумілий інструмент.

Під час проєктування було розглянуто три можливі стратегії. Перша стратегія – локально запускати відкриті моделі зображень і відео. Вона дала б більше контролю над виконанням моделей, але вимагала б потужного графічного процесора, налаштування ваг і окремої оптимізації залежностей.

Друга стратегія – обмежитися готовими вебсервісами. Вона найпростіша для користувача, але не створює власного програмного артефакту з авторизацією, API, журналюванням і тестовою інфраструктурою. Третя стратегія

– побудувати власний вебзастосунок, який працює через зовнішні API й має локальний режим Test. Саме цей варіант обрано для MediaGenerator.

Обраний підхід узгоджується з темою роботи, бо мовні моделі використовуються не ізольовано, а як частина медіасценаріїв. Шар мовної моделі допомагає формулювати й пояснювати інструкції, постачальник зображень створює або редагує візуальний результат, відеосервіс виконує довготривалу задачу, а режим Test забезпечує контрольований шлях перевірки. Така композиція краще відповідає прикладній розробці, ніж окремий виклик однієї моделі без інтерфейсу, ролей і сценаріїв контролю.

У цій архітектурі важливо розділити творчу й службову інформацію. До творчої належать запит, інструкція редагування, розкадрування, опис руху та очікуваний стиль. До службової – назва сервісу, модель, ідентифікатор задачі, статус, помилки й інформація про користувача. Якщо ці типи даних змішати, інтерфейс стає менш зрозумілим: користувач не бачить, що саме він редагує, а що є технічним станом задачі. У MediaGenerator творча інформація подається в основних формах і допоміжних панелях, а службова використовується для маршрутизації, історії та перевірки статусів.

Таке розмежування допомагає і під час тестування. Контрольне зображення або відео Test не є творчою метою, але має коректну службову поведінку: сервіс дорівнює test, статус відео переходить до SUCCEEDED, результат має очікуваний формат. Отже, другий розділ формує основу для третього: можна окремо описати реалізацію творчих сценаріїв і окремо довести, що службові контракти працюють.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ MEDIAGENERATOR

3.1 Загальна реалізація застосунку

MediaGenerator реалізовано як вебзастосунок із поділом на клієнтську частину, серверну частину, конфігурацію та допоміжні матеріали для перевірки. Клієнтський код зосереджено у frontend/src, серверний код – у backend/app, а параметри запуску описано через змінні середовища. Така структура дає змогу окремо розглядати інтерфейс користувача, серверну логіку, роботу з даними та взаємодію із зовнішніми сервісами генерації.

Серверну частину побудовано на FastAPI [37, 38], клієнтську – на React [36] і Vite, а для збереження користувачів та журналу операцій використано SQLite [34]. У контейнерному режимі клієнтська й серверна частини запускаються як окремі сервіси, а під час локальної роботи вони взаємодіють через HTTP. Конфігурація Tailwind CSS прив’язана до файлів клієнтської частини так, щоб стилі коректно збиралися незалежно від робочої папки запуску.

Конфігурація серверної частини винесена в config.py. У ній визначено ключі зовнішніх сервісів, стандартні моделі, параметри локального режиму Test і службові налаштування застосунку. Реальні значення ключів зберігаються у .env, а .env.example містить тільки назви очікуваних змінних. Завдяки цьому клієнтський інтерфейс не отримує секретів і працює лише з вибором сервісу, моделі, режиму та параметрів генерації.

Розроблена структура не прив’язує застосунок до одного зовнішнього сервісу. Якщо налаштовано ключі Stability, Hugging Face або відеосервісів, серверна частина може звертатися до них. Якщо зовнішній сервіс тимчасово недоступний або квоту вичерпано, користувач може обрати Test і пройти той самий інтерфейсний шлях без зовнішнього запиту. Це робить основні сценарії відтворюваними й дозволяє перевіряти власну логіку застосунку окремо від умов роботи сторонніх сервісів.

Важливо, що структура репозиторію відповідає логіці самого застосунку. Клієнтська частина не містить секретних ключів і не викликає зовнішні моделі напряму. Серверна частина приймає запит, перевіряє користувача, обирає постачальника сервісу й приводить відповідь до формату, який може показати інтерфейс. Такий поділ зменшує кількість залежностей між частинами системи: зміна сервісу генерації не потребує переписування сторінок інтерфейсу, а зміна компонента відображення результату не впливає на логіку автентифікації.

На рисунку 3.1 подано структуру репозиторію і запуск основних частин MediaGenerator. Схема показує, що клієнтська частина не звертається до зовнішніх API штучного інтелекту напряму. Усі запити проходять через серверну частину, де відбувається автентифікація, вибір сервісу, нормалізація відповіді й журналювання операції.

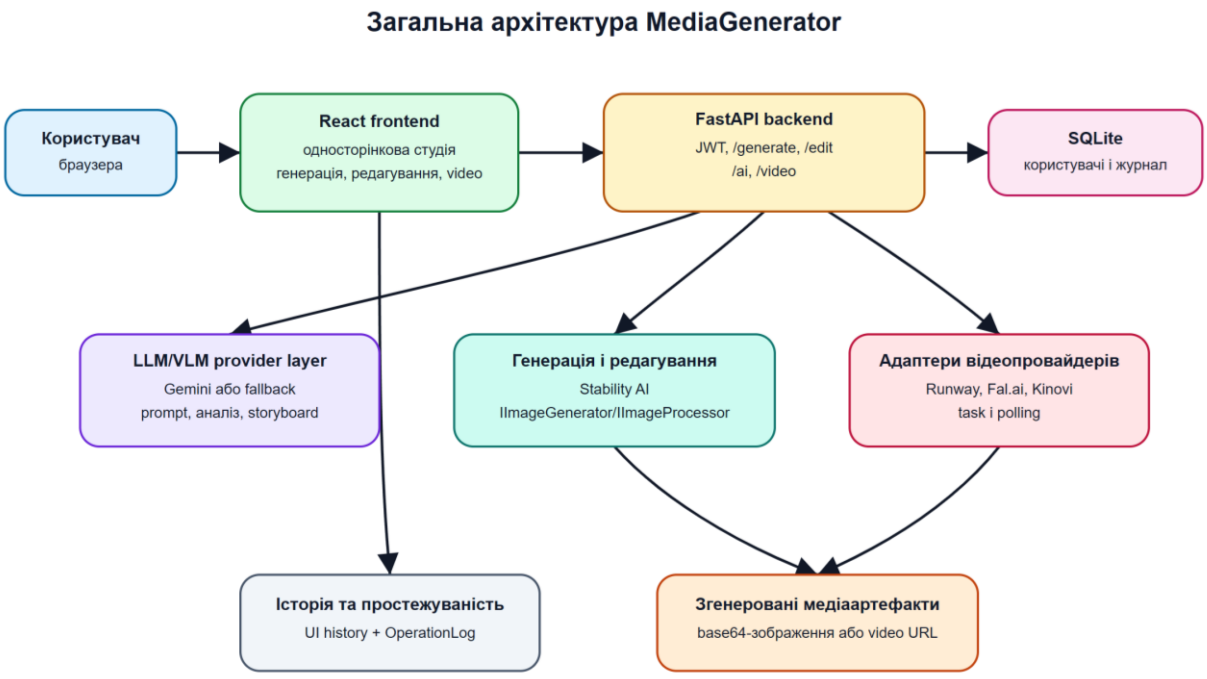

Рисунок 3.1 – Структура репозиторію та запуск MediaGenerator

У таблиці 3.1 наведено основні файли й групи файлів, які визначають практичну реалізацію застосунку.

Таблиця 3.1 – Основні файли практичної реалізації

Файл або група файлів	**Роль у реалізації**
backend/app/services/image_providers/*	шар постачальників сервісів для Stability, Hugging Face і Test
backend/app/routers/generate.py	маршрути генерації та редагування з вибором сервісу й моделі
backend/app/services/runway.py	адаптери відеосервісів і локальний TestVideoService
frontend/src/App.jsx	робоча студія, вибір сервісу, окремий стан вкладок, режим відео за текстом
frontend/src/components/ResultDisplay.jsx	відображення зображень, зовнішніх відео й локального MP4-артефакту
frontend/postcss.config.js, frontend/tailwind.config.js	стабільність збирання стилів клієнтського інтерфейсу

3.2 Реалізація автентифікації та ролей

Автентифікацію реалізовано через JWT [35]. Користувач входить у систему через форму, серверна частина перевіряє хеш пароля, повертає токен, а клієнтська частина додає його до приватних запитів. Без токена користувач бачить лише екран входу або реєстрації, а після успішної автентифікації відкривається робоча студія MediaGenerator.

Після входу клієнтська частина одразу отримує інформацію про поточного користувача. Це потрібно, щоб інтерфейс правильно визначив доступні режими й показав адміністративну панель тільки обліковому запису з відповідною роллю. Такий підхід поєднує перевірку прав і початкове завантаження студії, тому користувач не бачить елементів, до яких не має доступу.

На серверному рівні вхід реалізовано через OAuth2PasswordRequestForm. Після перевірки облікових даних сервер формує access token, а клієнтська частина викликає /api/auth/me для отримання ролі користувача. Це дає змогу не покладатися лише на дані, введені у формі, і щоразу перевіряти поточний стан облікового запису.

Реєстрація створює звичайного користувача. Адміністративна роль не може бути отримана через публічну форму, оскільки це створювало б ризик несанкціонованого доступу до службових функцій. Звичайний користувач працює з генерацією, редагуванням, розкадруванням, відео й історією, а адміністратор додатково переглядає список користувачів.

