Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ TELEGRAM ДЛЯ АВТОМАТИЗОВАНОЇ ОБРОБКИ ТА УНІКАЛІЗАЦІЇ МЕДІАКОНТЕНТУ
DEVELOPMENT OF A TELEGRAM WEB APPLICATION FOR AUTOMATED MEDIA PROCESSING AND CONTENT UNIQUENESS GENERATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-113сп
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

КАРПУС Д.М.
(ПРІЗВИЩЕ та ініціали)
Керівник ДУБРОВІН В.А.
(ПРІЗВИЩЕ та ініціали)
Рецензент ГОЛУБ Т.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КАРПУСА Дениса Максимовича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розроблення вебзастосунку Telegram для автоматизованної обробки та унікалізації медіаконтенту. Development of a Telegram Web Application for Automated Media Processing and Content Uniqueness Generation

керівник проєкту (роботи) к.т.н., професор, ДУБРОВІН Валерій Іванович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ДУБРОВІН В.І., професор		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Денис КАРПУС
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ _____ Валерій ДУБРОВІН
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 94 с., 8 табл., 39 рис., 3 дод., 8 джерел.

TELEGRAM WEB APP, ОБРОБКА МЕДІАКОНТЕНТУ, УНІКАЛІЗАЦІЯ ЗОБРАЖЕНЬ, УНІКАЛІЗАЦІЯ ВІДЕО, FLASK, SQLITE, FFMPEG, PILLOW, TELEGRAM BOT API.

Об’єкт роботи – процес розроблення програмного забезпечення для автоматизованої обробки та унікалізації медіаконтенту в середовищі Telegram.

Предмет роботи – програмні засоби створення Telegram Web App для завантаження, модифікації, унікалізації та подальшого надсилання фото- та відеоконтенту користувачам.

Мета роботи – скорочення часу підготовки унікалізованого медіаконтенту шляхом розроблення Telegram Web App, який забезпечує автоматизовану обробку зображень і відеофайлів та інтеграцію з Telegram Bot API.

Використані матеріали та методи: мова програмування Python, вебфреймворк Flask, система керування базами даних SQLite, бібліотека Pillow для обробки зображень, FFmpeg для обробки відео, Telegram Web Apps SDK, Telegram Bot API, JavaScript, HTML та CSS.

Результати. Результатом роботи є створення Telegram Web App «Variify», що забезпечує завантаження фото та відео, їх автоматизовану унікалізацію за допомогою зміни колірних характеристик, масштабування, випадкового обрізання, накладання водяних знаків, очищення метаданих та подальше надсилання результату користувачу через Telegram-бота.

Висновки. Розроблене програмне забезпечення дозволяє автоматизувати процес підготовки медіаконтенту, зменшити витрати часу на

його ручне редагування та забезпечити інтеграцію всіх етапів обробки в межах екосистеми Telegram.

Галузь використання. Digital-маркетинг, affiliate marketing, контент-менеджмент, SMM, рекламні агентства та приватні користувачі, які працюють із медіаконтентом.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 94 pages, 8 tables, 39 figures, 3 appendixes, 8 sources.

TELEGRAM WEB APP, MEDIA CONTENT PROCESSING, IMAGE UNIQUENESS ENHANCEMENT, VIDEO UNIQUENESS ENHANCEMENT, FLASK, SQLITE, FFMPEG, PILLOW, TELEGRAM BOT API.

The object of study is the software development process for automated media content processing and uniqueness enhancement within the Telegram environment.

The subject of study is software tools and methods for creating a Telegram Web App that enables uploading, modifying, enhancing the uniqueness of, and delivering photo and video content to users.

The purpose of the work is to reduce the time required for preparing unique media content by developing a Telegram Web App that provides automated image and video processing and integration with the Telegram Bot API.

Materials, methods, and technical tools: Python programming language, Flask web framework, SQLite database management system, Pillow image processing library, FFmpeg video processing toolkit, Telegram Web Apps SDK, Telegram Bot API, JavaScript, HTML, and CSS.

Results. The result of the work is the development of the Telegram Web App “Variify”, which allows users to upload photos and videos and automatically enhance their uniqueness through color adjustment, resizing, random cropping, watermark application, metadata removal, and delivery of the processed result via a Telegram bot.

Conclusions. The developed software automates the media content preparation process, reduces the time required for manual editing, and integrates all stages of content processing within the Telegram ecosystem.

Field of application. Digital marketing, affiliate marketing, content management, social media marketing (SMM), advertising agencies, and private users working with media content.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

EXIF – Exchangeable Image File Format;

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Сучасний розвиток цифрових технологій призвів до значного збільшення обсягів створення та поширення мультимедійного контенту. Особливо активно фото- та відеоматеріали використовуються у сфері цифрового маркетингу, соціальних мереж, реклами та електронної комерції. У таких умовах виникає необхідність швидкої модифікації медіафайлів та створення їх унікальних версій для різних платформ і цільових аудиторій.

Однією з актуальних проблем є необхідність виконання великої кількості однотипних операцій над медіаконтентом. До таких операцій належать зміна кольорових характеристик, масштабування, обрізання, видалення метаданих, додавання водяних знаків та інші методи модифікації файлів. Виконання зазначених дій вручну потребує значних витрат часу та використання спеціалізованого програмного забезпечення.

Окремого значення набуває інтеграція інструментів обробки медіаконтенту з популярними месенджерами. Одним із найпоширеніших месенджерів є Telegram, який надає можливість створення Telegram Web Apps – вебзастосунків, що запускаються безпосередньо всередині клієнта Telegram [2]. Це дозволяє забезпечити швидкий доступ до функціоналу системи без встановлення додаткового програмного забезпечення.

Для розв'язання зазначеної проблеми у даній роботі пропонується розроблення Telegram Web App для автоматизованої обробки та унікалізації медіаконтенту. Застосунок дозволяє завантажувати фото та відео, виконувати набір операцій з модифікації контенту та отримувати готовий результат безпосередньо в чаті Telegram-бота [3].

Метою роботи є розроблення Telegram Web App для автоматизованої обробки та унікалізації медіаконтенту шляхом реалізації функцій редагування зображень і відео, видалення метаданих, додавання водяних знаків та інтеграції із сервісами Telegram [2], [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз систем обробки та унікалізації медіаконтенту

Сучасний розвиток цифрових технологій супроводжується стрімким збільшенням обсягів мультимедійного контенту, що створюється та поширюється через мережу Інтернет. Фото та відеоматеріали стали основними інструментами передачі інформації в соціальних мережах, рекламних кампаніях, інформаційних ресурсах та системах електронної комерції. В умовах високої конкуренції між контентом особливої актуальності набуває питання його адаптації та модифікації для різних платформ і каналів поширення.

Одним із напрямків роботи з цифровими медіаданими є унікалізація контенту. Під унікалізацією розуміється процес зміни характеристик файлу зі збереженням його візуального або інформаційного змісту. Основною метою такого процесу є створення нової версії медіафайлу, яка відрізняється від оригіналу на рівні структури даних, метаданих або окремих параметрів зображення чи відео.

Необхідність унікалізації виникає в багатьох сферах діяльності:

– цифровому маркетингу;
– affiliate marketing;
– контент-менеджменті;
– електронній комерції;
– SMM-просуванні;
– рекламній діяльності.

Особливо актуальним питання унікалізації є для спеціалістів цифрового маркетингу, які щоденно працюють із великою кількістю рекламних креативів. Під час запуску рекламних кампаній часто виникає потреба створення декількох варіантів одного й того самого медіафайлу. Навіть незначні зміни параметрів файлу можуть призвести до формування

нового цифрового відбитка, що дозволяє отримати технічно відмінний медіаоб'єкт.

На сьогодні існують різні підходи до модифікації цифрового контенту:

– зміна розмірів зображення;
– обрізання окремих областей;
– дзеркальне відображення;
– корекція кольорів;
– зміна яскравості та контрастності;
– зміна частоти кадрів відео;
– перекодування файлів;
– додавання водяних знаків;
– очищення метаданих.

Незважаючи на наявність великої кількості графічних редакторів та онлайн-сервісів, більшість із них орієнтована на ручне виконання операцій. Це призводить до значних часових витрат та знижує продуктивність роботи користувачів.

Ситуація ускладнюється тим, що для роботи із різними типами медіаконтенту часто необхідно використовувати декілька незалежних програмних продуктів. Наприклад, для редагування фотографій застосовується один редактор, для відео - інший, а для видалення метаданих - окремі утиліти. Такий підхід збільшує складність робочого процесу та негативно впливає на швидкість підготовки контенту.

Важливим фактором також є розвиток мобільних платформ та месенджерів. Значна частина користувачів виконує робочі завдання безпосередньо зі смартфонів. У зв'язку з цим зростає попит на легкі веборієнтовані рішення, які не потребують встановлення додаткового програмного забезпечення.

Однією з перспективних технологій у даному напрямку є Telegram Web Apps [2]. Ця технологія дозволяє створювати повноцінні вебзастосунки, що працюють безпосередньо всередині Telegram. Користувач отримує

можливість працювати із системою через звичайний чат-бот без необхідності встановлення окремих програм.

Перевагами Telegram Web Apps є [2]:

– доступність на будь-якій платформі;
– інтеграція з Telegram API;
– підтримка мобільних пристроїв;
– швидкий запуск застосунку;
– отримання даних користувача Telegram;
– спрощена взаємодія з ботами.

Саме тому використання Telegram Web App як платформи для автоматизованої обробки медіаконтенту є перспективним напрямком розвитку сучасних програмних систем (рис. 1.1).

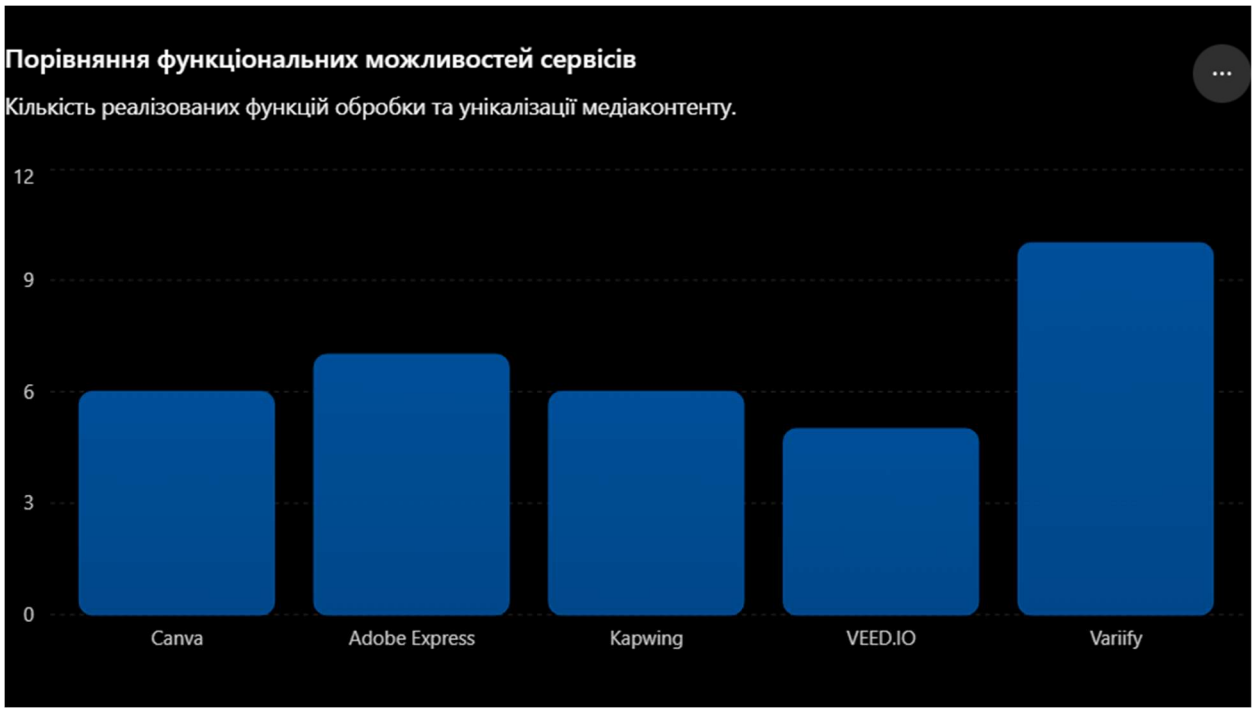


Рисунок 1.1 – Порівняння функціональних можливостей сучасних сервісів обробки медіаконтенту

Як видно з рисунка 1.1, існуючі програмні рішення забезпечують широкий набір засобів редагування мультимедійного контенту, проте не підтримують повноцінну інтеграцію з Telegram Web Apps та Telegram Bot API.

Розроблювана система Variify поєднує функції обробки фото і відео, видалення метаданих, накладання водяних знаків, ведення історії операцій та надсилання результатів через Telegram, що дозволяє підвищити рівень автоматизації процесу підготовки медіаконтенту.

1.2 Порівняльний аналіз існуючих аналогів

Сучасний ринок програмних засобів для обробки медіаконтенту характеризується наявністю великої кількості онлайн-сервісів, що дозволяють редагувати зображення, відео, створювати рекламні креативи, накладати текстові елементи, змінювати формат файлів та експортувати результат для подальшого використання. Найбільш поширеними серед таких рішень є Canva, Adobe Express, Kapwing та VEED.IO.

Зазначені сервіси мають розвинений функціонал і орієнтовані на широке коло користувачів. Водночас більшість із них не забезпечує повної автоматизації процесу унікалізації медіафайлів у межах Telegram. Для користувача це означає необхідність відкривати сторонній вебсервіс, завантажувати файл, виконувати редагування, експортувати результат і лише після цього вручну надсилати його в Telegram.