Локалізацію реалізовано у frontend/src/i18n.js. Інтерфейс підтримує українську та англійську мови, зокрема назви режимів, повідомлення про стан задачі, вибір API-сервісу, параметри генерації та підказки для роботи із зображеннями й відео. Обрана мова зберігається в localStorage, тому після оновлення сторінки користувач продовжує працювати у звичному мовному режимі.

На рисунку 3.2 і рисунку 3.3 показано сторінку входу та основний екран після автентифікації. Скріншоти підтверджують, що користувач не потрапляє до робочих режимів без входу, а після авторизації бачить студію з режимами генерації, редагування, розкадрування, відео й історії.

У реалізації автентифікація важлива не лише як захист, а й як частина логіки роботи застосунку. Прив’язка дій до користувача дозволяє вести простий журнал операцій і водночас обмежувати доступ до адміністративних функцій. Для системи генерації медіа це суттєво, оскільки результат створюється в межах керованого облікового сценарію, а не через відкритий публічний маршрут.

Журналювання операцій пов’язане саме з автентифікованим користувачем. Сервер не зберігає весь текст запиту або сам медіафайл у базі даних, але фіксує тип виконаної дії та час.

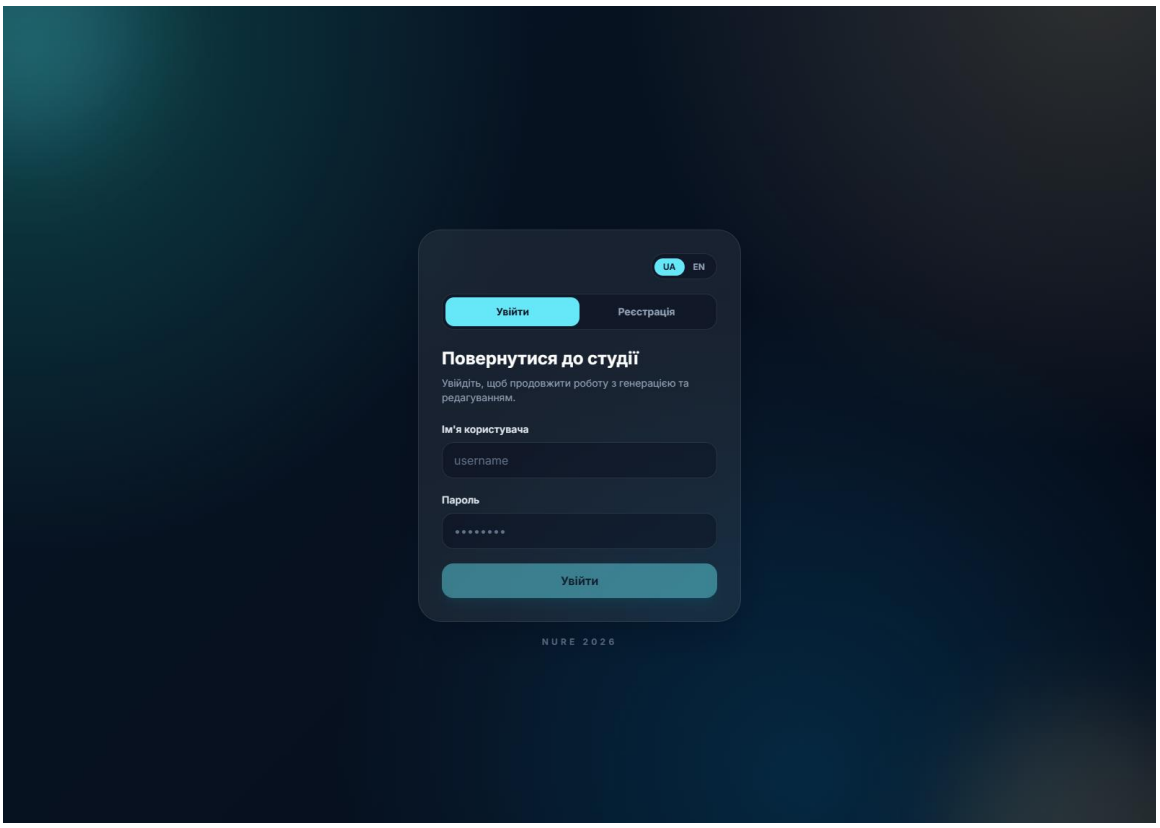

Рисунок 3.2 – Сторінка входу та реєстрації

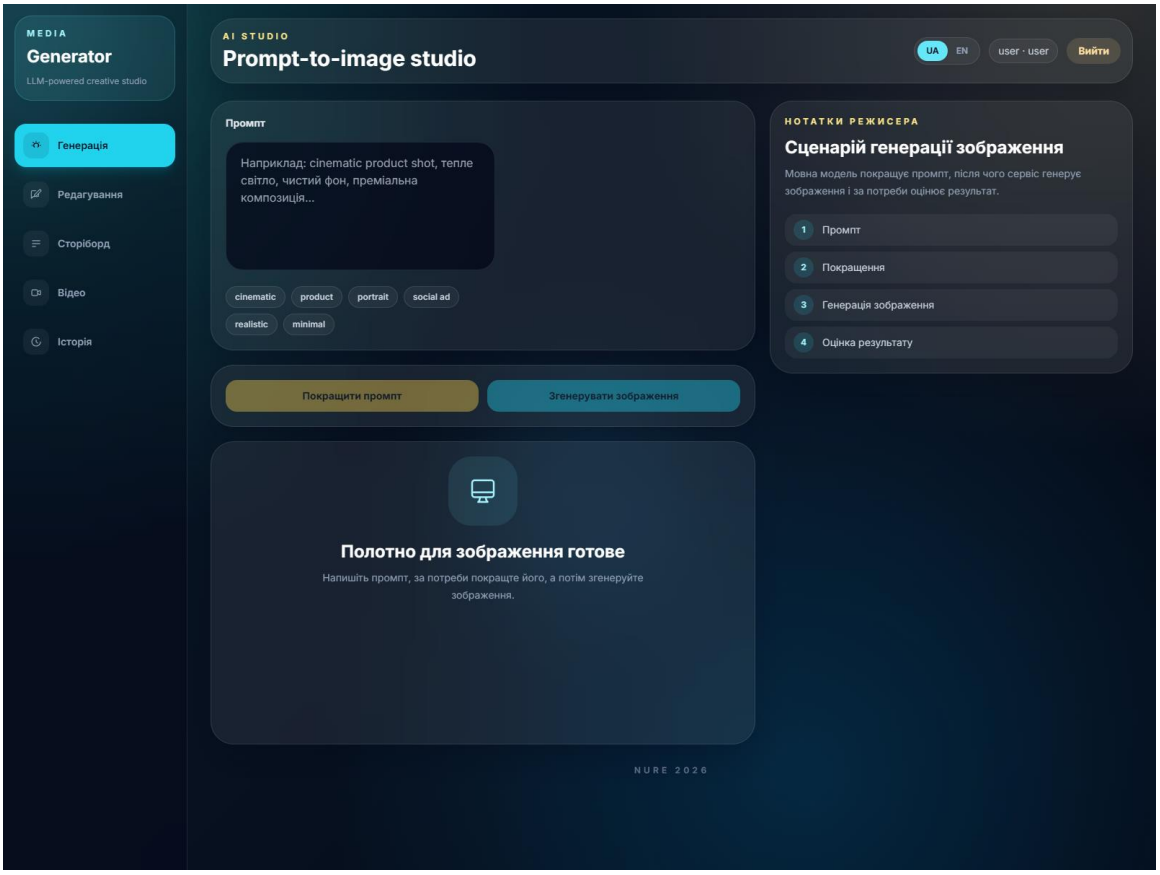

Рисунок 3.3 – Основний екран після автентифікації

Для розробленого обсягу цього достатньо, щоб показати простежуваність роботи без перевантаження бази великими файлами. Такий підхід також краще узгоджується з приватністю: завантажені зображення й отримані результати не перетворюються автоматично на постійні записи в сховищі.

3.3 Реалізація генерації зображень і відео

Режим генерації зображення працює через форму текстового запиту, стилістичні підказки, вибір співвідношення сторін і вибір постачальника сервісу. Користувач може обрати Stability, Hugging Face або локальний Test. Клієнтська частина передає на сервер текстовий опис, параметри кадру, сервіс і модель, а серверний маршрут /api/generate обирає відповідний сервісний шар.

Співвідношення сторін передається як частина задуму користувача, а не як другорядне службове поле. Горизонтальний формат зручний для широкого екрана, вертикальний – для мобільного контенту, квадратний – для універсальних ілюстрацій. Серверна частина враховує цей параметр у всіх підтримуваних режимах, а не тільки для одного сервісу, тому користувач отримує передбачуваніший формат результату.

Результат генерації повертається як base64-зображення. Для клієнтської частини не має значення, чи його отримано від Stability, Hugging Face або Test: інтерфейс працює з єдиним форматом відповіді. Додаткові службові поля з назвою сервісу й моделлю використовуються для локальної історії та уточнення виконаного сценарію. Якщо зовнішній сервіс повертає помилку, користувач бачить контрольоване повідомлення в області результату.

Для розширення набору сервісів додано інтеграційний шар Hugging Face. У конфігурації серверної частини передбачено токен, стандартну модель для зображень, постачальника сервісу, модель для редагування і модель для відео. Інтеграція реалізована через huggingface_hub і InferenceClient: сервер викликає

відповідний сервіс, перетворює результат на формат застосунку й повертає його клієнтській частині.

Hugging Face Inference Providers використовується як сучасний додатковий напрям інтеграції, але його не слід розглядати як гарантовано безкоштовний у будь-яких умовах. Доступність моделей, спосіб маршрутизації запитів через InferenceClient, квоти та вартість після вичерпання безоплатних лімітів залежать від конкретного постачальника сервісу [44–46]. Тому поряд із зовнішніми сервісами збережено локальний Test, який не залежить від токенів і зовнішніх кредитів.

Інтеграційний шар також стабілізує інтерфейс у разі змін у зовнішніх API. Якщо конкретний сервіс змінює формат відповіді, це не повинно одразу ламати логіку React-компонентів. Сервер отримує відповідь від моделі, перетворює її на внутрішній формат і тільки після цього повертає клієнту. Для користувача різні сервіси виглядають як варіанти одного списку, а для серверної частини вони залишаються окремими адаптерами з власними правилами оброблення.

У клієнтській частині список сервісів описано як набір структурованих варіантів. Для режиму генерації показуються тільки ті варіанти, які підтримують створення зображення. Якщо згодом додається новий сервіс, його потрібно внести до переліку доступних варіантів і реалізувати відповідний серверний адаптер. Клієнтська частина не потребує повного переписування, бо вона передає узгоджені поля, а не працює з внутрішньою логікою конкретної моделі.

Окремо реалізовано поведінку під час помилки зовнішнього сервісу. Якщо ключ не налаштований, сервіс повертає зрозуміле повідомлення, а інтерфейс не переходить у порожній або некерований стан. Якщо обрано Test, генерація виконується локально, але результат має той самий тип відповіді, що й зовнішній сервіс. Завдяки цьому користувач може відрізнити обмеження зовнішньої моделі від помилки власного застосунку.

На рисунку 3.4 показано сценарій генерації через Test. У полі API-сервіс вибрано Test, після запуску в області результату з’являється контрольне PNG-

зображення. Цей сценарій перевіряє повний шлях від інтерфейсу до серверного сервісного шару і назад до компонента результату.

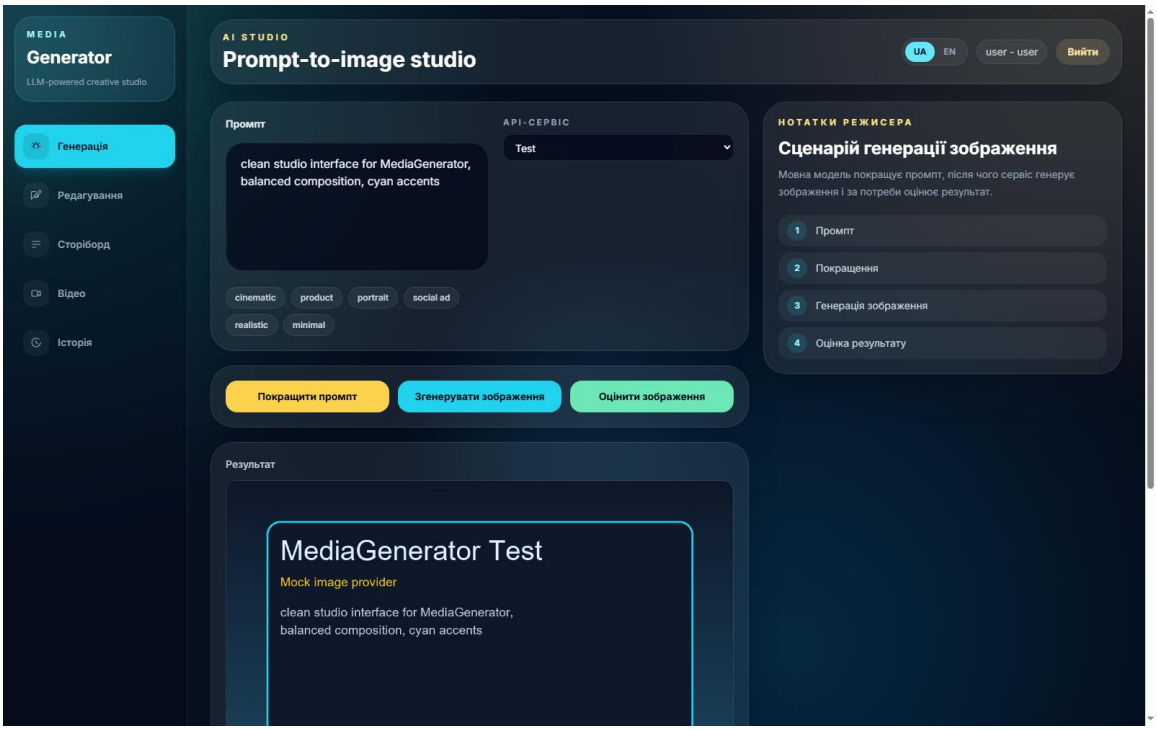

Рисунок 3.4 – Генерація зображення через Test

Відеогенерація реалізована як асинхронний робочий процес. Користувач може обрати створення відео за текстовим описом або створення відео на основі початкового зображення. У першому випадку файл не потрібний, а вся сцена описується природною мовою. У другому випадку файл використовується як опорний кадр, а текстовий опис уточнює рух, дію або зміну сцени.

Після запуску відеосценарію клієнтська частина надсилає форму на /api/video/generate. Сервер створює задачу у відповідного постачальника сервісу й повертає ідентифікатор, модель, назву сервісу та початковий статус. Далі інтерфейс періодично перевіряє стан через маршрут статусу. Такий підхід потрібний тому, що відео генерується довше за зображення і не завжди може бути повернуте в межах одного HTTP-запиту.

Під час перевірки статусу клієнтська частина не повинна залежати від назв, які використовує конкретний зовнішній сервіс. Один постачальник може

повертати processing, інший – queued, третій – власний числовий код. Сервер нормалізує такі стани до спільних значень, зрозумілих інтерфейсу. Завдяки цьому користувач бачить однакову логіку виконання задачі: очікування, оброблення, завершення або помилку, незалежно від фактичного відеосервісу.

Для відео передбачено вибір тривалості та співвідношення сторін. Тривалість впливає на характер руху: короткий ролик потребує стислого опису дії, а довший дозволяє показати плавніший розвиток сцени. Співвідношення сторін визначає формат кадру й має бути доступним користувачу до запуску задачі. Якщо обраний сервіс має власні обмеження, серверна частина обробляє їх контрольованим повідомленням.

На рівні інтерфейсу відеосценарій відрізняється від генерації зображення тривалішим життєвим циклом. Після натискання кнопки запуску користувач не одразу отримує готовий ролик, а бачить активну задачу, її статус, модель і сервіс. Це важливо для довготривалих операцій: якщо показувати тільки індикатор очікування без ідентифікатора задачі, користувач не розумітиме, чи запит було прийнято. Тому відеоблок зберігає службову інформацію про поточне завдання й оновлює її після кожної перевірки статусу.

У відеорежимі кнопка доповнення опису руху працює як засіб уточнення текстового запиту. Вона не створює рух самостійно, а додає до опису камеру, темп, напрям, дію та характер зміни сцени. Це робить назву функції змістовно точнішою: користувач отримує не абстрактне «покращення руху», а доповнений опис, який можна передати відеосервісу.

Локальний TestVideoService відтворює асинхронну логіку відео без зовнішнього сервісу. Сервер створює task_id, після короткої затримки переводить задачу до SUCCEEDED, формує службове test://-посилання, а клієнтська частина перетворює його на маршрут MP4-артефакту. У результаті користувач бачить повноцінний відеоплеєр і може завантажити саме відеофайл, а не JSON-опис задачі.

Для сценарію відео за текстом не вимагається початкове зображення. Це реалізовано через окремий тип входу: якщо користувач обирає текстовий режим,

форма не блокує запуск через відсутність файлу, а сервер формує запит на основі опису сцени. Якщо обрано режим на основі зображення, початковий кадр стає обов'язковим. Такий поділ важливий, бо два відеосценарії мають різну природу і не повинні перевірятися однією умовою.

На рисунку 3.5 показано сценарій тестового відео через Test. У правій панелі видно активне завдання, сервіс, модель і статус SUCCEEDED, а в області результату відображено відеоплеєр із локальним MP4.

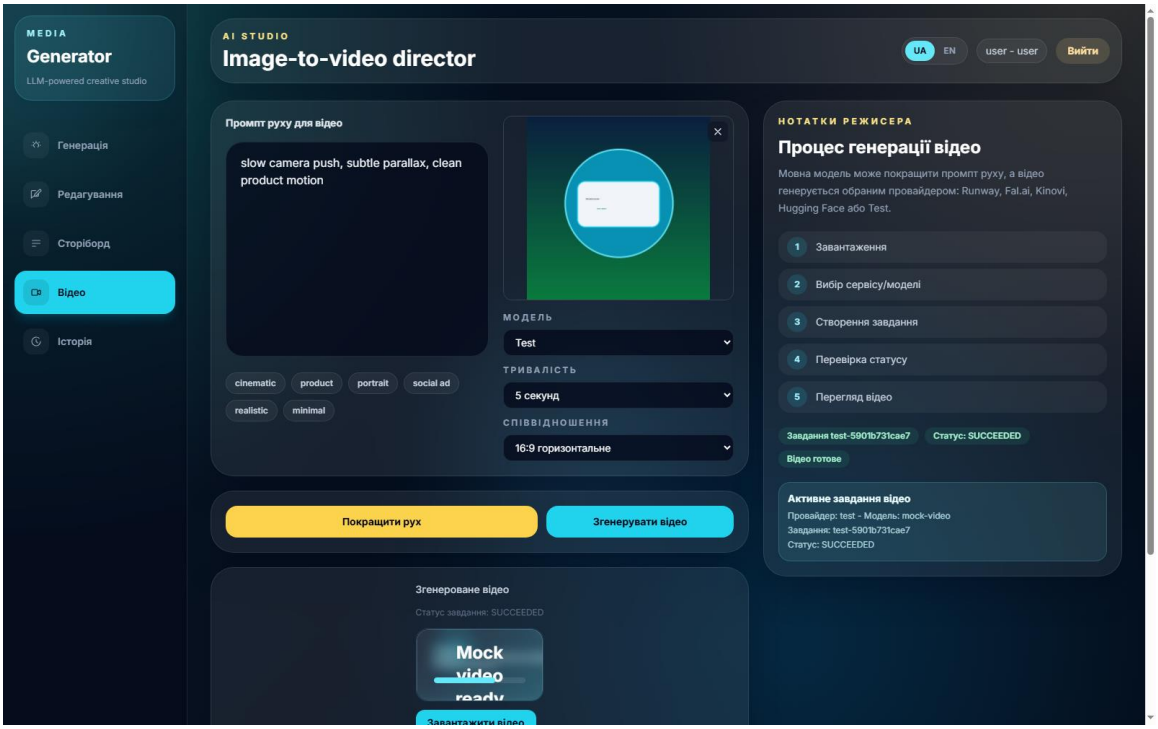

Рисунок 3.5 – Результат відеогенерації через Test

3.4 Реалізація редагування зображень

Редагування зображень реалізовано як сценарій із завантаженням файлу, введенням інструкції, вибором API-сервісу і вибором типу редагування. Користувач може описати зміну природною мовою: замінити фон, уточнити освітлення, змінити елемент або підготувати зображення до подальшої генерації