Першим аналогом, розглянутим у межах дослідження, є сервіс Canva. Це популярна онлайн-платформа для створення графічного контенту, рекламних матеріалів, презентацій та зображень для соціальних мереж. Canva має зручний інтерфейс, підтримує роботу з шаблонами, дозволяє змінювати розміри зображень, додавати текст, графічні елементи та водяні знаки. Проте система орієнтована насамперед на дизайнерське редагування, а не на автоматизовану технічну унікалізацію файлів (рис. 1.2).

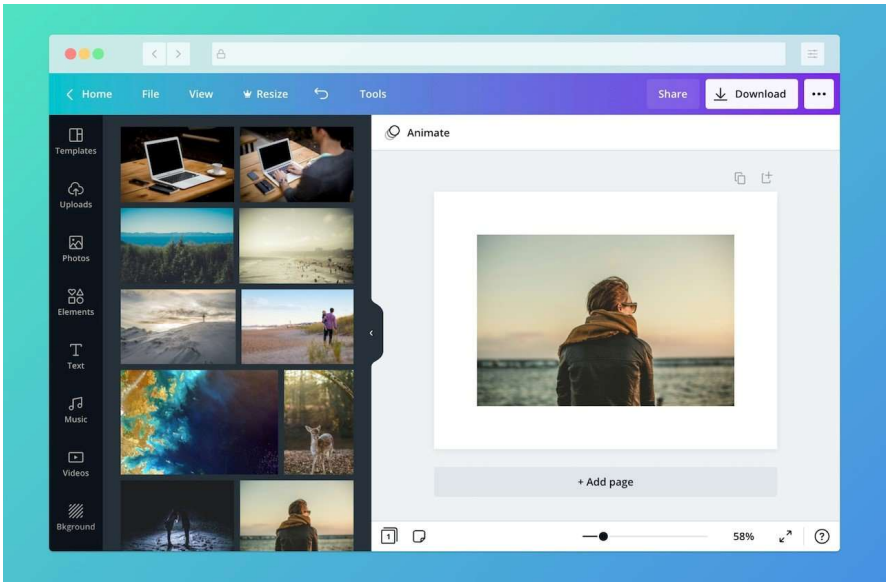


Рисунок 1.2 – Вебсервіс Canva

До основних переваг Canva можна віднести простоту використання, велику кількість готових шаблонів та можливість швидкого створення візуальних матеріалів. Недоліками є залежність від вебінтерфейсу, обмеження безкоштовного тарифу та відсутність інтеграції з Telegram Web Apps. Крім того, сервіс не надає користувачу окремого сценарію для автоматичного очищення метаданих і подальшого надсилання результату в Telegram-бот.

Другим аналогом є Adobe Express. Це веборієнтований інструмент компанії Adobe, призначений для швидкого створення та редагування графічного і відеоконтенту. Система має якісні засоби роботи із зображеннями, підтримує шаблони, базову обробку відео, зміну кольорів і додавання елементів оформлення. Adobe Express є зручним для підготовки рекламних матеріалів, але значна частина функціоналу доступна лише за умов використання облікового запису Adobe та відповідного тарифного плану (рис. 1.3).

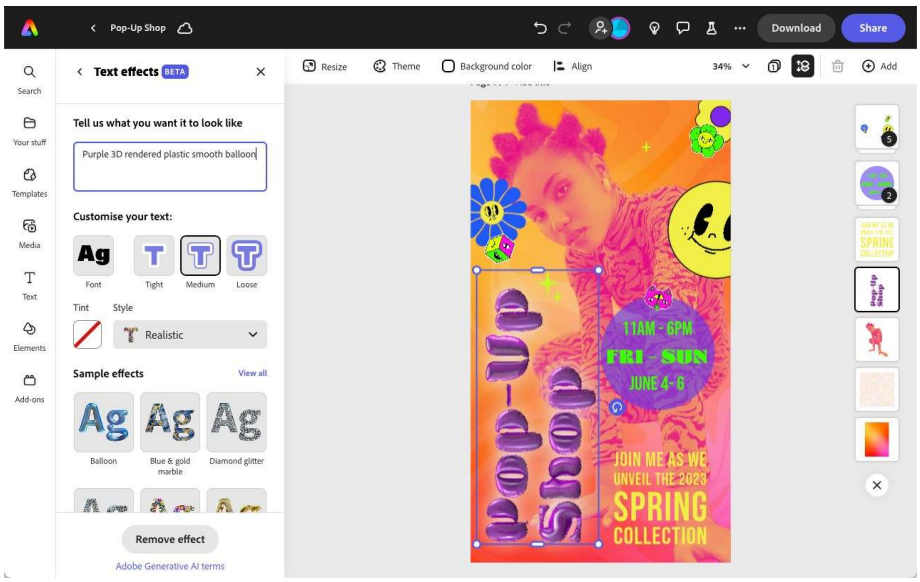


Рисунок 1.3 – Вебплатформа Adobe Express

Перевагою Adobe Express є належність до екосистеми Adobe, що забезпечує якісну роботу з мультимедійними файлами. Водночас недоліком є відсутність вбудованої інтеграції з Telegram Bot API, неможливість запуску як Telegram Web App та недостатня орієнтація на масову автоматизовану унікалізацію файлів.

Третім розглянутим аналогом є Kapwing. Це онлайн-сервіс для редагування відео, зображень, GIF-анімацій та створення контенту для соціальних мереж. Kapwing підтримує обрізання відео, додавання субтитрів, зміну формату, накладання тексту, зміну розмірів і командну роботу. Система є достатньо функціональною для створення мультимедійного контенту, однак процес роботи все одно залишається ручним (рис. 1.4).

Рисунок 1.4 – Онлайн-сервіс Kapwing

Основною перевагою Kapwing є широкий набір функцій для відеоредагування. До недоліків можна віднести обмеження безкоштовного тарифу, необхідність ручного експорту файлів та відсутність інтеграції з Telegram. Для використання результату в месенджері користувач має додатково завантажити файл і самостійно надіслати його в потрібний чат.

Четвертим аналогом є VEED.IO. Цей сервіс спеціалізується переважно на роботі з відеоконтентом. Він дозволяє виконувати базове редагування, обрізання, додавання тексту, субтитрів, фільтрів та експорт відео у потрібному форматі. VEED.IO зручний для створення відеороликів для соціальних мереж, проте менш орієнтований на технічну унікалізацію медіафайлів (рис. 1.5).

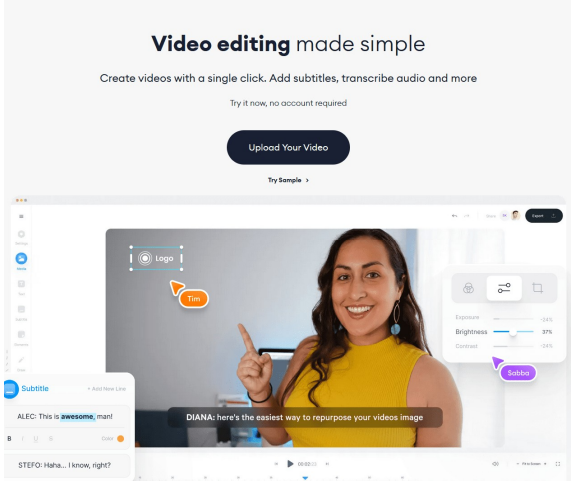


Рисунок 1.5 – Вебсервіс VEED.IO

Перевагами VEED.IO є зручний відеоредактор, підтримка онлайн-обробки та наявність сучасного інтерфейсу. Основними недоліками є обмеження безкоштовного використання, відсутність Telegram-інтеграції та неможливість ведення локальної історії обробки в межах Telegram-бота.

На відміну від розглянутих аналогів, розроблювана система Variify орієнтована не на універсальне дизайнерське редагування, а саме на швидку автоматизовану обробку та унікалізацію медіаконтенту безпосередньо в Telegram. Система реалізована як Telegram Web App і дозволяє користувачу завантажити фото або відео, вибрати параметри обробки, переглянути результат і надіслати його назад у чат Telegram-бота.

Згідно з реалізованим проєктом, Variify підтримує обробку фото та відео, дзеркальне відображення, чорно-білий режим, зміну яскравості, контрастності, насиченості, RGB-балансу, випадковий crop, зміну розміру, додавання watermark, очищення EXIF/metadata, історію обробок та надсилання результату через Telegram Bot API.

Порівняння функціональних можливостей існуючих систем наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих рішень для обробки медіаконтенту

Характеристика	Canva	Kapwing	Telegram Bots	Variify
Обробка зображень	+	+	+/-	+
Обробка відео	-	+	+/-	+
Додавання watermark	+	+	-	+
Очищення метаданих	-	-	-	+
Очищення EXIF metadata	-	-	+	+
Інтеграція з Telegram	+	+	+	+

Під час аналізу встановлено, що більшість існуючих сервісів забезпечує достатній рівень дизайнерського редагування, однак не вирішує задачу швидкої унікалізації контенту в межах Telegram. Для користувачів, які працюють із рекламними креативами, це створює додаткові незручності, оскільки процес складається з великої кількості ручних дій.

Розроблювана система Variify усуває цей недолік за рахунок поєднання Telegram Web App, Flask backend, SQLite, Pillow, FFmpeg та Telegram Bot API в єдиному робочому процесі [1]- [6]. Архітектура системи передбачає запуск застосунку всередині Telegram, завантаження файлу, обробку на сервері, збереження історії та надсилання результату користувачу через Telegram-бота.

Таким чином, основна відмінність Variify від існуючих аналогів полягає в орієнтації на автоматизовану технічну обробку медіафайлів, а не лише на візуальне редагування. Система дозволяє скоротити кількість дій користувача та забезпечити повний цикл роботи з медіаконтентом у межах одного середовища.

Для наочного представлення результатів порівняльного аналізу на рисунку 1.6 наведено узагальнену діаграму функціонального покриття розглянутих сервісів.

Рисунок 1.6 – Порівняння функціонального покриття сервісів обробки медіаконтенту

З діаграми видно, що розроблюване програмне забезпечення має ширше функціональне покриття саме в контексті теми дипломної роботи. Це пояснюється тим, що Variify реалізує не тільки базове редагування медіафайлів, а й спеціалізовані функції унікалізації, очищення метаданих, інтеграції з Telegram та збереження історії операцій.

Отже, проведений порівняльний аналіз підтверджує доцільність розроблення Telegram Web App для автоматизованої обробки та унікалізації медіаконтенту. Існуючі сервіси залишаються ефективними для загального редагування, однак не забезпечують повноцінного сценарію роботи, який потрібен користувачам Telegram для швидкої підготовки та отримання унікалізованих медіафайлів.

1.3 Висновки до розділу

У першому розділі було проведено аналіз предметної області, пов’язаної з автоматизованою обробкою та унікалізацією медіаконтенту. Дослідження показало, що в умовах активного розвитку цифрового маркетингу, соціальних мереж та рекламних платформ значно зростає потреба у швидкому створенні великої кількості варіантів фото- та відеоматеріалів. У більшості випадків підготовка такого контенту виконується вручну із використанням декількох програмних засобів, що збільшує витрати часу та ускладнює робочий процес.

У процесі аналізу було розглянуто сучасні програмні рішення для роботи з медіаконтентом, серед яких Canva, Adobe Express, Kapwing та VEED.IO. Встановлено, що зазначені системи забезпечують широкий набір інструментів для редагування зображень та відео, однак переважно орієнтовані на ручну роботу користувача. Крім того, більшість із них не підтримує автоматизовану унікалізацію файлів, очищення метаданих, інтеграцію з Telegram Web Apps та надсилання результатів через Telegram Bot API.

Проведений порівняльний аналіз аналогів показав, що існуючі сервіси мають певні функціональні обмеження щодо роботи з Telegram та автоматизації процесів підготовки медіаконтенту. Водночас встановлено, що поєднання можливостей Telegram Web App, серверної обробки файлів та інтеграції з Telegram Bot API дозволяє реалізувати більш зручний і ефективний підхід до роботи з мультимедійними даними.

На основі результатів дослідження сформовано основні вимоги до розроблюваної системи. Майбутній програмний продукт повинен забезпечувати завантаження фото та відеофайлів, виконання операцій унікалізації, зміну параметрів зображень і відео, додавання водяних знаків, очищення метаданих, збереження історії обробки та надсилання результатів користувачу через Telegram.

Таким чином, проведений аналіз підтвердив актуальність розроблення Telegram Web App для автоматизованої обробки та унікалізації медіаконтенту. Розробка власного програмного рішення дозволить усунути недоліки існуючих систем, скоротити кількість дій користувача під час підготовки контенту та забезпечити виконання всіх необхідних операцій у межах єдиного середовища Telegram.

Результати першого розділу стали основою для подальшого проєктування архітектури програмного забезпечення, вибору технологій розробки, формування структури бази даних та побудови сценаріїв взаємодії користувача із системою, які будуть розглянуті в наступному розділі.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

Після проведення аналізу предметної області та дослідження існуючих аналогів було визначено основні вимоги до програмного забезпечення для автоматизованої обробки та унікалізації медіаконтенту. Для реалізації поставлених завдань необхідно спроєктувати архітектуру системи, яка забезпечуватиме зручну взаємодію користувача із застосунком, обробку мультимедійних файлів, збереження інформації про результати роботи та інтеграцію з Telegram.

Основною особливістю розроблюваного програмного забезпечення є використання технології Telegram Web App, що дозволяє запускати застосунок безпосередньо всередині Telegram. Такий підхід дає можливість забезпечити єдине середовище роботи користувача без необхідності встановлення додаткових програм або переходу на сторонні вебресурси.

Архітектура системи будується за клієнт-серверним принципом. Клієнтська частина відповідає за взаємодію з користувачем, завантаження медіафайлів та відображення результатів обробки. Серверна частина забезпечує прийом запитів, виконання операцій над файлами, взаємодію з базою даних та надсилання результатів через Telegram Bot API [3]. Для обробки зображень використовуються засоби бібліотеки Pillow [4], а для роботи з відеофайлами – програмний пакет FFmpeg [5]. Інформація про користувачів, завантажені файли та історію обробок зберігається в базі даних SQLite [6].