відео. Клієнтська частина передає на сервер інструкцію, файл, режим редагування, сервіс і модель.

У цьому сценарії файл і текстова інструкція працюють разом. Сам файл ще не визначає очікуваний результат, а сама інструкція без зображення не містить достатнього контексту. Тому форма редагування поєднує завантаження, опис і вибір режиму в одному екрані. Користувач бачить, що саме буде змінено, який сервіс обрано і який результат очікується після виконання.

Завантаження файлу реалізовано через компонент FileUpload. Після вибору зображення користувач бачить попередній перегляд, а файл може бути переданий до сценарію редагування або використаний як початковий кадр для відео. Це зменшує ризик помилково обрати не той матеріал і робить робочий процес більш наочним.

Редагування має більше умов, ніж генерація. Користувач може ввести загальну інструкцію, обрати режим, вказати пошуковий опис для заміни або задати масштаб. Не кожен постачальник сервісу підтримує всі режими, тому клієнтська частина фільтрує доступні варіанти, а сервер повторно перевіряє сумісність запиту. Подвійний контроль потрібний тому, що інтерфейс допомагає користувачу, але остаточну коректність операції має забезпечити серверна частина.

Серверна перевірка потрібна ще й тому, що клієнтський інтерфейс не можна вважати єдиним джерелом істини. Користувач може відправити змінений запит напряму, обрати несумісний режим або передати файл неочікуваного типу. У таких випадках сервер має не виконувати небезпечну або некоректну операцію, а повернути контрольовану помилку. Це особливо важливо для редагування, бо воно працює з користувацькими файлами й передає їх у сервісний шар.

Для локальних операцій передбачено видалення фону і масштабування. Ці дії можуть виконуватися без повної генеративної заміни сцени. Для редагування через зовнішній сервіс основним є перетворення зображення за інструкцією або

пошук і заміна фрагмента. Після виконання сервер повертає base64-зображення, а ResultDisplay показує його як результат редагування.

Масштабування зображення реалізовано як окремий режим, оскільки воно відрізняється від змістового редагування. Під час змістового редагування змінюється сцена, фон або об’єкт, а під час масштабування користувач очікує насамперед зміну розміру або підвищення деталізації без зміни основного змісту. Тому параметри масштабування мають передаватися незалежно від того, який сервіс обрано для редагування. Якщо режим підтримується локально, результат можна перевірити без зовнішнього ключа; якщо використовується зовнішній сервіс, сервер має передати відповідні параметри або повернути контрольоване повідомлення про обмеження.

Під час редагування важливо не переносити допоміжні панелі з інших режимів. Якщо користувач покращив запит у генерації, ця відповідь не повинна залишатися видимою під час редагування. Тому клієнтська частина зберігає окремий стан для кожного режиму: власний текстовий запит, файл, результат, помилку, допоміжну відповідь мовної моделі та відеостан. Після повернення до попереднього режиму його контекст може залишатися доступним, але не змішується з іншими вкладками.

План редагування виконує роль проміжного етапу між наміром користувача і викликом сервісу. Він допомагає окремо побачити ціль зміни, елементи, які потрібно зберегти, і уточнений текстовий запит. Для користувача це корисно, бо редагування завантаженого зображення має бути обережнішим, ніж генерація нового. Якщо потрібно змінити фон, але не змінити головний об’єкт, план редагування робить це обмеження явним до запуску сервісу.

На практиці Test-редагування особливо корисне для перевірки завантаження файлу. Якщо локальний сервіс повертає PNG, це означає, що клієнтська частина правильно сформувала запит, сервер отримав файл, сервісний шар виконав операцію, результат було закодовано в base64, а компонент результату зміг його показати. Після цього помилки зовнішнього сервісу можна розглядати окремо.

На рисунку 3.6 наведено сценарій редагування через Test. Скріншот показує завантажений файл, вибір сервісу, режим редагування і результат.

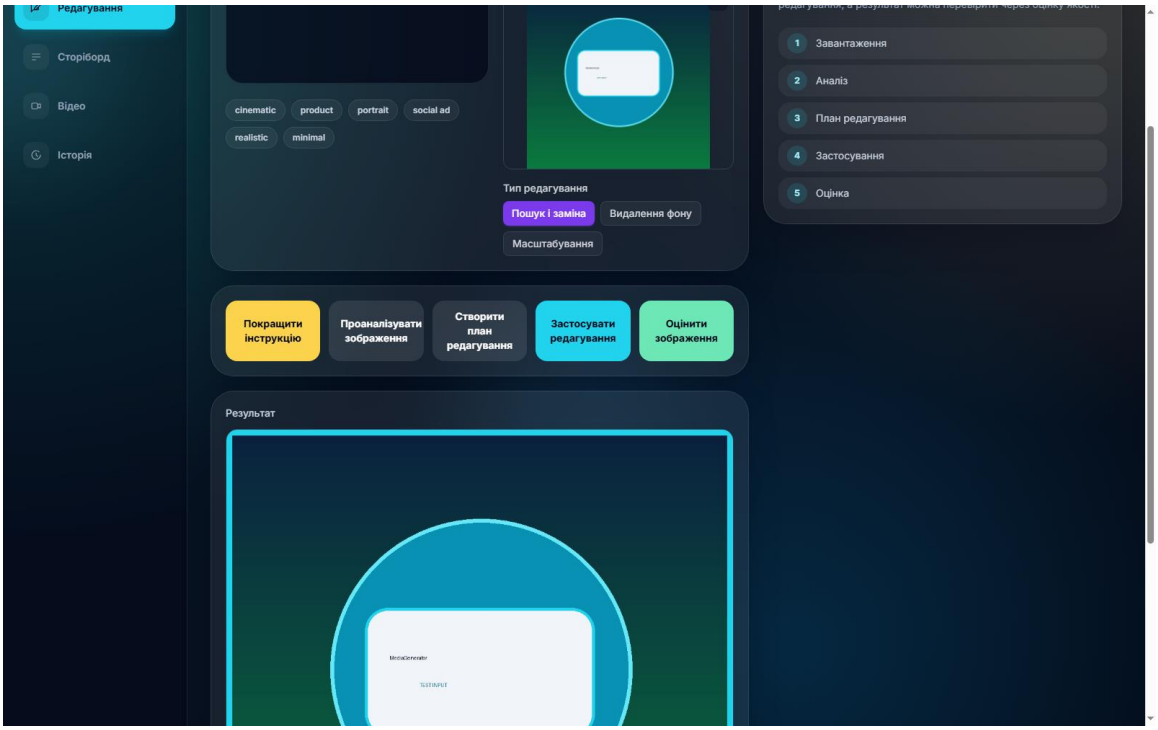

Рисунок 3.6 – Редагування зображення через Test

У таблиці 3.2 подано локальні сценарії режиму Test, які використовуються для перевірки генерації, редагування та відео.

Таблиця 3.2 – Локальні сценарії режиму Test

Сценарій	**Що перевіряється**	**Зовнішній сервіс**
1	**2**	**3**
provider=test у /api/generate	текстовий запит, JWT, вибір сервісу, base64-зображення	не потрібний
provider=test у /api/edit	завантаження файлу, маршрут редагування, повернення результату	не потрібний

Продовження таблиці 3.2

1	2	3
provider=test у /api/video/generate	створення ідентифікатора задачі та перевірка статусу	не потрібний
test:// у ResultDisplay	перетворення на MP4-артефакт і показ у відеоплеєрі	не потрібний

3.5 Адміністративна панель і допоміжні сценарії

Окрім генерації й редагування, MediaGenerator містить допоміжні сценарії мовної моделі. Покращення текстового запиту перетворює короткий опис на конкретнішу інструкцію. План редагування допомагає відокремити об’єкти зміни від елементів, які потрібно зберегти. Аналіз зображення описує завантажений файл, а оцінювання результату зіставляє його з початковим запитом [47, 48]. Розкадрування перетворює ідею відео на послідовність сцен, опис руху камери, підказки для озвучення і запити для подальшої генерації.

Усі ці сценарії мають спільну роль: мовна модель не створює медіа безпосередньо, а допомагає сформулювати, уточнити або перевірити змістовний намір користувача. Наприклад, коротка фраза може бути перетворена на повніший опис сцени, а загальна інструкція редагування – на план із конкретними об’єктами та очікуваними змінами. Така допомога не замінює оцінку користувача, але робить проміжний задум видимим.

Відповіді мовної моделі подаються у структурованому вигляді. Для покращення запиту це уточнений опис і пояснення, для плану редагування – мета зміни та елементи збереження, для розкадрування – послідовність сцен. Такий формат потрібний для стабільного відображення в інтерфейсі: користувач не отримує довільний великий абзац там, де очікує короткий план дій. Якщо

зовнішня мовна модель повертає неповну відповідь, сервер може сформувати резервну структуру, щоб інтерфейс залишився працездатним.

Панель Director Notes у правій частині інтерфейсу виконує роль пояснювального шару. У ній відображаються активний сценарій, етапи генерації або редагування, статус відеозадачі, назва сервісу, модель та ідентифікатор задачі. Для відео ця панель особливо важлива, оскільки користувач не отримує результат миттєво і має бачити стан оброблення.

Для сценарію розкадрування ця панель дає змогу подати результат не як суцільний абзац, а як послідовність майбутніх сцен. Користувач бачить короткий опис кадру, рух камери, тривалість і підказку для подальшої генерації. Такий формат зручний тому, що відео рідко створюється з однієї фрази: перед запуском ролика потрібно зрозуміти, що саме має відбутися в кадрі й як сцена змінюється в часі.

Локальна історія дій показує останні операції сесії. Вона не є повноцінним сховищем медіафайлів, але дає користувачу змогу бачити послідовність виконаних дій: генерацію, редагування, розкадрування, відеозадачу або оцінювання зображення мовною моделлю. Серверний OperationLog доповнює цей механізм і фіксує типи операцій на рівні серверної частини.

Історія не замінює серверну базу результатів, але виконує важливу інтерфейсну функцію. У творчому процесі користувач часто повертається до попереднього запиту, порівнює кілька результатів або перевіряє, який сервіс був обраний. Локальний список останніх дій дає цей контекст без ускладнення серверної моделі даних. Водночас він не змішується з журналом операцій, який потрібний для базової простежуваності на сервері.

На рисунку 3.7 показано локальну історію робочого процесу. Вона дає змогу швидко повернутися до останніх дій у межах поточного сеансу і не змішує результати різних режимів.

Адміністративна панель демонструє роботу ролей і захищених маршрутів. Адміністратор може переглянути користувачів, тоді як звичайний користувач не має доступу до цієї інформації. Такий поділ показує, що система не зводиться до

одного відкритого маршруту генерації, а має базову структуру вебзастосунку з автентифікацією, ролями та обмеженнями доступу.

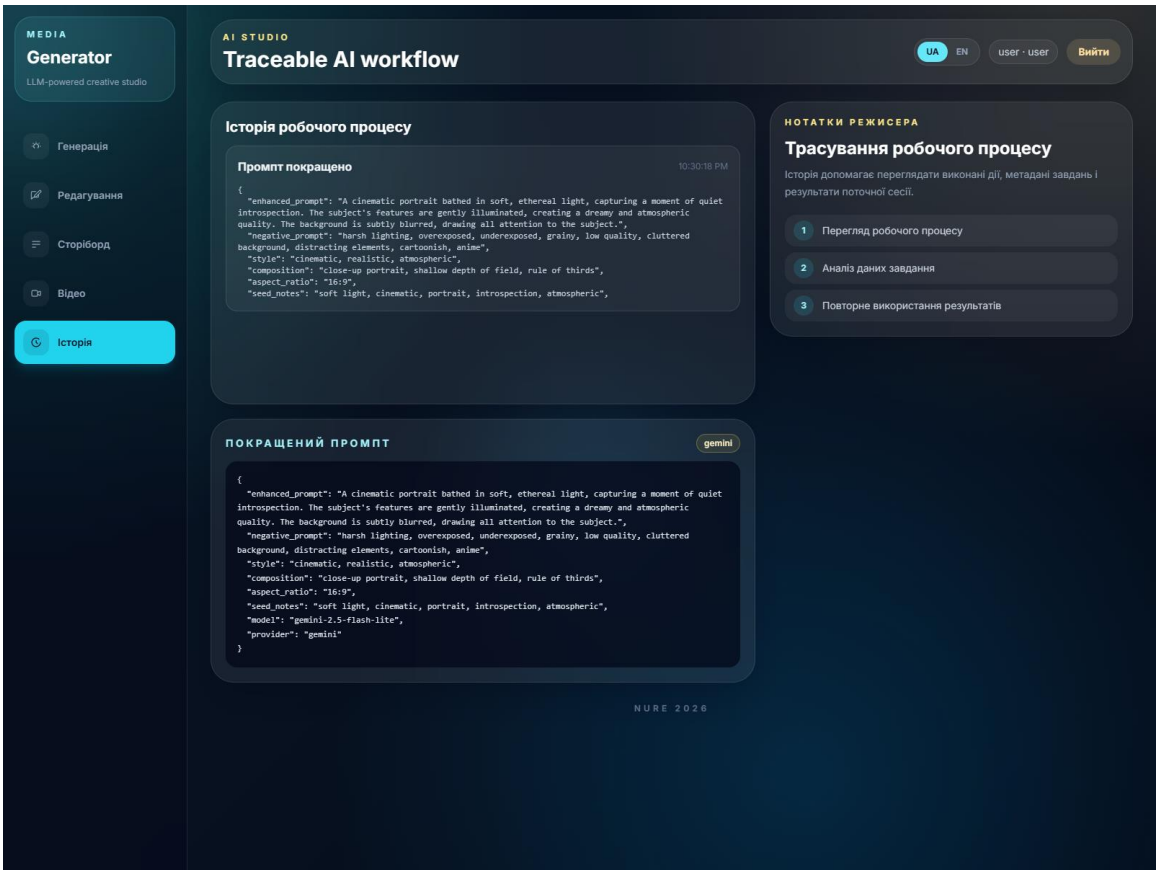

Рисунок 3.7 – Локальна історія робочого процесу

Наявність адміністративної панелі також перевіряє, що сервер не довіряє лише клієнтському інтерфейсу. Навіть якщо користувач спробує звернутися до адміністративного маршруту напряму, серверна частина повинна перевірити роль. Тому панель є не тільки додатковим екраном, а й практичним підтвердженням рольової моделі: права доступу визначаються на сервері, а клієнтська частина лише відображає дозволені можливості.

На рисунку 3.8 наведено адміністративну панель користувачів. Її наявність підтверджує, що в системі враховано не лише творчий сценарій, а й керування доступом.

Практичний результат реалізації полягає в тому, що MediaGenerator підтримує повний цикл роботи з генеративними медіа: вхід у систему, генерацію

зображень за текстовим запитом, природномовне редагування зображень, допоміжні сценарії мовної моделі, розкадрування, створення відео з тексту або зображення, локальну історію і адміністративний перегляд.

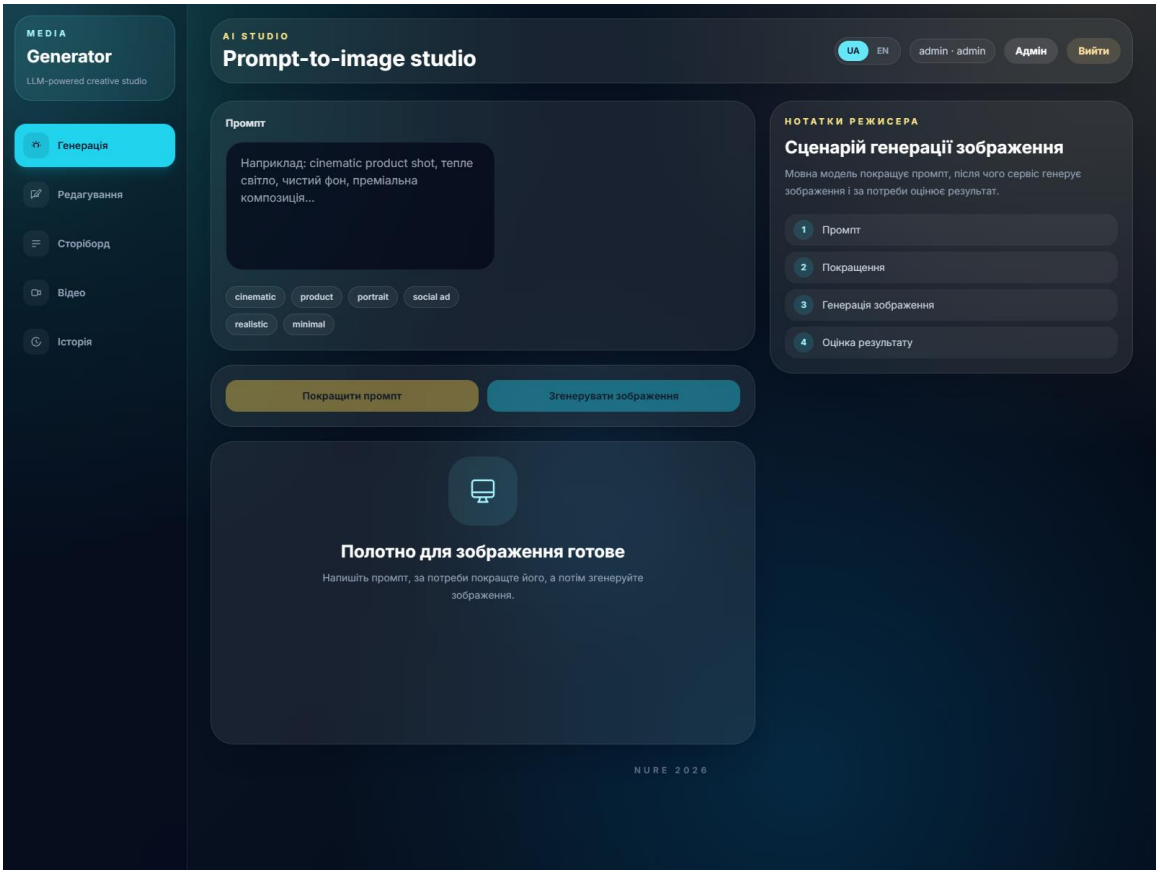

Рисунок 3.8 – Адміністративна панель користувачів

Вибір сервісу, інтеграція Hugging Face і локальний режим Test роблять застосунок стійкішим до обмежень зовнішніх ключів і квот.