У даному розділі буде виконано обґрунтування вибору засобів розробки, розглянуто архітектурні підходи до побудови системи, визначено основних акторів та сценарії взаємодії, а також спроєктовано структуру бази даних для зберігання інформації про роботу застосунку.

2.1 Вибір засобів проектування

Для реалізації Telegram Web App «Variify» необхідно було обрати програмні засоби, які забезпечують швидку розробку, простоту розгортання, підтримку роботи з мультимедійними файлами та можливість інтеграції з Telegram API.

До основних критеріїв вибору належали:

– підтримка веброзробки;
– можливість роботи з мультимедійними файлами;
– простота інтеграції з Telegram;
– кросплатформеність;
– висока швидкість розробки;
– наявність готових бібліотек для обробки фото та відео;
– підтримка роботи з базами даних.

Відповідно до зазначених критеріїв було проведено аналіз мов програмування, вебфреймворків та допоміжних інструментів розробки.

2.1.1 Обґрунтування вибору мови програмування

Одним із ключових етапів проектування програмного забезпечення є вибір мови програмування. Для реалізації серверної частини розроблюваної системи було розглянуто декілька популярних мов програмування, що використовуються у сфері веброзробки.

До порівняння включено:

– Python;
– Java;
– C#;
– PHP.

Основні характеристики мов наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування для розробки вебзастосунку

Критерій	Python	Java	C#	PHP
Простота розробки	Висока	Середня	Середня	Висока
Кількість бібліотек	Висока	Висока	Висока	Середня
Підтримка веброзробки	Висока	Висока	Висока	Висока
Робота з медіафайлами	Висока	Середня	Середня	Низька
Швидкість розробки	Висока	Середня	Середня	Низька

Аналіз наведених характеристик показав, що найбільш доцільним вибором для реалізації серверної частини системи є мова програмування Python. Основними перевагами Python є простий синтаксис, велика кількість готових бібліотек, активна спільнота розробників та висока швидкість створення програмного забезпечення.

Додатковою перевагою є наявність бібліотеки Pillow для роботи із зображеннями [4] та можливість інтеграції з FFmpeg для обробки відеофайлів [5]. Саме ці засоби використовуються у реалізованому проєкті Variify для виконання операцій унікалізації медіаконтенту.

Таким чином, для розроблення серверної частини системи було обрано мову програмування Python, яка найкраще відповідає поставленим вимогам щодо функціональності, швидкості розробки та підтримки мультимедійних технологій.

2.1.2 Вибір та обґрунтування вебфреймворків

Для реалізації серверної частини (Backend) програмного забезпечення обрано вебфреймворк Flask [1]. Даний фреймворк є одним із найпоширеніших рішень для створення вебзастосунків мовою Python та належить до категорії мікрофреймворків. Використання Flask дозволяє створювати легкі вебсервіси з мінімальною кількістю залежностей та забезпечує високу швидкість розробки програмного забезпечення.

Вибір Flask обґрунтований такими факторами:

– простота архітектури – фреймворк не нав'язує жорсткої структури проєкту, що дозволяє організувати компоненти системи відповідно до потреб розробки;

– легкість інтеграції – Flask легко взаємодіє з бібліотеками для роботи із мультимедійними файлами, такими як Pillow та FFmpeg [4, 5], що є важливим для реалізації функцій обробки фото та відео;

– REST API – фреймворк дозволяє швидко реалізувати набір API-запитів для завантаження файлів, запуску процесів обробки та взаємодії із клієнтською частиною застосунку;

– масштабованість – архітектура Flask дозволяє поступово розширювати функціональність системи без суттєвих змін базової структури проєкту;

– підтримка Telegram Bot API [3] – використання Flask спрощує інтеграцію із сервісами Telegram та організацію взаємодії між Telegram Web App і Telegram-ботом.

У проєкті Flask використовується як центральний компонент серверної частини та забезпечує роботу всіх основних програмних модулів: завантаження файлів, обробку медіаконтенту, взаємодію з базою даних та надсилання результатів користувачу через Telegram Bot API.

Для розробки користувацького інтерфейсу (Frontend) використано стандартні вебтехнології HTML5, CSS3 та JavaScript. Такий підхід дозволяє

забезпечити повну сумісність із платформою Telegram Web Apps та підтримку різних мобільних і настільних пристроїв.

Основними причинами вибору даних технологій є:

– кросплатформеність – застосунок може працювати на будь-якому пристрої, який підтримує сучасний веббраузер;

– простота інтеграції з Telegram Web Apps SDK [2] – JavaScript забезпечує прямий доступ до функцій Telegram Web App та даних користувача;

– висока швидкість роботи інтерфейсу – відсутність додаткових фронтенд-фреймворків зменшує навантаження на клієнтську частину застосунку;

– адаптивність інтерфейсу – використання CSS дозволяє реалізувати mobile-first підхід, який є оптимальним для роботи всередині Telegram на смартфонах;

– простота підтримки – невелика кількість залежностей спрощує супровід та подальший розвиток програмного забезпечення.

Для взаємодії із Telegram використовується Telegram Web Apps SDK, який забезпечує отримання інформації про користувача, роботу всередині Telegram-клієнта та передачу даних між Web App і Telegram Bot API. Завдяки цьому користувач може працювати із застосунком без необхідності створення окремого облікового запису або проходження додаткової авторизації.

Таким чином, обрана сукупність технологій — Flask, HTML, CSS, JavaScript та Telegram Web Apps SDK – забезпечує необхідний рівень функціональності, продуктивності та зручності використання для реалізації Telegram Web App «Variify» та повністю відповідає вимогам розроблюваної системи.

2.1.3 Архітектурні підходи та взаємодії

Взаємодія між клієнтською та серверною частинами системи базується на використанні REST API для передачі даних між Telegram Web App та сервером Flask [1], [2]. Користувач взаємодіє із застосунком через інтерфейс Telegram Web App, після чого дані передаються на сервер для обробки медіаконтенту. Архітектура системи побудована за клієнт-серверним принципом та забезпечує розділення інтерфейсу користувача, бізнес-логіки та рівня зберігання даних.

Для обробки зображень використовується бібліотека Pillow [4], а для роботи з відеофайлами – програмний пакет FFmpeg [5]. Зберігання інформації про користувачів, файли та історію операцій реалізовано за допомогою SQLite [6]. Передача результатів користувачу здійснюється через Telegram Bot API [3]. Архітектурна взаємодія компонентів системи наведена на рисунку 2.1.

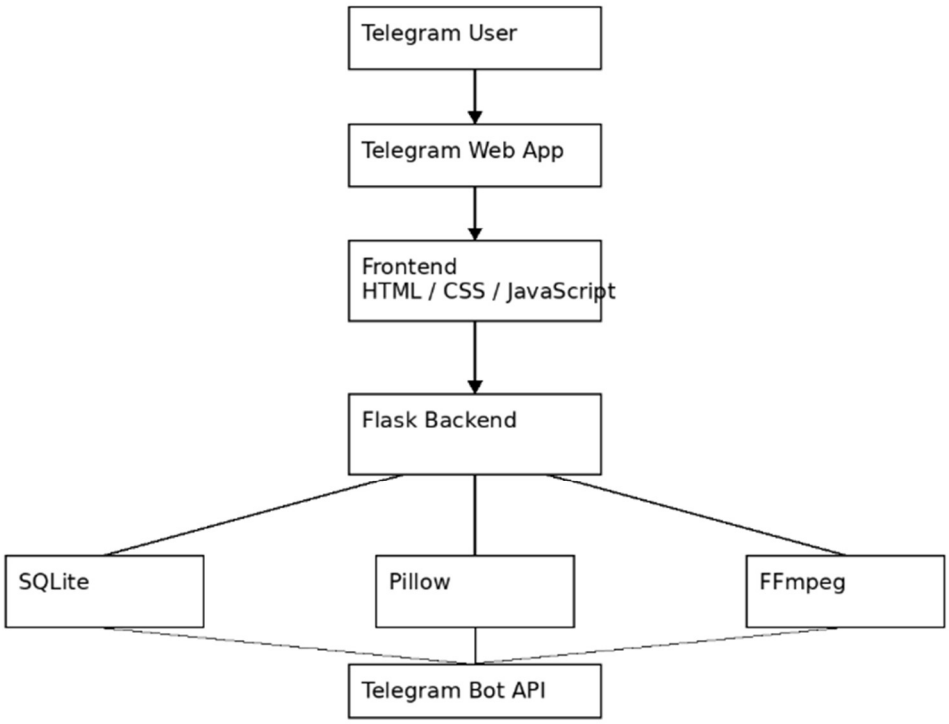


Рисунок 2.1 – UML-діаграма архітектури застосунку Variify

Розроблена архітектура забезпечує модульність системи та дозволяє незалежно розвивати окремі компоненти програмного забезпечення. Такий підхід спрощує підтримку проєкту, покращує масштабованість та забезпечує можливість подальшого розширення функціоналу системи.

2.1.4 Середовище розробки та допоміжні інструменти

Основним середовищем розробки програмного забезпечення обрано Visual Studio Code. Дане середовище є зручним для створення вебзастосунків, підтримує роботу з мовою Python, JavaScript, HTML та CSS, а також має велику кількість розширень для підсвічування синтаксису, форматування коду, роботи з Git та запуску серверних застосунків.

Для реалізації серверної частини використано інтерпретатор Python та вебфреймворк Flask. Залежності проєкту зберігаються у файлі requirements.txt, що спрощує встановлення необхідних бібліотек на іншому комп’ютері або сервері. У проєкті використовуються Flask, Pillow та Werkzeug.

Для обробки зображень застосовується бібліотека Pillow, яка забезпечує зміну кольорових параметрів, обрізання, масштабування, дзеркальне відображення та накладання watermark. Для обробки відеофайлів використовується FFmpeg, який дозволяє виконувати перекодування, зміну FPS, очищення metadata та застосування відеофільтрів.

Для зберігання даних обрано SQLite, оскільки дана система керування базами даних не потребує окремого серверного середовища та є зручною для MVP-рішень і дипломного проєкту. У базі даних зберігаються користувачі, завантажені медіафайли, задачі обробки, watermark-файли та налаштування.

Для інтеграції із Telegram використано Telegram Web Apps SDK та Telegram Bot API. Telegram Web Apps SDK дозволяє отримувати дані користувача та запускати інтерфейс застосунку безпосередньо всередині Telegram, а Telegram Bot API використовується для надсилання готового результату назад у чат користувача [7].

У процесі локального тестування може використовуватися ngrok, який створює публічне HTTPS-посилання на локальний Flask-сервер. Це необхідно для коректного запуску Telegram Web App, оскільки Telegram вимагає використання HTTPS-посилань для відкриття вебзастосунків.

Обраний набір інструментів забезпечує просте розгортання системи, зручність розробки, підтримку роботи з мультимедійними файлами та можливість інтеграції з Telegram. Надалі ця архітектура може бути розширена за рахунок переходу на PostgreSQL, додавання асинхронної черги задач та впровадження production-перевірки Telegram initData.

2.2 Сценарії використання програми

Для забезпечення ефективної взаємодії користувача із системою необхідно визначити основні сценарії використання програмного забезпечення. Аналіз сценаріїв дозволяє описати поведінку системи під час виконання основних функцій та сформувати вимоги до інтерфейсу, серверної частини й структури бази даних.

У розробленому Telegram Web App «Variify» усі дії користувача виконуються через інтерфейс застосунку, який відкривається безпосередньо в Telegram. Після запуску користувач отримує доступ до функцій завантаження медіафайлів, налаштування параметрів обробки, перегляду результатів та роботи з історією обробок.

Основою побудови сценаріїв є UML-модель варіантів використання (Use Case), яка описує взаємодію між користувачами та функціональними можливостями системи.

2.2.1 Класифікація акторів системи

Для роботи програмного забезпечення визначено декілька акторів, які взаємодіють із системою або беруть участь у виконанні окремих процесів.

Основні актори системи наведені в таблиці 2.2.

Таблиця 2.2 – Актори системи

Актор	Опис
Користувач	Завантажує файли, налаштовує параметри обробки та отримує результат
Telegram Web App	Забезпечує взаємодію між Telegram та вебзастосунком
Flask Backend	Обробляє запити та координує роботу системи
Telegram	Надсилає результати користувачу
SQLite	Зберігає інформацію про користувачів та файли

Головним актором системи є користувач. Саме він ініціює всі процеси, пов’язані із завантаженням, обробкою та отриманням медіаконтенту.

На рисунку 2.2 наведено UML-діаграму варіантів використання системи.

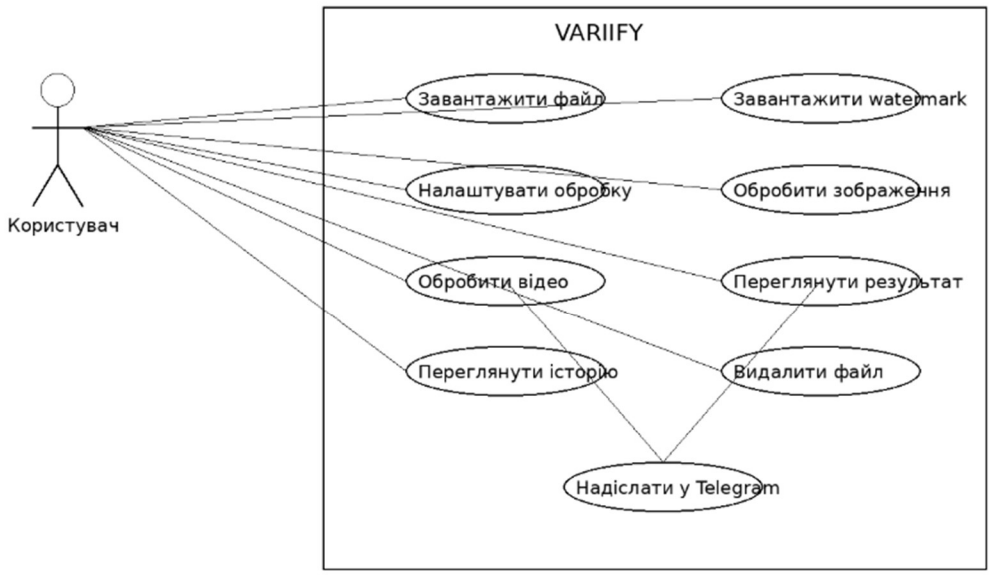


Рисунок 2.2 – UML Use Case діаграма системи Variify

Представлена діаграма демонструє основні можливості системи та сценарії взаємодії користувача із програмним забезпеченням.