У результаті користувач отримує не набір розрізнених сторінок, а послідовну студію. Дії пов’язані між собою спільним інтерфейсом, окремими станами вкладок, єдиним підходом до результатів і зрозумілим відображенням статусів. Саме ця послідовність є практичним втіленням теми роботи, оскільки мовні моделі використовуються не ізольовано, а як частина робочого процесу редагування зображень і генерації відео.

3.6 Тестування працездатності

Перевірку MediaGenerator виконано за сценарним принципом. Для такого застосунку недостатньо переконатися, що окремий маршрут повертає відповідь. Потрібно пройти весь шлях користувача: вхід у систему, вибір режиму, передавання текстового запиту або файлу, запуск сервісу, отримання результату, відображення його в інтерфейсі та фіксацію дії в історії. Окремо перевірено випадки, у яких зовнішні сервіси недоступні, а локальний Test має підтвердити працездатність власної логіки застосунку.

Спочатку виконано технічні перевірки серверної та клієнтської частин. Серверна частина перевірялася через тести FastAPI, а клієнтська – через збирання Vite. Також перевірено запуск застосунку з окремою локальною папкою даних, щоб контрольні облікові записи й журнали операцій не змішувалися з основними файлами. Такий підхід дає змогу повторювати перевірки в однакових умовах.

Технічні перевірки мають різне призначення. Збирання клієнтської частини підтверджує, що інтерфейсні компоненти, стилі, імпорти та локалізація не мають помилок, які завадили б запуску студії. Перевірки серверної частини підтверджують, що маршрути приймають очікувані дані, повертають узгоджений формат і не пропускають приватні операції без автентифікації. Разом ці два рівні не замінюють браузерний сценарій, але відсікають базові помилки до ручної перевірки інтерфейсу.

На рівні інтерфейсу перевірялися не лише успішні відповіді, а й поведінка екранів після переходу між вкладками. Якщо користувач у режимі генерації натискає кнопку покращення текстового запиту, відповідна панель має залишатися в межах генерації. Після переходу до редагування або відео вона не повинна відображатися як результат іншого режиму. Так само відеорезультат має залишатися в межах відеосценарію, а не з’являтися в режимах генерації чи редагування.

Окремо перевірено параметри зображень і відео. Співвідношення сторін має впливати на формат результату в усіх підтримуваних режимах, а не тільки для одного сервісу. Для відео перевірено ширший вибір тривалості, режим створення відео на основі зображення і режим перетворення текстового опису на відео без початкового фото. Також перевірено, що кнопка доповнення опису руху змінює саме текстовий запит, а не створює окремий непов'язаний результат.

Під час перевірки окремо фіксувалося, чи зберігається відповідність між дією користувача і тим, що відображається на екрані. Для генерації важливим є показ саме зображення, для редагування – показ результату на основі завантаженого файлу, для відео – поява відеоплеєра після завершення задачі. Якщо система замість відео показує маленьке статичне зображення або завантажує службовий JSON, користувач не може перевірити кінцеву функцію. Тому для відеосценарію перевірявся не тільки статус SUCCEEDED, а й фактичне відтворення MP4-артефакту.

Для API-перевірок використано локальний режим Test. У таблиці 3.3 наведено основні маршрути, очікувані результати й фактичний стан виконання.

Таблиця 3.3 – API-перевірки локального режиму Test

Запит	**Очікуваний результат**	**Статус**
1	**2**	**3**
POST /api/auth/register	створено звичайного користувача	успішно
POST /api/auth/login	повернено JWT	успішно
POST /api/generate, provider=test	повернено PNG у base64	успішно
POST /api/edit, provider=test	повернено PNG у base64	успішно

Продовження таблиці 3.3

1	2	3
POST /api/video/generate, provider=test	створено task_id	успішно
GET /api/video/status/test/{task_id}	статус переходить до SUCCEEDED	успішно
GET /api/video/artifacts/test/{task_id}.mp4	повернено MP4-файл	успішно

Також перевірено маршрут /api/health/full. Він повертає загальний стан системи і список доступних сервісів зображень, серед яких є test. Це дає змогу швидко визначити, чи сервер після запуску бачить локальний сервісний шар і може відповідати на контрольні запити.

Генерація та редагування повертають JSON, у якому зображення передається як base64-рядок. Разом із зображенням сервер може передати назву сервісу, модель і режим. Для відео використовується інший контракт: створення task_id, перехід статусу до SUCCEEDED, поява test://-посилання та отримання реального MP4 через внутрішній маршрут. Завдяки цьому кнопка завантаження відео працює з відеофайлом, а не зі службовим JSON-описом.

Такий контракт зручний для повторної перевірки. Якщо зображення не відображається, перевіряються кодування base64, формат JSON і компонент результату. Якщо не працює відео, аналізується ланцюг із трьох кроків: створення задачі, перевірка статусу і завантаження MP4. Це важливо, бо успішне створення задачі ще не гарантує, що користувач побачить фінальний ролик. У MediaGenerator ці кроки розділено, але вони поєднані в одному користувацькому сценарії.

Другий рівень перевірки виконано через браузерний сценарій. Користувач входить у систему, відкриває режим генерації, обирає Test, вводить текстовий запит і отримує контрольне зображення. Далі відкривається режим редагування: користувач завантажує зображення, обирає Test, виконує редагування й

переглядає результат. Після цього перевіряється режим відео, у якому можна створити задачу на основі зображення або повністю за текстовим описом, дочекатися статусу SUCCEEDED і переглянути MP4 у плеєрі.

На рисунках 3.4–3.6 наведено основні екрани після виконання цих дій. Вони підтверджують фактичну роботу інтерфейсу після входу користувача: вибір сервісу Test, контрольний результат генерації зображення, редагування завантаженого файлу та тестове відео зі статусом SUCCEEDED.

На екрані генерації видно, що користувач обрав локальний сервіс і отримав зображення в області результату. На екрані редагування видно завантажений файл і результат після виконання операції. На екрані відео перевіряється саме наявність відеоплеєра, а не лише службового статусу. Такі візуальні підтвердження важливі, бо частина помилок виникає не на рівні маршруту, а під час показу результату.

У таблиці 3.4 наведено браузерні сценарії, які підтверджують працездатність основних екранів.

Таблиця 3.4 – Браузерні сценарії

Сценарій	**Скріншот**	**Що підтверджено**
Генерація через Test	Рисунок 3.4	вибір сервісу, текстовий запит, base64-результат
Редагування через Test	Рисунок 3.5	завантаження файлу, режим редагування, результат
Відео через Test	Рисунок 3.6	ідентифікатор задачі, перевірка статусу, SUCCEEDED, MP4-плеєр

Негативні сценарії перевірялися окремо, оскільки застосунок працює із зовнішніми сервісами штучного інтелекту. Користувач може не мати ключа

доступу до певного сервісу, зовнішня квота може завершитися, модель може бути тимчасово недоступною або сервіс може повернути нестандартний статус. У таких випадках MediaGenerator має показувати контрольоване повідомлення, а не внутрішній технічний слід помилки.

Якщо ключ Stability або Hugging Face відсутній, сервер повертає зрозумілу помилку сервісу. Якщо використовується Test, зовнішній ключ не потрібний. Якщо відеосценарій повертає test://, клієнтська частина не намагається завантажити JSON як відео, а звертається до MP4-артефакту. Якщо вибрана модель не відповідає режиму, список варіантів фільтрується або значення скидається на сумісний варіант.

Негативні сценарії не розглядаються як другорядні. У застосунку з кількома зовнішніми сервісами саме вони найчастіше визначають, чи буде користувацький досвід зрозумілим. Помилка ключа, квоти або недоступної моделі не повинна виглядати як випадкове зникнення результату. Тому очікувана реакція системи формулюється для кожного типу проблеми: заборона доступу, повідомлення про недоступність сервісу, перехід до локального режиму або збереження результату в межах правильної вкладки.

Для користувача робота з MediaGenerator починається зі сторінки входу або реєстрації. Після входу відкривається студія з режимами генерації, редагування, розкадрування, відео та історії. Якщо потрібно перевірити систему без зовнішніх ключів, у полі вибору API-сервісу обирається Test. Цей варіант доступний у генерації зображень, редагуванні та відеосценарії.

У таблиці 3.5 подано негативні сценарії та очікувану реакцію системи.

Таблиця 3.5 – Негативні сценарії та очікувана реакція

Ситуація	**Очікувана реакція системи**
1	**2**
Запит без JWT	приватний маршрут не виконується
Звичайний користувач відкриває адміністративний список	доступ заборонено

Продовження таблиці 3.5

1	2
Відсутній зовнішній API-ключ	повертається зрозуміла помилка сервісу
Вичерпано зовнішню квоту	основний шлях застосунку перевіряється через Test
test:// як video URL	формується MP4-артефакт і показується відеоплеєр
Перехід між вкладками	результат залишається в межах відповідного режиму

У режимі генерації користувач вводить текстовий запит, обирає співвідношення сторін і сервіс. Після запуску застосунок показує стан очікування, а потім виводить контрольне або згенероване зображення. Якщо обрано зовнішній сервіс, результат залежить від доступності ключа, моделі та квоти; якщо обрано Test, результат створюється локально.

У режимі редагування користувач завантажує зображення, вводить інструкцію й обирає режим. Наприклад, можна змінити фон, покращити освітлення або виконати масштабування. Для Test результат є контрольним PNG, який підтверджує правильність завантаження файлу, передавання запиту й відображення результату. Для зовнішнього сервісу очікується реальне редагування, якщо відповідна модель підтримує обраний сценарій.

У режимі відео користувач обирає створення відео за текстом або на основі зображення. Для сценарію із зображенням потрібно завантажити початковий кадр, ввести опис руху, обрати сервіс, модель, тривалість і співвідношення сторін. Для сценарію за текстом початкове зображення не потрібне, а вся сцена описується природною мовою. Після запуску застосунок показує ідентифікатор задачі та статус. Для Test статус після короткого очікування переходить до SUCCEEDED, а в області результату з’являється відеоплеєр.

Практичний результат перевірки полягає в тому, що застосунок підтримує повний цикл роботи з генеративними медіа без обов'язкової залежності від зовнішніх кредитів. Користувач може увійти, обрати Test, створити контрольне зображення, виконати редагування завантаженого файлу, пройти сценарій створення відео за текстом або на основі зображення до статусу SUCCEEDED і переглянути відео в плеєрі. Якщо зовнішні ключі доступні, той самий інтерфейс може звертатися до Stability, Hugging Face або відеосервісів.

Також перевірено, що тестовий режим не змінює логіку користувацького сценарію. Користувач так само вводить запит, обирає сервіс, очікує результат і бачить його в робочій області. Відмінність полягає лише в тому, що результат формується локально і є контрольованим. Це дає змогу відпрацювати форму запиту, стан очікування, обробку відповіді, історію й завантаження результату без витрат зовнішніх сервісів.

Перевірка також показала, що важливим практичним результатом є відокремлення власної логіки застосунку від зовнішніх залежностей. Якщо Test проходить, але зовнішній сервіс ні, насамперед аналізуються ключ, квота, мережа або модель. Якщо не проходить навіть Test, причина знаходиться у власному клієнтсько-серверному сценарії. Такий поділ спрощує налагодження і робить поведінку системи зрозумілішою.

Окремий результат стосується створення відео за текстом. Для сервісу генерації відео не можна обмежуватися тільки сценарієм із початковим зображенням. Іноді користувач має лише текстовий задум і не має готового кадру. Тому у відеорежимі передбачено вибір типу входу: текстовий опис або початкове зображення. У першому випадку файл не потрібний, у другому – обов'язковий.

Загальний результат тестування підтверджує, що MediaGenerator працює як цілісна студія, а не як набір ізольованих форм. Користувач може перейти від генерації до редагування, потім до відео, а після цього переглянути історію виконаних дій. При цьому допоміжні панелі не переносяться між вкладками, а результат кожного режиму залишається пов'язаним із власним сценарієм. Це

особливо важливо для системи з мовними моделями, де проміжні відповіді можуть бути схожими за виглядом, але різними за змістом.

3.7 Обмеження поточної реалізації і напрями розвитку

MediaGenerator має визначені обмеження. Він не навчає власну генеративну модель, не гарантує художню якість зовнішнього синтезу, не зберігає медіафайли як повноцінну бібліотеку і не має розширеного керування витратами зовнішніх API. Сторонні сервіси можуть змінювати умови доступу, статуси, моделі та формати відповідей, тому застосунок зберігає контрольований локальний сценарій і зрозумілу обробку помилок.

Ці обмеження не зменшують практичну цінність розробленого застосунку, але визначають межі його використання. MediaGenerator призначений для організації робочого процесу з генеративними сервісами, а не для заміни самих моделей. Якість зображення або відео залежить від обраного сервісу, текстового опису, доступної моделі та умов виконання. Власна частина застосунку відповідає за керованість цього процесу: приймання даних, вибір сервісу, показ результату, обробку помилок і контрольований тестовий сценарій.

Першим напрямом розвитку є серверне збереження медіарезультатів. Локальна історія корисна для поточного сеансу, але після перезавантаження сторінки вона зникає. Повноцінна медіабібліотека потребувала б окремих сутностей у базі даних, файлового або об’єктного сховища, політики очищення файлів і контролю доступу до результатів.

Під час такого розширення потрібно врахувати різницю між текстовими запитами, зображеннями та відео. Текстовий запит займає мало місця, але може містити приватну інформацію. Зображення може бути важливим для повторного редагування, але потребує зберігання файлу й контролю доступу. Відео має більший розмір і довший життєвий цикл задачі. Тому майбутня медіабібліотека

не повинна бути просто таблицею з усіма результатами: для неї потрібні окремі правила збереження, видалення, приватності й повторного використання.

Другим напрямом є розширення редагування зображень. Реалізована логіка підтримує інструкційне редагування, локальні операції та сценарій через зовнішній сервіс. Для складніших задач доцільно додати маски, розширення меж зображення, опорне зображення, контроль збереження об’єкта та попередній перегляд області зміни. Це зробило б редагування точнішим і ближчим до професійного робочого процесу.

Третім напрямом є поглиблення відеосценаріїв. Створення відео за текстом і на основі зображення можна доповнити чергою задач, серверним збереженням статусів, повторним запуском із тим самим запитом, вибором кількох варіантів результату та докладнішим розкадруванням. Для генеративного відео це особливо важливо, оскільки один текстовий опис може давати різні результати, а користувачу потрібно порівнювати варіанти.

Для відео також доцільно розширити роботу зі статусами. Зараз користувач бачить активну задачу і фінальний результат, але в майбутньому можна додати чергу кількох задач, повторну перевірку після оновлення сторінки та збереження історії статусів на сервері. Це дасть змогу краще працювати з довготривалими сервісами, де результат може з’являтися не одразу, а після очікування в черзі зовнішнього постачальника.

Четвертим напрямом є розвиток простежуваності. Зараз серверний журнал фіксує типи операцій, а локальна історія показує дії в межах сесії. Для повнішої версії доцільно зберігати зв’язок між користувачем, текстовим запитом, сервісом, моделлю, результатом і часом виконання, але робити це з урахуванням приватності. Не кожний запит або завантажений файл можна безпечно зберігати в базі даних, тому потрібна політика вибіркового збереження та очищення медіаартефактів.

Окремо варто розвивати відтворюваність результатів. Для генеративних сервісів один і той самий запит не завжди означає однаковий медіафайл, тому в історії доцільно фіксувати не тільки текст, а й сервіс, модель, seed або інші

службові параметри, якщо їх підтримує обраний API. Локальний режим Test частково розв’язує це завдання, бо повертає передбачуваний результат і дає змогу перевірити маршрут незалежно від зовнішньої моделі.

Ще одним напрямом є покращення користувацької оцінки результатів. Зараз система може показати результат і допоміжні відповіді мовної моделі, але не має окремого механізму порівняння кількох варіантів. У майбутньому доцільно додати позначення вдалих результатів, повторний запуск із тим самим запитом, збереження кількох варіантів і короткий коментар користувача. Це особливо корисно для генеративних медіа, де перша відповідь моделі не завжди є найкращою.

З позиції безпеки подальший розвиток має включати обмеження CORS, обов’язковий стійкий SECRET_KEY, захист від надмірної кількості запитів, контроль розміру файлів, перевірку MIME-типу та детальнішу політику журналювання. Такі зміни не змінюють основну ідею MediaGenerator, але потрібні для переходу від локального застосунку до сервісу з ширшим колом користувачів.

ВИСНОВКИ

У кваліфікаційній роботі на тему «Застосування мовних моделей для редагування зображень і генерації відео» розв’язано прикладну задачу створення вебзастосунку MediaGenerator для роботи з генеративними медіа через природномовні інструкції. Розроблений застосунок поєднує генерацію зображень, редагування завантажених файлів, покращення текстових запитів за допомогою мовної моделі, аналіз зображень, розкадрування, генерацію відео за текстовим описом і на основі початкового зображення, автентифікацію, ролі користувачів, локальну історію й журналювання операцій.

У процесі виконання роботи отримано такі результати:

– зроблено аналіз сучасних підходів до застосування мовних моделей у генеративних медіасистемах і визначено їхню роль у природномовному керуванні процесами створення, аналізу та редагування медіаконтенту;

– розглянуто особливості генерації зображень за текстовим описом і природномовного редагування зображень, зокрема залежність результату від якості запиту, вибору сервісу, параметрів генерації та обмежень зовнішніх API;

– визначено вимоги до сценаріїв перетворення текстового опису на відео та зображення на відео, зокрема потребу у виборі тривалості, співвідношення сторін, постачальника сервісу, перевірці статусу задачі та отриманні готового відеорезультату;

– обґрунтовано архітектуру вебзастосунку MediaGenerator, яка поєднує клієнтську частину, серверний прикладний інтерфейс, автентифікацію, базу даних і окремі шари взаємодії із сервісами генерації зображень та відео;

– реалізовано генерацію зображень із можливістю вибору сервісу, введення текстового опису, налаштування співвідношення сторін і отримання результату через інтегровані або тестові механізми;

– реалізовано редагування завантажених зображень, зокрема сценарії природномовного редагування, зміну масштабу, використання тестового режиму та передавання даних між клієнтською і серверною частинами застосунку;

– реалізовано сценарії покращення запиту, аналізу зображення, формування плану редагування і побудови розкадрування, що дає змогу використовувати мовну модель як допоміжний інструмент для підготовки медіазавдань;

– реалізовано генерацію відео з тексту або зображення з вибором тривалості, співвідношення сторін, постачальника сервісу та перевіркою статусу виконання задачі;

– забезпечено локальний тестовий режим для зображень і відео, який дає змогу перевіряти роботу MediaGenerator без обов'язкового використання зовнішніх токенів, платних квот або доступу до сторонніх сервісів;

– перевірено працездатність основних сценаріїв застосунку, зокрема реєстрації та входу користувача, генерації зображення, редагування зображення, створення відеозадачі, отримання статусу виконання, відтворення тестового MP4-результату, роботи історії та адміністративного перегляду.