2.2.2 Опис основних сценаріїв взаємодії

Для кожної групи акторів системи розроблено набір прецедентів, що охоплюють повний цикл роботи з медіаконтентом — від завантаження файлу до отримання результату через Telegram Bot API. Use Case діаграма системи наведена на рисунку 2.3.

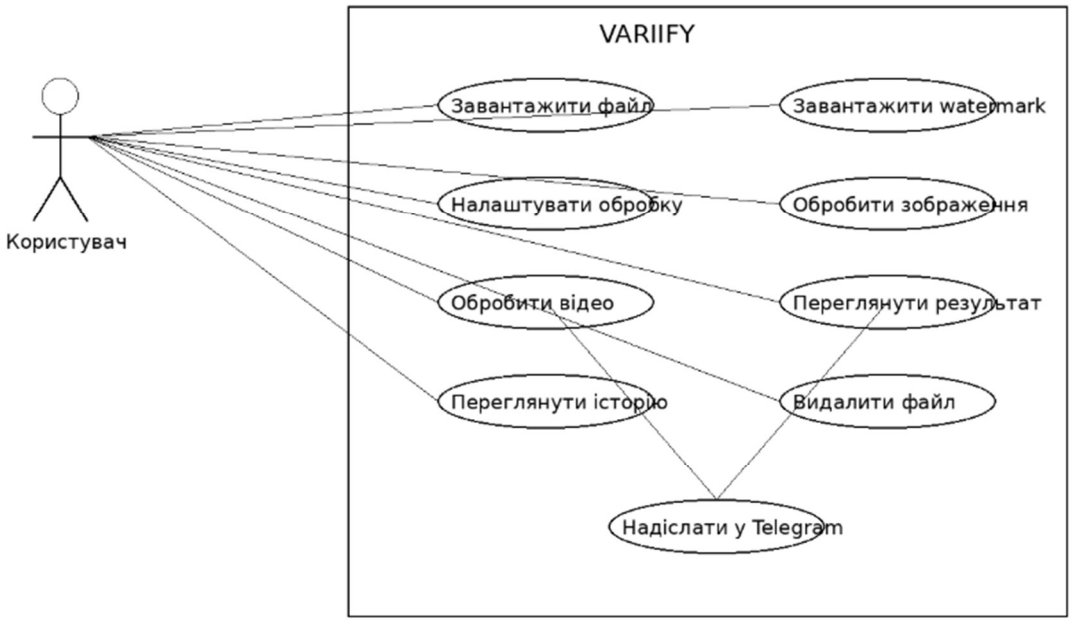


Рисунок 2.3 – Use Case діаграма системи Variify

Основний потік сценаріїв для користувача.

Користувач Telegram Web App має можливість:

– відкрити Telegram Web App через Telegram-бота;

– завантажити зображення для обробки;

– завантажити відеофайл для обробки;

– переглянути попередній перегляд файлу;

– завантажити watermark;

– налаштувати параметри обробки;

– змінити яскравість зображення;

– змінити контрастність;

– змінити насиченість;

– налаштувати RGB-баланс;

– застосувати дзеркальне відображення;

– виконати випадкове обрізання (Random Crop);

– змінити розмір файлу;

– очистити EXIF metadata;

– очистити metadata відеофайлу;

– обробити файл;

– переглянути результат обробки;

– переглянути історію виконаних операцій;

– повторно відкрити оброблений файл;

– видалити запис із історії;

– надіслати результат через Telegram Bot API.

Сценарій обробки зображення.

Сценарій обробки фотографії забезпечує автоматизовану модифікацію графічного файлу із застосуванням алгоритмів Pillow та подальшим збереженням результату в системі (рис. 2.4).

Основна послідовність виконання:

– користувач обирає фотографію;

– Frontend передає файл на Flask API;

– Flask зберігає файл;

– Pillow виконує обробку;

– SQLite зберігає інформацію про операцію;

– результат повертається користувачу.

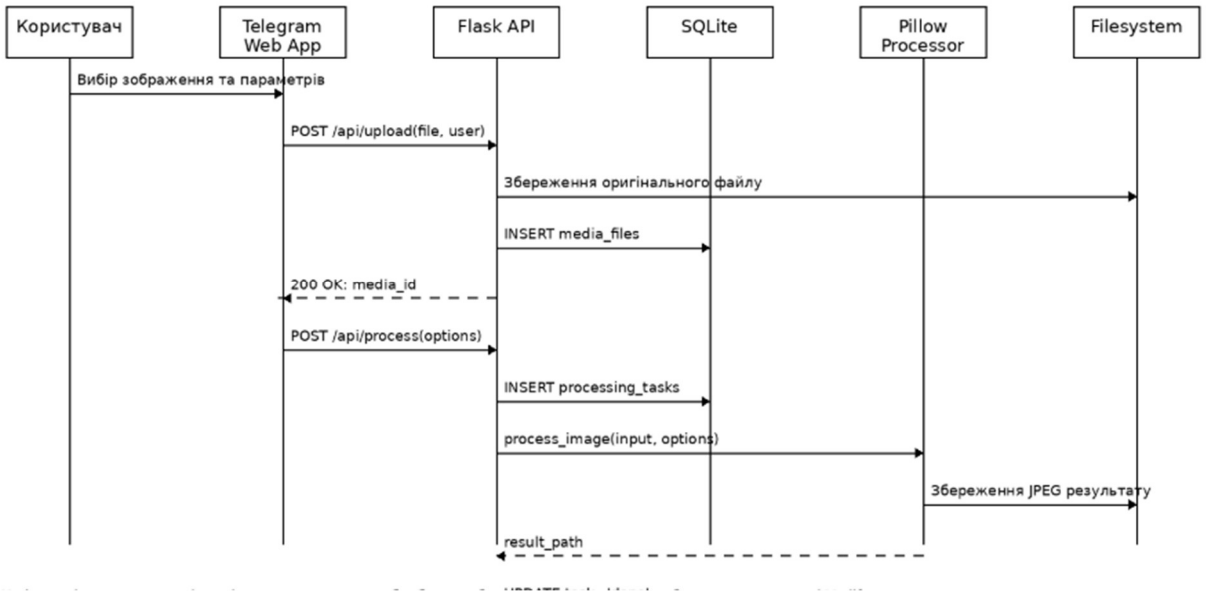


Рисунок 2.4 – UML діаграма послідовності обробки зображення

Сценарій обробки відеофайлу.

Сценарій обробки відео базується на використанні програмного пакета FFmpeg та забезпечує модифікацію технічних характеристик мультимедійного файлу (рис. 2.5).

Основна послідовність виконання:

– користувач завантажує відео;
– Flask приймає файл;
– формується FFmpeg-команда;
– виконується обробка;
– SQLite фіксує результат;
– готовий файл повертається користувачу.

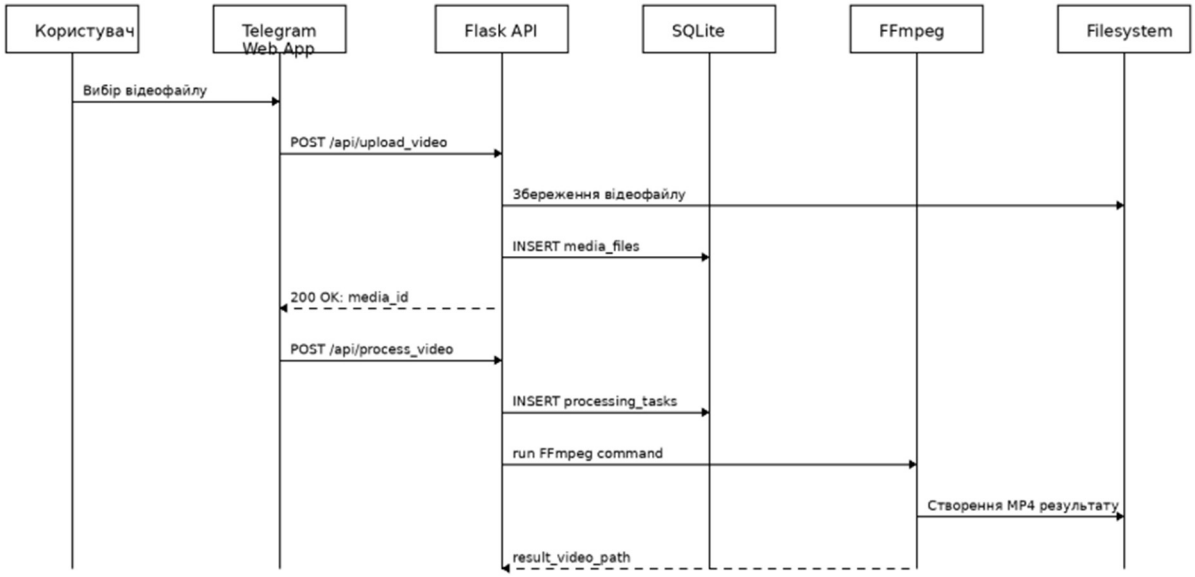


Рисунок 2.5 – UML діаграма послідовності обробки відеофайлу

Сценарій надсилання результату через Telegram.

Сценарій передбачає інтеграцію системи з Telegram Bot API та автоматичне надсилання обробленого файлу користувачу (рис. 2.6).

Основна послідовність:

– користувач натискає кнопку «Надіслати»;
– Frontend формує API-запит;

– Flask отримує ідентифікатор файлу;
– Telegram Bot API виконує надсилання;
– користувач отримує результат у чаті Telegram.

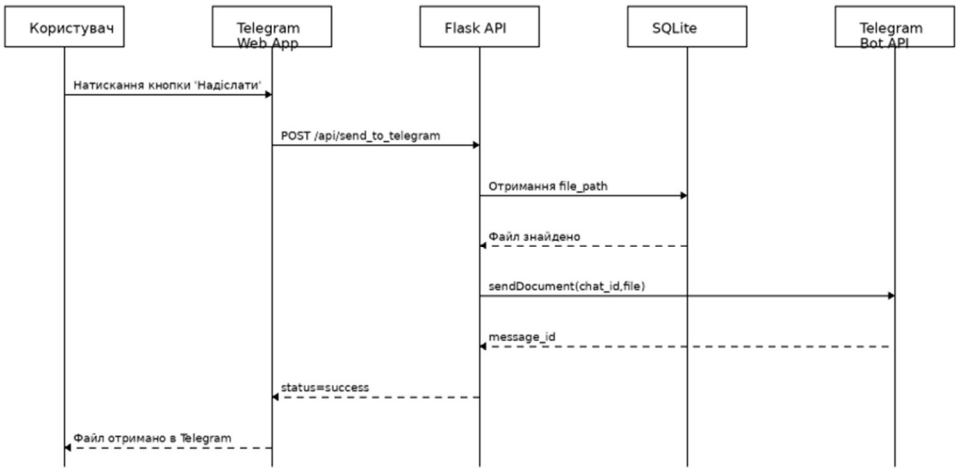


Рисунок 2.6 – UML діаграма послідовності надсилання результату через Telegram Bot API

2.2.3 Прототипування

На етапі проектування програмного забезпечення було виконано прототипування користувацького інтерфейсу Telegram Web App «Variify». Метою створення прототипу стало визначення структури майбутнього інтерфейсу, логіки взаємодії користувача із системою та розташування основних елементів керування до початку реалізації програмного забезпечення.

Прототип було розроблено відповідно до вимог мобільної платформи Telegram Web Apps та принципів адаптивного інтерфейсу. Особлива увага приділялася мінімізації кількості дій користувача під час виконання операцій обробки та унікалізації медіаконтенту.

На рисунку 2.7 представлено загальний прототип інтерфейсу системи Variify. Прототип складається з чотирьох основних екранів, які відображають ключові сценарії роботи користувача із застосунком.

Перший екран містить головні елементи взаємодії із системою. У верхній частині відображається назва застосунку та інформація про користувача Telegram. Нижче розташовані блоки вибору медіафайлу та watermark-файлу. Центральну частину займає область попереднього перегляду результатів обробки. У нижній частині інтерфейсу знаходяться вкладки налаштувань, кнопка запуску обробки та кнопка надсилання результату через Telegram Bot API.

Другий екран демонструє вкладку «Основне», яка містить базові параметри унікалізації медіаконтенту. До них належать дзеркальне відображення, переведення зображення в чорно-білий режим, випадкове обрізання зображення, накладання водяного знака та очищення EXIF-метаданих. Використання зазначених функцій дозволяє виконувати базову унікалізацію контенту без залучення сторонніх графічних редакторів.

Третій екран відображає вкладку «Колір», яка призначена для налаштування кольорових характеристик зображення. У даному розділі користувач може змінювати параметри яскравості, контрастності та насиченості, а також регулювати значення RGB-каналів. Такі зміни дозволяють створювати додаткові варіанти одного й того самого зображення та підвищувати рівень його унікальності.

Четвертий екран демонструє вкладку «Розмір», яка містить параметри зміни геометричних характеристик файлу. Користувач може задавати ширину та висоту зображення, налаштовувати FPS для відеофайлів та регулювати рівень прозорості водяного знака. Зазначені параметри використовуються під час формування фінального результату обробки.