Перевірка працездатності виконана на рівні технічного запуску, прикладних маршрутів і браузерної взаємодії. Перевірено вхід користувача, генерацію через Test, редагування через Test, створення відеозадачі, перехід статусу до SUCCEEDED, отримання MP4-артефакту, роботу маршруту перевірки стану системи та ізоляцію результатів між вкладками. Такий підхід підтверджує працездатність власного клієнтсько-серверного сценарію без обов'язкової залежності від зовнішніх токенів і квот.

Практичне значення роботи полягає у створенні функціонально цілісного вебзастосунку, який демонструє інтеграцію мовних моделей і генеративних сервісів у реальний користувацький процес. Система не навчає власну генеративну модель і залежить від доступності зовнішніх сервісів, однак саме це відповідає прикладному характеру роботи: основну увагу приділено не навчанню нейронної мережі, а розробленню керованого програмного середовища для генерації, редагування та перевірки медіарезультатів.

Поставлену мету досягнуто: розроблено MediaGenerator як вебзастосунок для генерації, аналізу, редагування зображень і генерації відео на основі мовних

моделей та зовнішніх сервісів. Подальший розвиток може включати серверне збереження медіарезультатів, підтримку масок для точнішого редагування, розширення сценаріїв розкадрування, додавання нових сервісів, автоматизовані браузерні тести, посилення безпеки й докладніший контроль витрат зовнішніх API.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., & Hudáková, M. (2026). Feature space quantization and application of metrics in structural methods of image recognition. IEEE Access, 14, 17812–17824.

2. Гороховатський, В., & Творошенко, І. (2026). Продуктивні моделі аналізу даних у методах розпізнавання зображень: Монографія. Харків: ХНУРЕ.

3. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., & Hudáková, M. (2025). Image description compression in classification structural methods. IEEE Access, 13, 43631–43641.

4. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image classification methods. Advanced Information Systems, 9(1), 5–12.

5. Gorokhovatsky, V. A., & Putyatin, Ye. P. (2009). Image likelihood measures of the basis of the set of conformities. Telecommunications and Radio Engineering, 68(9), 763–778.

6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 27, 2672–2680.

7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008.

8. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.

9. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021).

Learning transferable visual models from natural language supervision. Proceedings of the 38th International Conference on Machine Learning, 139, 8748–8763.

10. Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. Proceedings of the 32nd International Conference on Machine Learning, 37, 2256–2265.

11. Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems, 33, 6840–6851.

12. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10684–10695.

13. Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., & Sutskever, I. (2021). Zero-shot text-to-image generation. Proceedings of the 38th International Conference on Machine Learning, 139, 8821–8831.

14. Nichol, A. Q., Dhariwal, P., Ramesh, A., Shyam, P., Mishkin, P., McGrew, B., Sutskever, I., & Chen, M. (2022). GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. Proceedings of the 39th International Conference on Machine Learning, 162, 16784–16804.

15. Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Lopes, R. G., Karagol Ayan, B., Salimans, T., Ho, J., Fleet, D. J., & Norouzi, M. (2022). Photorealistic text-to-image diffusion models with deep language understanding. Advances in Neural Information Processing Systems, 35, 36479–36494.

16. Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., & Vlasenko, N. (2023). Explanation of CNN image classifiers with hiding parts. In Explainable deep learning artificial intelligence (pp. 125–146). Academic Press.

17. McKinney, W. (2022). Python for data analysis: Data wrangling with pandas, NumPy, and Jupyter (3rd ed.). O'Reilly Media.

18. Hertz, A., Mokady, R., Tenenbaum, J., Aberman, K., Pritch, Y., & Cohen-Or, D. (2022). Prompt-to-prompt image editing with cross attention control. arXiv. URL: https://arxiv.org/abs/2208.01626 (дата звернення 20.05.2026).

19. Brooks, T., Holynski, A., & Efros, A. A. (2023). InstructPix2Pix: Learning to follow image editing instructions. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 18392–18402.

20. Mokady, R., Hertz, A., Aberman, K., Pritch, Y., & Cohen-Or, D. (2023). Null-text inversion for editing real images using guided diffusion models. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 6038–6047.

21. Zhang, L., Rao, A., & Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. Proceedings of the IEEE/CVF International Conference on Computer Vision, 3836–3847.

22. Gorokhovatskyi, V. A. (2003). Recognition of images in conditions of incomplete information. Kharkiv: KNURE.

23. Gorokhovatskyi, V. A. (2018). Image classification methods in the space of descriptions in the form of a set of the key point descriptors. Telecommunications and Radio Engineering, 77(9), 787–797.

24. Gadetska, S., Gorokhovatskyi, V., & Stiahlyk, N. (2020). Image structural classification technologies based on statistical analysis of descriptions in the form of bit descriptor set. CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems, 2608, 1027–1039.

25. Gorokhovatsky, V. (2014). Structural analysis and intellectual data processing in computer vision. Kharkiv: SMIT.

26. Gorokhovatskyi, V., Putyatin, Y., & Stolyarov, V. (2017). Research of effectiveness of structural image classification methods using cluster data model. Radio Electronics, Computer Science, Control, 3(42), 78–85.

27. Ho, J., Salimans, T., Gritsenko, A., Fleet, D. J., Norouzi, M., & Sukthankar, R. (2022). Video diffusion models. Advances in Neural Information Processing Systems, 35, 8633–8646.

28. Singer, U., Polyak, A., Hayes, T., Yin, X., An, J., Zhang, S., Hu, Q., Yang, H., Ashual, O., Gafni, O., Parikh, D., Gupta, S., & Taigman, Y. (2023). Make-A-Video: Text-to-video generation without text-video data. International Conference on Learning Representations. URL: https://openreview.net/forum?id=nJfylDvgzlq (дата звернення 24.05.2026).

29. Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A., Kingma, D. P., Poole, B., Norouzi, M., Fleet, D. J., & Salimans, T. (2022). Imagen Video: High definition video generation with diffusion models. arXiv. URL: https://arxiv.org/abs/2210.02303 (дата звернення 28.05.2026).

30. Veo. Google DeepMind. 2026. URL: https://deepmind.google/models/veo/ (дата звернення 10.06.2026).

31. Guo, Y., Yang, C., Rao, A., Wang, Y., Qiao, Y., Lin, D., & Dai, B. (2024). AnimateDiff: Animate your personalized text-to-image diffusion models without specific tuning. International Conference on Learning Representations. URL: https://openreview.net/forum?id=Fx2SbBgcte (дата звернення 29.05.2026).

32. Blattmann, A., Dockhorn, T., Kulal, S., Mendelevitch, D., Kilian, M., Lorenz, D., Levi, Y., English, Z., Voleti, V., Letts, A., Jampani, V., & Rombach, R. (2023). Stable video diffusion: Scaling latent video diffusion models to large datasets. arXiv. URL: https://arxiv.org/abs/2311.15127 (дата звернення 10.06.2026).

33. Tvoroshenko, I., Pomazan, V., Gorokhovatskyi, V., & Kobylin, O. (2023). Application of video data classification models using convolutional neural networks. International Journal of Academic and Applied Research, 7(11), 134–145.

34. SQLite documentation. URL: https://www.sqlite.org/docs.html (дата звернення 13.06.2026).

35. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) (RFC 7519). 2015. URL: https://www.rfc-editor.org/info/rfc7519 (дата звернення 13.06.2026).

36. React: A JavaScript library for building user interfaces. URL: https://react.dev/ (дата звернення 13.06.2026).

37. Voron, F. (2024). Full stack FastAPI, React, and MongoDB. Packt Publishing.

38. Lubanovic, B. (2023). FastAPI: Building high-performance, asynchronous web APIs. O'Reilly Media.39. Rombach R., Blattmann A., Lorenz D., Esser P., Ommer B. High-resolution image synthesis with latent diffusion models. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022. P. 10684–10695.

39. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 10684–10695)

40. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.

41. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning transferable visual models from natural language supervision. Proceedings of the 38th International Conference on Machine Learning, 139, 8748–8763.

42. Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating image classification based on a model for estimating descriptor-to-class distance. International Journal of Computing, 22(4), 485–492.

43. Гороховатський, В. О., Гадецька, С. В., & Стяглик, Н. І. (2019). Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. Радіоелектроніка, інформатика, управління, 2, 100–107.

44. Inference Providers. Hugging Face. URL: https://huggingface.co/docs/inference-providers/en/index (дата звернення 03.06.2026).

45. InferenceClient. Hugging Face. URL: https://huggingface.co/docs/huggingface_hub/en/package_reference/inference_client (дата звернення 03.06.2026).

46. Pricing and billing. Hugging Face. URL: https://huggingface.co/docs/inference-providers/en/pricing (дата звернення 03.06.2026).

47. Гороховатський В.О., Творошенко І.С. Оцінювання значущості ознак для підвищення продуктивності структурних методів класифікації зображень. Проблеми інформатики та моделювання (ПІМ-2025): тези XXV міжнародної науково-технічної конференції. Харків: НТУ «ХПІ», 2025. С. 38-43.

48. Gorokhovatskyi V., Tvoroshenko I. Identification of visual objects by the search request. Proceedings of the International Scientific Symposium «Intelligent Solutions-S»: Computational Intelligence. Kyiv-Uzhorod, 2023. P. 25-27.