Окремим функціональним елементом системи є блок історії обробок. Він забезпечує збереження інформації про раніше оброблені файли та дозволяє виконувати повторний перегляд результатів, надсилання файлів через Telegram або видалення записів із системи. Наявність історії підвищує зручність використання програмного забезпечення та спрощує повторну роботу з медіаконтентом.

Результати прототипування дозволили визначити оптимальну структуру інтерфейсу, перевірити зручність розташування елементів керування та сформувати остаточні вимоги до реалізації клієнтської частини Telegram Web App. Отриманий прототип став основою для подальшої розробки програмного забезпечення та реалізації функціональних можливостей системи Variify (рис. 2.7).

Рисунок 2.7 – Прототип користувацького інтерфейсу Telegram Web App «Variify»

2.3 Структура бази даних

Для забезпечення надійного зберігання даних, підтримки цілісності та можливості подальшого розширення системи було обрано реляційну систему керування базами даних SQLite. Вибір зумовлений простотою розгортання, автономністю роботи та достатніми можливостями для реалізації Telegram Web App, що обробляє медіафайли користувачів.

На рис. 2.8 представлена ER-модель БД, що складається з п’яти основних таблиць, які забезпечують функціонування облікових записів користувачів, збереження медіафайлів, виконання задач обробки, роботу з водяними знаками та зберігання налаштувань системи.

Проєктування реляційної структури базується на принципах нормалізації, де центральною сутністю є таблиця користувачів users, що слугує вузлом для ідентифікації користувача Telegram та прив’язки до нього всіх виконаних операцій. Взаємодія між користувачами та завантаженими файлами реалізована через зв’язок типу один-до-багатьох, що дозволяє одному користувачу мати необмежену кількість медіафайлів.

Завантажені файли зберігаються в таблиці media_files, де фіксуються назва оригінального файлу, службова назва збереженого файлу, тип медіа, шлях до оригіналу та шлях до результату обробки. Задачі обробки представлені таблицею processing_tasks, яка зберігає статус виконання, параметри обробки у форматі JSON, повідомлення про помилки та часові мітки створення й оновлення задачі.

Окремо реалізовано таблицю watermarks, яка призначена для збереження файлів водяних знаків користувачів. Таблиця settings зарезервована для майбутнього зберігання користувацьких налаштувань, таких як стандартні параметри обробки, пресети або індивідуальні конфігурації інтерфейсу.

Використання зовнішніх ключів забезпечує логічну цілісність даних між таблицями користувачів, медіафайлів, задач обробки, водяних знаків та

налаштувань. Така структура дозволяє зберігати історію операцій, повторно відкривати результати обробки та масштабувати систему шляхом додавання нових сутностей без зміни основної архітектури бази даних.

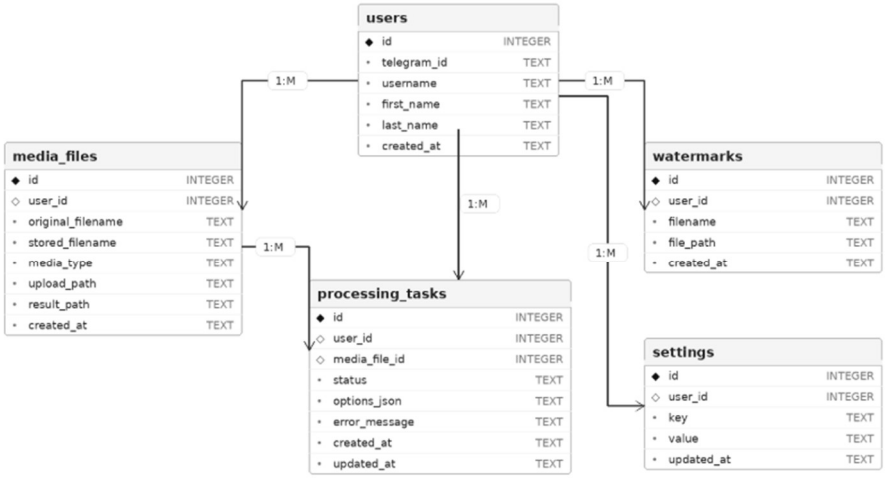


Рисунок 2.8 – ER-модель БД

Структура бази даних представлена набором взаємопов'язаних сутностей, опис яких наведено нижче.

Сутність «Users» визначає облікові записи користувачів Telegram та їх ідентифікаційні дані. Структура таблиці наведена в таблиці 2.3.

Таблиця 2.3 – Структура таблиці сутності «Users»

Назва стовпця	Тип даних	Призначення
id	INTEGER	Унікальний ідентифікатор користувача (Primary Key)
telegram_id	BIGINT	Унікальний Telegram ID користувача
username	TEXT	Ім'я користувача Telegram
created_at	DATETIME	DATETIME

Сутність «Media_Files» використовується для зберігання інформації про завантажені користувачами медіафайли (табл. 2.4).

Таблиця 2.4 – Структура таблиці сутності «Media_Files»

Назва стовпця	Тип даних	Призначення
id	INTEGER	Первинний ключ
user_id	INTEGER	Посилання на користувача
filename	TEXT	Назва файлу
file_type	TEXT	Тип файлу (image/video)
file_path	TEXT	Шлях до файлу
created_at	DATETIME	Дата завантаження

Сутність «Processing_Tasks» використовується для збереження інформації про процеси обробки медіаконтенту (табл. 2.5).

Таблиця 2.5 – Структура таблиці сутності «Processing_Tasks»

Назва стовпця	Тип даних	Призначення
id	INTEGER	Первинний ключ
media_id	INTEGER	Посилання на медіафайл
status	TEXT	Статус виконання
options	JSON	Параметри обробки
result_path	TEXT	Шлях до результату
created_at	DATETIME	Час створення задачі

Сутність «Watermarks» призначена для зберігання водяних знаків користувачів (табл. 2.6).

Таблиця 2.6 – Структура таблиці сутності «Watermarks»

Назва стовпця	Тип даних	Призначення
id	INTEGER	Первинний ключ
user_id	INTEGER	Посилання на користувача
file_path	TEXT	Шлях до watermark
opacity	REAL	Прозорість watermark

Сутність «Settings» використовується для зберігання параметрів роботи застосунку (2.7).

Таблиця 2.7 – Структура таблиці сутності «Settings»

Назва стовпця	Тип даних	Призначення
id	INTEGER	Первинний ключ
user_id	INTEGER	Посилання на користувача
key	TEXT	Назва параметра
value	TEXT	Назва параметра

Побудована структура бази даних забезпечує збереження всієї інформації, необхідної для функціонування Telegram Web App «Variify». Використання зв'язків між таблицями дозволяє підтримувати цілісність даних, а також забезпечує швидкий доступ до інформації про користувачів, медіафайли та результати їх обробки. Структура бази даних легко масштабується та може бути розширена шляхом додавання нових сутностей без зміни основної архітектури програмного забезпечення.

2.4 Висновки розділу 2

У ході виконання другого розділу було розв’язано комплекс архітектурних та проєктних задач, спрямованих на створення Telegram Web App для автоматизованої обробки та унікалізації медіаконтенту. Результати даного етапу роботи дозволяють сформулювати наступні підсумкові положення.

Проведено аналіз та підтверджено вибір засобів реалізації програмного забезпечення. Для серверної частини обрано мову програмування Python та вебфреймворк Flask, які забезпечують швидку розробку, просту інтеграцію із Telegram API та підтримку роботи з мультимедійними файлами. Для реалізації клієнтської частини використано HTML, CSS та JavaScript, що дозволяє створити адаптивний інтерфейс Telegram Web App та забезпечити сумісність із різними пристроями.

Розроблено клієнт-серверну архітектуру програмного забезпечення, яка забезпечує розділення інтерфейсу користувача, серверної логіки, механізмів обробки медіаконтенту та рівня зберігання даних. Побудовані UML-діаграми архітектури дозволили формалізувати взаємодію між компонентами системи та визначити основні інформаційні потоки.

Визначено основних акторів системи та розроблено детальні сценарії взаємодії користувача із програмою. Описано прецеденти, що охоплюють завантаження файлів, налаштування параметрів обробки, виконання операцій унікалізації, роботу з історією та надсилання результатів через Telegram Bot API. Для ключових бізнес-процесів побудовано UML-діаграми послідовності.

Розроблено прототип Telegram Web App «Variify», який дозволив визначити структуру екранів, розташування елементів керування та логіку взаємодії користувача із системою. Результати прототипування стали основою для реалізації клієнтської частини програмного забезпечення та подальшого тестування інтерфейсу.

Спроєктовано логічну схему бази даних, яка повністю адаптована для зберігання інформації про користувачів, медіафайли, задачі обробки, водяні знаки та налаштування системи. Структура БД підтримує реляційні зв’язки між сутностями та забезпечує цілісність даних. Використання зовнішніх ключів і нормалізованої структури дозволяє підтримувати масштабованість та подальший розвиток програмного забезпечення.

Таким чином, сформований набір архітектурних та проєктних рішень створює необхідний теоретичний і технічний фундамент для переходу до етапу безпосередньої реалізації програмних модулів Telegram Web App «Variify», розробки алгоритмів обробки медіаконтенту та проведення тестування системи, що буде розглянуто в наступному розділі.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Визначення порядку роботи програми

Логіка функціонування програмного забезпечення «Variify» базується на автоматизованій обробці та унікалізації медіаконтенту в середовищі Telegram. Порядок роботи системи охоплює повний цикл взаємодії користувача із Telegram Web App — від завантаження медіафайлу до отримання готового результату після виконання обробки. Загальна функціональна схема роботи програмного забезпечення наведена на рис. 3.1.

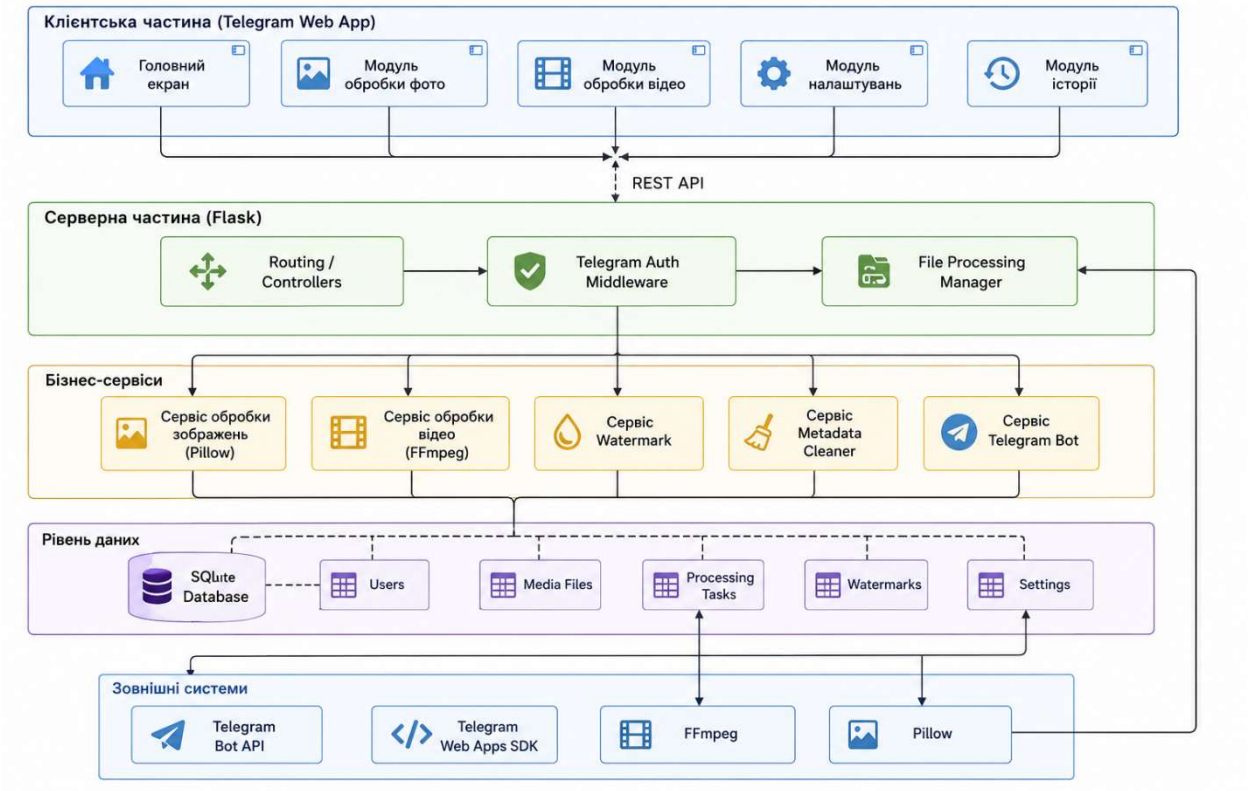


Рисунок 3.1 – Функціональна схема роботи Telegram Web App «Variify»

При відкритті Telegram Web App користувач потрапляє на головний екран застосунку, де відображаються основні елементи керування системою. Через інтерфейс користувач може вибрати фотографію або відеофайл для подальшої обробки, завантажити водяний знак (watermark), переглянути історію попередніх обробок та налаштувати параметри унікалізації контенту.

Взаємодія між клієнтською та серверною частинами здійснюється через REST API. Після вибору медіафайлу Telegram Web App формує HTTP-запит до серверної частини, яка реалізована на основі фреймворку Flask. Перед виконанням операцій обробки система перевіряє коректність вхідних даних, тип файлу та параметри, обрані користувачем.

Серверна частина містить модулі маршрутизації запитів, механізми автентифікації користувача через Telegram Web Apps та менеджер обробки файлів. Після отримання запиту система визначає тип медіаконтенту та передає його до відповідного сервісу обробки.

Для роботи із зображеннями використовується бібліотека Pillow. За її допомогою виконуються операції зміни яскравості, контрастності та насиченості, коригування кольорових каналів RGB, випадкове обрізання зображення, дзеркальне відображення, очищення EXIF-метаданих та накладання водяних знаків. Усі перетворення виконуються автоматично відповідно до параметрів, встановлених користувачем.

Обробка відеофайлів здійснюється за допомогою програмного пакета FFmpeg. Даний інструмент забезпечує перекодування відео, зміну частоти кадрів (FPS), масштабування, очищення службових метаданих та виконання інших операцій, необхідних для підвищення унікальності контенту.

Результати роботи сервісів обробки передаються до рівня зберігання даних. Для збереження інформації використовується база даних SQLite, у якій зберігаються відомості про користувачів, медіафайли, задачі обробки, водяні знаки та параметри системи. Використання бази даних дозволяє реалізувати історію обробок та забезпечує можливість повторного доступу до раніше створених результатів.

Після завершення процесу обробки користувач отримує попередній перегляд результату безпосередньо у Telegram Web App. У разі необхідності він може повторно змінити параметри та виконати нову обробку файлу.

Для швидкого отримання готового результату реалізовано інтеграцію із Telegram Bot API. Після натискання кнопки «Надіслати в бот» серверна

частина автоматично формує запит до Telegram Bot API та надсилає оброблений файл у чат користувача. Такий підхід усуває необхідність ручного завантаження файлів через браузер та забезпечує зручний спосіб отримання результатів роботи системи.

Таким чином, реалізований порядок роботи програмного забезпечення забезпечує повний цикл автоматизованої обробки медіаконтенту, починаючи від завантаження файлу та налаштування параметрів і завершуючи отриманням готового результату через Telegram. Побудована функціональна схема відображає взаємодію всіх основних компонентів системи та демонструє логіку роботи Telegram Web App «Variify».

3.2 Результати реалізації програмних одиниць застосунку

Реалізація програмного забезпечення «Variify» виконана за модульним принципом. Структура проєкту включає серверну частину на основі Flask, клієнтський Telegram Web App інтерфейс, модуль роботи з базою даних SQLite та модуль обробки медіаконтенту. Архітектура застосунку побудована таким чином, щоб логіка роботи з користувачем, збереження даних та обробка файлів були відокремлені один від одного, що спрощує підтримку та подальший розвиток системи.

Центральним компонентом системи є файл app.py, який реалізує серверну частину застосунку. У ньому створюється екземпляр Flask-застосунку, налаштовуються маршрути API, обробники помилок та взаємодія з іншими модулями системи. Сервер забезпечує прийом медіафайлів, запуск процесів обробки, формування історії операцій та інтеграцію з Telegram Bot API.

Лістинг 3.1 – Ініціалізація Flask-застосунку.

```
app = Flask(__name__, static_folder="static", static_url_path="/static")
app.config["MAX_CONTENT_LENGTH"] = 250 * 1024 * 1024
```

```
@app.before_request
def ensure_database():
    init_db()
```

Наведений фрагмент коду демонструє створення серверного застосунку та автоматичну ініціалізацію бази даних перед обробкою запитів користувачів.

Для зберігання інформації використовується модуль database.py, який реалізує створення таблиць користувачів, медіафайлів, задач обробки, водяних знаків та налаштувань системи. База даних працює на основі SQLite та створюється автоматично під час першого запуску програми.

Лістинг 3.2 – Створення таблиці користувачів.

```
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    telegram_id TEXT UNIQUE NOT NULL,
    username TEXT,
    first_name TEXT,
    last_name TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
```

Таблиця призначена для збереження інформації про користувачів Telegram та використовується всіма іншими сутностями системи.

Обробка медіаконтенту реалізована в модулі media_processor.py. Для фотографій використовується бібліотека Pillow, а для відеофайлів — програмний пакет FFmpeg. Модуль автоматично визначає тип файлу та запускає відповідний алгоритм обробки.

Лістинг 3.3 – Визначення типу медіафайлу.

```
def detect_media_type(filename):
    suffix = Path(filename).suffix.lower()

    if suffix in IMAGE_EXTENSIONS:
        return "image"

    if suffix in VIDEO_EXTENSIONS:
        return "video"

    return "unknown"
```

Даний метод використовується під час завантаження файлів та визначає подальший сценарій їх обробки.

Лістинг 3.4 – Реалізація маршруту завантаження медіафайлу.

```
@app.post("/api/upload")
def upload_media():
    user = upsert_user(request.form)
    file = request.files.get("file")

    if not file:
        return jsonify({"error": "File is required"}), 400

    media_type = detect_media_type(file.filename)

    if media_type == "unknown":
        return jsonify({"error": "Unsupported file type"}), 400

    original = secure_filename(file.filename)
    stored = f"{uuid.uuid4().hex}_{original}"
```

```
    upload_path = UPLOAD_DIR / stored
    file.save(upload_path)
```

Фрагмент коду реалізує API-маршрут завантаження медіафайлів до системи. Після отримання HTTP POST-запиту сервер перевіряє наявність файлу, визначає його тип та зберігає у директорії uploads. Для уникнення конфліктів імен використовується генерація унікального ідентифікатора UUID, який додається до назви файлу. Після успішного завантаження інформація про файл записується до бази даних SQLite.

Лістинг 3.5 – Запуск процесу обробки медіаконтенту.

```
result_path = process_media(
    Path(media["upload_path"]),
    OUTPUT_DIR,
    media["media_type"],
    options,
    watermark_path,
)
```

Наведений фрагмент виконує запуск основного алгоритму обробки медіаконтенту. Функція process_media() автоматично визначає тип файлу та передає його до відповідного модуля обробки. Для фотографій використовується бібліотека Pillow, а для відеофайлів — пакет FFmpeg. Такий підхід дозволяє реалізувати єдину точку входу для обробки різних типів контенту.

Лістинг 3.6 – Алгоритм обробки зображення.

```
if options.get("mirror"):
    image = ImageOps.mirror(image)
```

```
if options.get("grayscale"):
    image = ImageOps.grayscale(image).convert("RGB")

image = ImageEnhance.Brightness(image).enhance(
    float(options.get("brightness", 1))
)

image = ImageEnhance.Contrast(image).enhance(
    float(options.get("contrast", 1))
)
```

Даний фрагмент реалізує базові механізми унікалізації фотографій. Користувач може застосовувати дзеркальне відображення, чорно-білий режим, а також змінювати яскравість і контрастність зображення. Навіть незначні зміни параметрів дозволяють сформувати новий медіафайл із відмінною піксельною структурою та зміненим цифровим відбитком.

Лістинг 3.7 – Надсилання результату через Telegram Bot API.

```
ok, error = send_telegram_document(
    bot_token=bot_token,
    chat_id=row["telegram_id"],
    file_path=result_path,
    caption="Готовий результат Variify",
)
```

Фрагмент коду забезпечує інтеграцію системи із Telegram Bot API. Після завершення обробки користувач може натиснути кнопку «Надіслати в бот», після чого сервер автоматично формує запит до API Telegram та передає готовий файл у чат користувача. Такий механізм спрощує отримання

результатів роботи системи без необхідності додаткового завантаження файлів через браузер.

3.3 Висновки до розділу 3

У ході виконання третього розділу було здійснено практичну реалізацію Telegram Web App «Variify» для автоматизованої обробки та унікалізації медіаконтенту. Результати розробки дозволяють сформулювати наступні висновки.

Програмний продукт реалізовано відповідно до функціональних вимог та архітектурних рішень, визначених на етапі проєктування. У процесі розробки створено повноцінний Telegram Web App, що забезпечує завантаження, обробку, збереження та надсилання медіафайлів користувачам через Telegram Bot API.

Використання клієнт-серверної архітектури на основі Flask дозволило забезпечити розділення інтерфейсу користувача, серверної логіки та механізмів обробки медіаконтенту. Такий підхід підвищує масштабованість системи, спрощує підтримку програмного коду та створює можливість подальшого розширення функціональних можливостей застосунку.

Успішно реалізовано механізми обробки зображень із використанням бібліотеки Pillow. Програмне забезпечення підтримує зміну яскравості, контрастності та насиченості, коригування кольорових каналів, випадкове обрізання зображень, дзеркальне відображення, очищення EXIF metadata та накладання водяних знаків. Зазначені функції дозволяють автоматизувати процес створення унікального медіаконтенту без використання сторонніх графічних редакторів.

Для роботи з відеофайлами реалізовано інтеграцію з програмним пакетом FFmpeg. Система підтримує зміну частоти кадрів, масштабування відео, накладання водяних знаків, очищення метаданих та виконання інших операцій, спрямованих на підвищення унікальності відеоконтенту.

Автоматичне формування команд FFmpeg забезпечує високу гнучкість налаштування параметрів обробки.

Підтверджено працездатність механізму збереження інформації у базі даних SQLite. Реалізована структура БД забезпечує надійне зберігання даних про користувачів, медіафайли, історію обробок, параметри налаштувань та водяні знаки. Використання реляційної моделі даних дозволяє підтримувати цілісність інформації та забезпечує швидкий доступ до результатів обробки.

Окрему увагу приділено інтеграції із Telegram Bot API. Реалізований механізм надсилання результатів обробки безпосередньо в чат користувача значно підвищує зручність використання системи та дозволяє отримувати готові файли без додаткових дій із боку користувача.

У результаті тестування підтверджено коректність роботи основних функціональних модулів програмного забезпечення. Усі сценарії взаємодії користувача із системою виконуються відповідно до поставлених вимог, а процеси обробки фото- та відеофайлів завершуються успішним формуванням результату.

Таким чином, завершення етапу програмної реалізації підтвердило працездатність Telegram Web App «Variify» та готовність програмного забезпечення до проведення комплексного тестування й оцінювання ефективності роботи системи, що буде розглянуто в наступному розділі.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програмний продукт «Variify» призначений для автоматизованої обробки та унікалізації медіаконтенту в середовищі Telegram. Основною метою системи є спрощення процесу модифікації фотографій та відеофайлів шляхом автоматичного застосування набору алгоритмів обробки без використання сторонніх графічних редакторів або програм для монтажу відео.

Застосунок дозволяє користувачам виконувати завантаження медіафайлів безпосередньо через Telegram Web App, налаштовувати параметри обробки, застосовувати водяні знаки, очищувати метадані та отримувати готовий результат через Telegram Bot API. Реалізований функціонал забезпечує створення унікальних версій фото- та відеоконтенту з мінімальними часовими витратами.

Програмне забезпечення може використовуватися контент-менеджерами, власниками Telegram-каналів, маркетологами та іншими користувачами, які регулярно працюють із цифровим медіаконтентом та потребують швидкої автоматизованої обробки файлів.

4.2 Умови виконання програми

Безперебійне функціонування Telegram Web App «Variify» залежить від коректної роботи серверної частини, бази даних SQLite та зовнішніх сервісів Telegram.

Для стабільної роботи програмного забезпечення рекомендовані такі мінімальні характеристики сервера:

– процесор: 2 ядра x86-64 із тактовою частотою від 2.0 ГГц;
– оперативна пам'ять: від 4 ГБ;
– вільний простір на накопичувачі: від 10 ГБ SSD;

– операційна система: Windows 10/11, Ubuntu 22.04 LTS або новіші версії.

Для коректного запуску застосунку необхідно встановити:

– Python версії 3.10 або вище;

– Flask;

– Pillow;

– SQLite3;

– FFmpeg;

– бібліотеку Requests;

– Telegram Bot Token.

Успішний запуск програмного забезпечення можливий лише за умови правильної конфігурації середовища виконання.

Для коректного запуску застосунку необхідно встановити:

– наявність активного Telegram Bot;

– наявність токена доступу Telegram Bot API;

– встановлений та доступний у системній змінній PATH пакет FFmpeg;

– створена база даних SQLite;

– налаштовані каталоги для збереження завантажених та оброблених файлів.

4.3 Виконання та тестування програми

Метою даного етапу є перевірка працездатності Telegram Web App «Variify», коректності взаємодії між клієнтською та серверною частинами, а також перевірка функціонування алгоритмів обробки медіаконтенту та інтеграції із Telegram Bot API.

Процес тестування виконується шляхом проходження повного сценарію роботи користувача - від запуску Telegram Web App до отримання готового результату в чаті Telegram.

На початковому етапі тестування здійснюється запуск застосунку в середовищі Telegram. Після відкриття Web App користувачу відображається головний інтерфейс системи, який містить блок вибору медіафайлу, блок завантаження водяного знака, область попереднього перегляду, вкладки налаштувань та панель історії обробок (рис. 4.1).

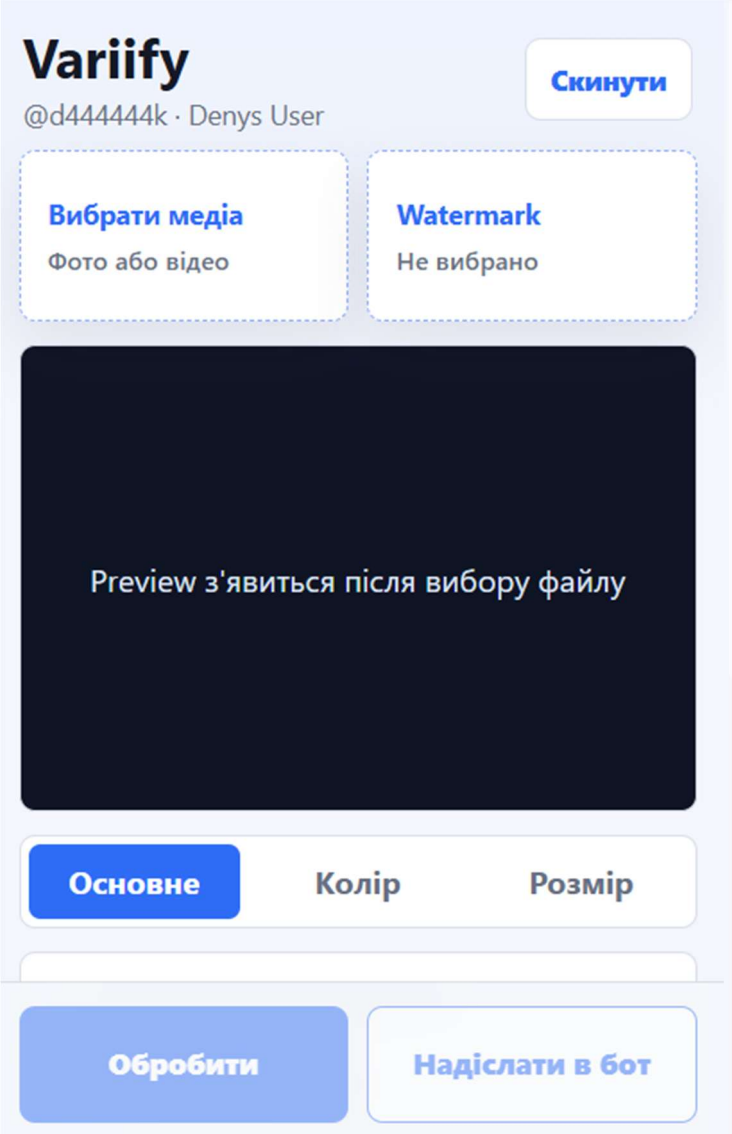


Рисунок 4.1 – Головний екран Telegram Web App «Variify»

Після запуску користувач може вибрати фотографію або відеофайл для подальшої обробки. Після успішного завантаження система відображає попередній перегляд вибраного медіаконтенту та активує елементи керування редактором (рис. 4.2).

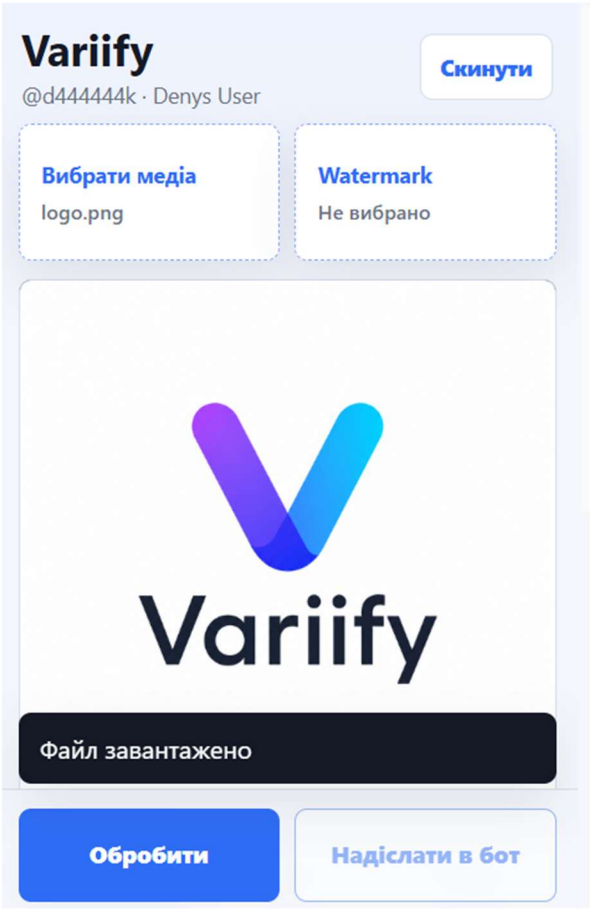


Рисунок 4.2 – Завантаження медіафайлу до системи

Для перевірки механізму обробки зображень було виконано завантаження графічного файлу та водяного знака. Після цього користувач задає параметри обробки та запускає алгоритм унікалізації (рис. 4.3).

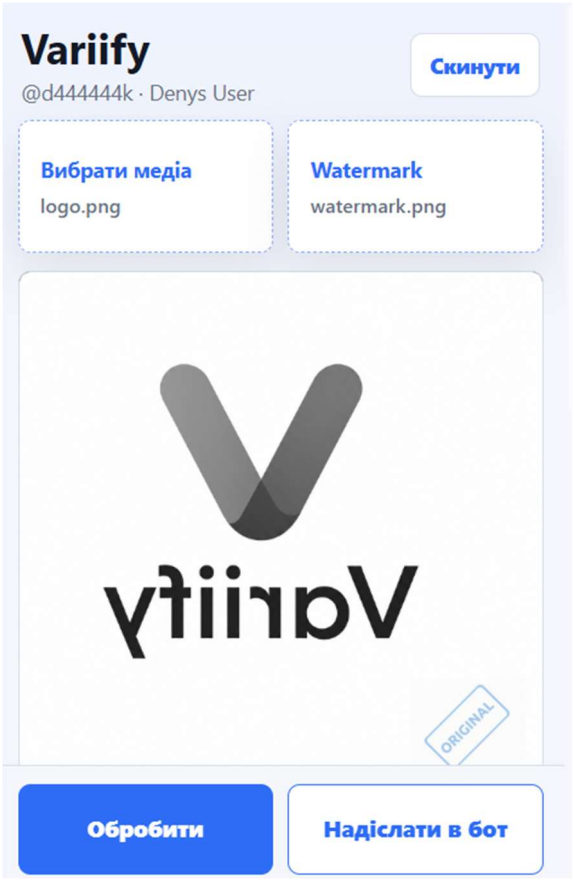


Рисунок 4.3 – Завантаження watermark та налаштування параметрів обробки

Після завершення обробки система формує нову версію зображення, яка відображається у вікні попереднього перегляду. На отриманому результаті можна спостерігати застосування вибраних параметрів та накладення водяного знака(рис. 4.4).

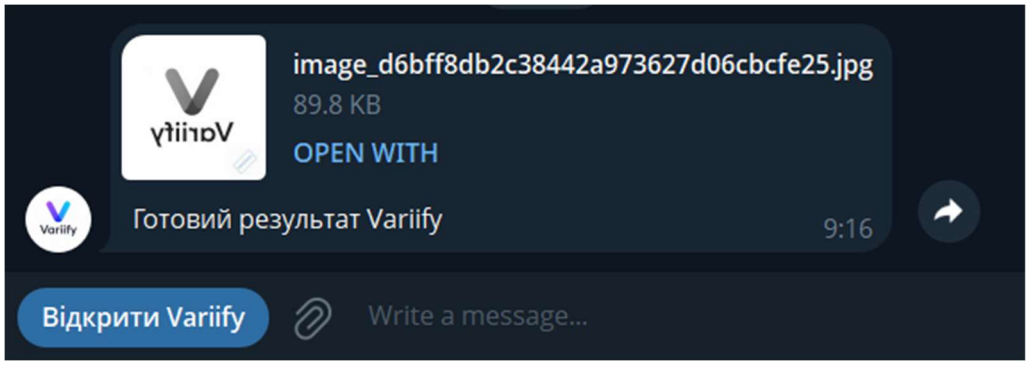


Рисунок 4.4 – Результат обробки зображення

Для перевірки роботи із відеоконтентом було виконано завантаження відеофайлу до системи. Після завершення завантаження у вікні попереднього перегляду відображається вбудований відеоплеєр (рис. 4.5).

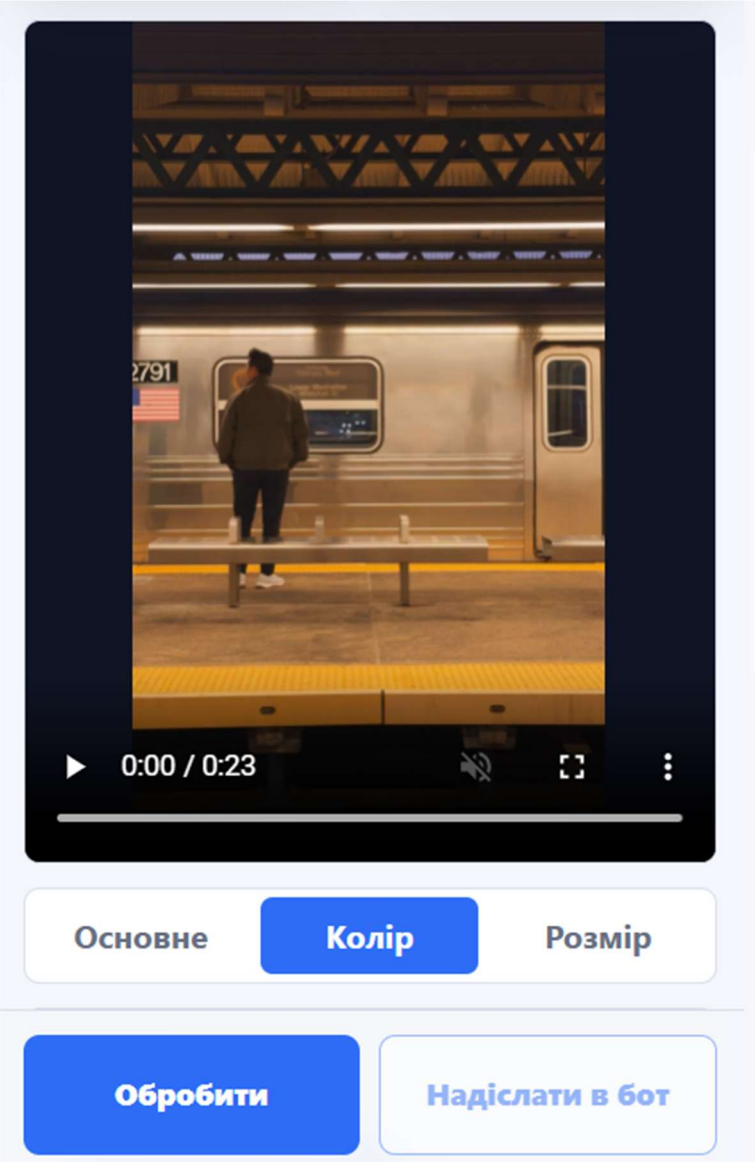


Рисунок 4.5 – Завантаження відеофайлу

Наступним етапом тестування є перевірка вкладки налаштування кольорових характеристик. Користувач може змінювати яскравість, контрастність, насиченість та значення кольорових каналів RGB (рис. 4.6).

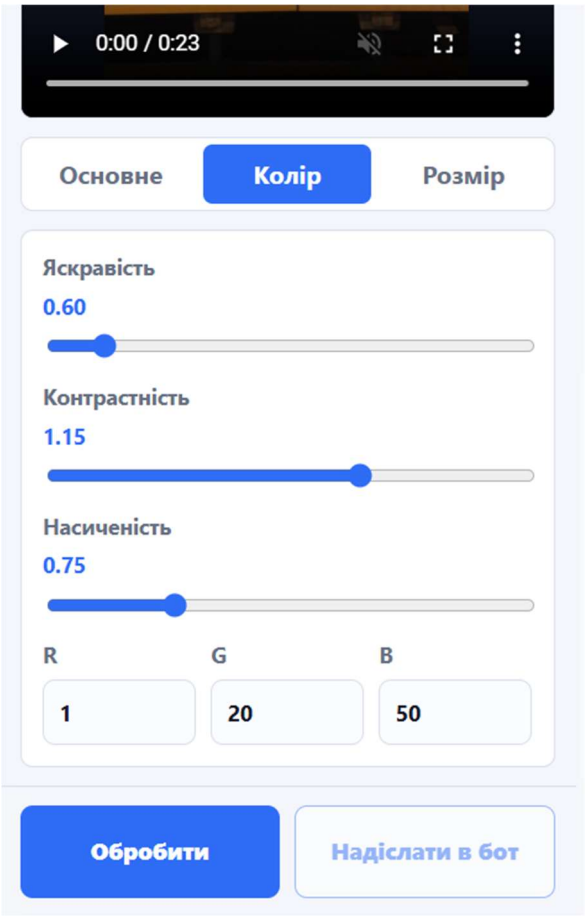


Рисунок 4.6 – Налаштування кольорових параметрів відео

Для перевірки геометричних параметрів виконується зміна ширини, висоти, частоти кадрів та прозорості водяного знака (рис. 4.7).

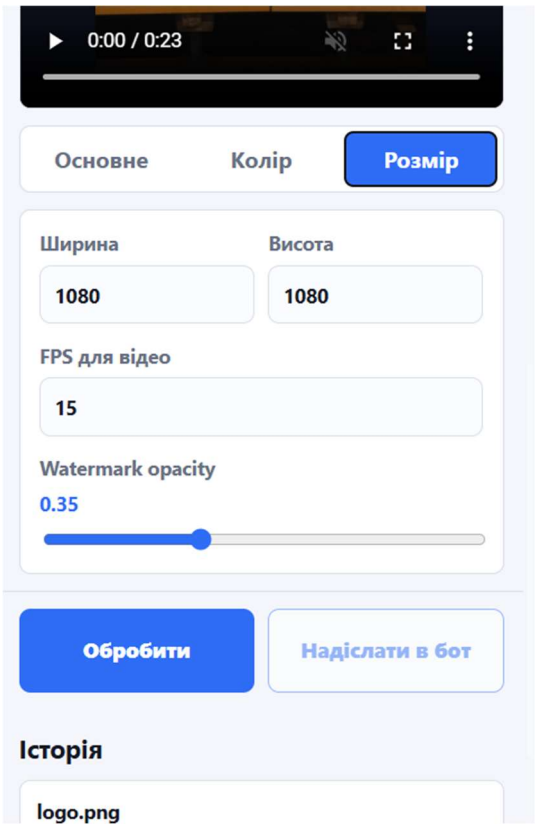


Рисунок 4.7 – Налаштування параметрів розміру та FPS

Після натискання кнопки «Обробити» серверна частина запускає алгоритм обробки відеофайлу через FFmpeg та виконує застосування всіх вибраних параметрів (рис. 4.8).

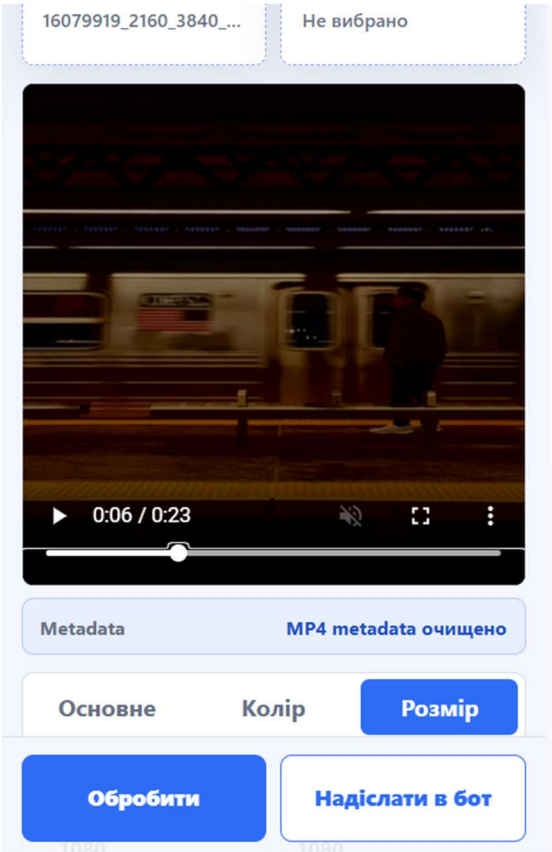


Рисунок 4.8 – Результат обробки відеофайлу

Окремо було перевірено механізм скидання параметрів редактора. Після натискання кнопки «Скинути» усі параметри повертаються до початкових значень, а користувач отримує відповідне повідомлення про успішне виконання операції (рис. 4.9).

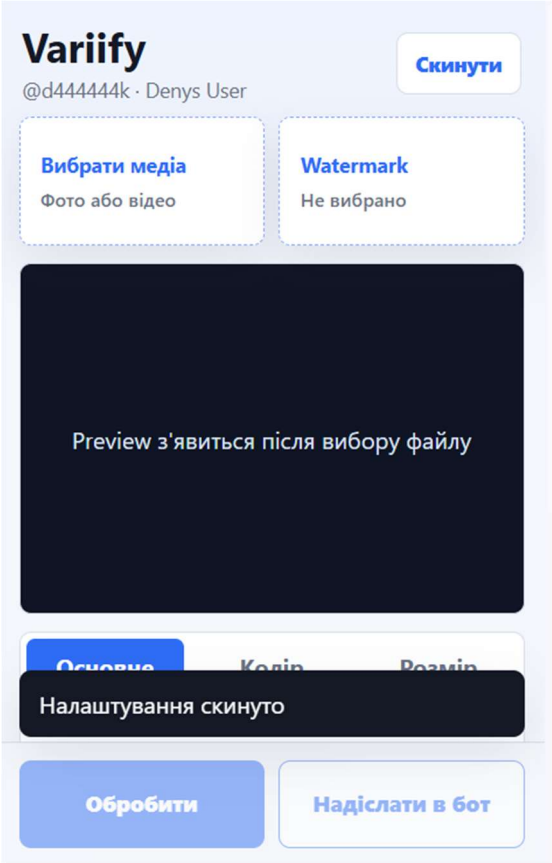


Рисунок 4.9 – Скидання параметрів редактора

Для перевірки інтеграції із Telegram Bot API було виконано надсилання обробленого файлу через кнопку «Надіслати в бот». Серверна частина успішно сформувала запит до Telegram API та передала результат користувачу безпосередньо у чат Telegram (рис. 4.10).

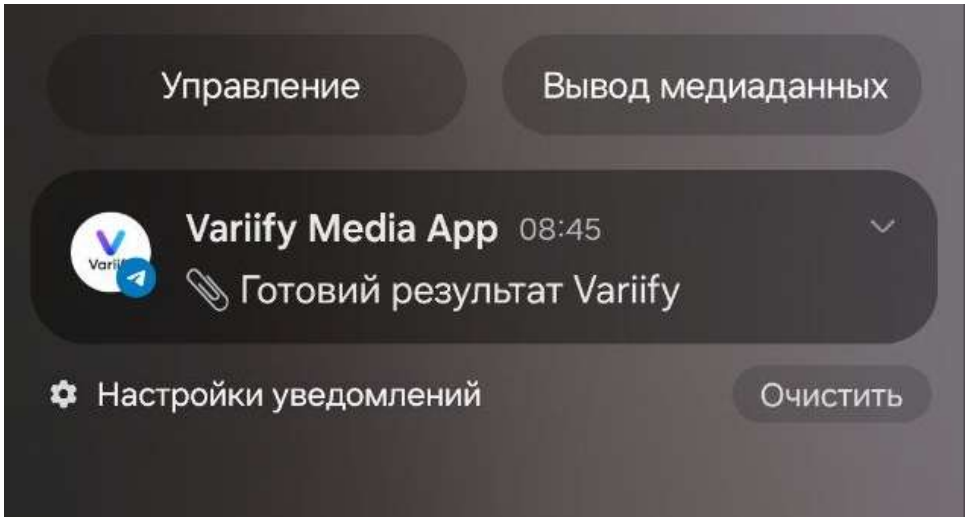


Рисунок 4.10 – Отримання результату через Telegram Bot API

4.4 Висновки до розділу 4

Під час тестування програмного забезпечення для автоматизованої обробки та унікалізації медіаконтенту були виконані перевірки основних функціональних можливостей системи. Було проведено тестування процесів завантаження зображень і відеофайлів, налаштування параметрів обробки, накладання водяних знаків, очищення метаданих, збереження історії операцій та надсилання результатів через Telegram Bot API.

Під час перевірки роботи редактора підтверджено коректність функціонування інструментів зміни кольорових параметрів, геометричних характеристик файлів, дзеркального відображення, випадкового обрізання та інших механізмів унікалізації медіаконтенту. Для відеофайлів перевірено роботу зміни FPS, масштабування та очищення службових метаданих.

Також було протестовано механізм взаємодії із базою даних SQLite, який забезпечує збереження інформації про користувачів, історію обробок та результати виконаних операцій. Перевірка інтеграції з Telegram Bot API підтвердила можливість автоматичного надсилання оброблених файлів безпосередньо до чату користувача.

Застосунок працює справно, критичних помилок під час тестування не виявлено.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи поставлена мета – розроблення Telegram Web App для автоматизованої обробки та унікалізації медіаконтенту – була повністю досягнута шляхом проєктування та реалізації програмного забезпечення Variify.

Аналіз предметної області, дослідження існуючих рішень та практична реалізація програмного продукту дозволяють сформулювати наступні висновки.

У ході виконання роботи було проведено аналіз сучасних методів автоматизованої обробки цифрового медіаконтенту та визначено основні підходи до створення систем унікалізації зображень і відеофайлів. Дослідження показало актуальність використання Telegram Web Apps як платформи для створення легкодоступних сервісів обробки мультимедійних даних без необхідності встановлення додаткового програмного забезпечення.

На основі результатів аналізу було спроєктовано архітектуру програмного забезпечення, яка включає клієнтську частину Telegram Web App, серверну частину на базі Flask, систему зберігання даних SQLite та інтеграцію з Telegram Bot API. Запропонована архітектура забезпечує чітке розмежування функціональних модулів та можливість подальшого масштабування системи.

У процесі реалізації було створено програмний продукт Variify, який підтримує автоматизовану обробку фотографій та відеофайлів. Для роботи із зображеннями використано бібліотеку Pillow, що дозволило реалізувати зміну яскравості, контрастності та насиченості, коригування кольорових каналів, дзеркальне відображення, випадкове обрізання, очищення EXIF metadata та накладання водяних знаків. Для обробки відеоконтенту використано програмний пакет FFmpeg, який забезпечує зміну частоти кадрів, масштабування відео, очищення службових метаданих та застосування додаткових параметрів унікалізації.

Розроблено структуру бази даних, що забезпечує зберігання інформації про користувачів, медіафайли, результати обробки та історію виконаних операцій. Використання реляційної моделі даних дозволило забезпечити цілісність інформації та ефективний доступ до результатів роботи системи.

Особливу увагу приділено інтеграції із Telegram Bot API. Реалізований механізм автоматичного надсилання результатів обробки безпосередньо до чату користувача підвищує зручність використання системи та дозволяє отримувати готовий контент без виконання додаткових дій.

Проведене тестування підтвердило коректність функціонування всіх основних модулів програмного забезпечення. Успішно перевірено процеси завантаження файлів, налаштування параметрів обробки, виконання операцій унікалізації, роботу з водяними знаками, збереження історії обробок та передачу результатів через Telegram. Результати тестування показали стабільну роботу системи та відповідність реалізованого функціоналу поставленим вимогам.

Таким чином, розроблений Telegram Web App Variify є функціонально завершеним програмним продуктом, готовим до практичного використання. Подальший розвиток системи може бути пов'язаний із впровадженням додаткових алгоритмів обробки медіаконтенту, використанням технологій штучного інтелекту для автоматичного підбору параметрів унікалізації та розширенням можливостей інтеграції з іншими сервісами Telegram.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Pallets. Flask Documentation. URL: https://flask.palletsprojects.com/ (date of access: 09.05.2026).
2. Telegram. Telegram Mini Apps. URL: https://core.telegram.org/bots/webapps (date of access: 09.05.2026).
3. Telegram. Telegram Bot API. URL: https://core.telegram.org/bots/api (date of access: 09.06.2026).
4. Pillow Developers. Pillow Documentation. URL: https://pillow.readthedocs.io/ (date of access: 11.05.2026).
5. FFmpeg Developers. FFmpeg Documentation. URL: https://ffmpeg.org/documentation.html (date of access: 11.05.2026).
6. SQLite Consortium. SQLite Documentation. URL: https://www.sqlite.org/docs.html (date of access: 19.05.2026).
7. Telegram. sendDocument. URL: https://core.telegram.org/bots/api#senddocument (date of access: 19.05.2026).
8. Telegram. Telegram Mini Apps. URL: https://core.telegram.org/bots/webapps (date of access: 19.05.2026).

ДОДАТОК А

Технічне завдання

Вступ

У сучасних умовах стрімкого розвитку цифрових технологій та соціальних мереж обсяги мультимедійного контенту постійно зростають. Фотографії та відеоматеріали стали основними засобами передачі інформації, реклами продукції, просування брендів та комунікації між користувачами. Значна частина контенту створюється для публікації в соціальних мережах, месенджерах, інформаційних платформах та інших цифрових сервісах, що зумовлює потребу в його швидкій обробці та адаптації до вимог конкретних платформ.

Разом із розвитком цифрового середовища зростає необхідність створення унікального медіаконтенту. Використання однакових зображень або відеоматеріалів на різних ресурсах може негативно впливати на ефективність рекламних кампаній, поширення інформації та просування контенту. У багатьох випадках виникає потреба у створенні декількох модифікованих версій одного медіафайлу, які відрізняються параметрами кольору, розмірами, метаданими або іншими характеристиками. Виконання таких операцій вручну потребує значних витрат часу та вимагає використання спеціалізованого програмного забезпечення.

Особливо актуальною ця проблема є для користувачів месенджера Telegram, який сьогодні використовується не лише як засіб обміну повідомленнями, а й як платформа для ведення бізнесу, просування контенту, роботи з клієнтами та керування інформаційними ресурсами. Велика кількість адміністраторів Telegram-каналів, контент-менеджерів та маркетологів регулярно працюють із графічними та відеоматеріалами, які необхідно адаптувати перед публікацією. При цьому використання окремих графічних редакторів або програм для монтажу відео значно ускладнює та сповільнює робочий процес.

Сучасна концепція Telegram Mini Apps (Telegram Web Apps) відкриває нові можливості для створення інтерактивних вебзастосунків, які працюють

безпосередньо всередині Telegram. Такий підхід дозволяє забезпечити користувачам швидкий доступ до функціоналу системи без встановлення додаткового програмного забезпечення та без переходу на сторонні вебресурси. Інтеграція вебзастосунку із Telegram Bot API дозволяє організувати повний цикл роботи з файлами — від завантаження та обробки до автоматичного отримання результатів у чаті користувача.

У рамках даної кваліфікаційної роботи планується розроблення Telegram Web App «Variify», призначеного для автоматизованої обробки та унікалізації медіаконтенту. Програмний продукт повинен забезпечувати завантаження фотографій і відеофайлів, зміну їхніх параметрів, очищення метаданих, накладання водяних знаків, виконання операцій коригування кольору та інших перетворень, спрямованих на підвищення унікальності контенту. Додатково система повинна підтримувати збереження історії обробок та надсилання готових результатів безпосередньо через Telegram Bot API.

Розроблення такого програмного забезпечення дозволить автоматизувати рутинні операції обробки медіаконтенту, скоротити час підготовки матеріалів до публікації та підвищити зручність роботи користувачів із цифровими файлами в екосистемі Telegram. Саме на вирішення зазначених задач спрямована дана робота.

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується відповідно до наказу кафедри комп’ютерних наук та інформаційних технологій Національного університету «Запорізька політехніка» та присвячена темі: «Розроблення вебзастосунку Telegram для автоматизованої обробки та унікалізації медіаконтенту».

А.2 Призначення розробки

Програмний продукт призначений для автоматизованої обробки фотографій та відеофайлів у середовищі Telegram.

Застосунок повинен забезпечувати:

– завантаження фото та відео;
– попередній перегляд медіафайлів;
– налаштування параметрів обробки;
– накладання водяних знаків;
– очищення метаданих;
– автоматичну унікалізацію контенту;
– збереження історії обробок;
– надсилання результатів через Telegram Bot API.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програмного забезпечення повинні включати:

– авторизацію користувача через Telegram Web App;
– завантаження фотографій;
– завантаження відеофайлів;
– завантаження watermark-зображень;
– попередній перегляд вибраного контенту;
– дзеркальне відображення зображень;
– дзеркальне відображення відео;
– переведення зображень у чорно-білий режим;
– переведення відео у чорно-білий режим;
– випадкове обрізання зображень;
– випадкове обрізання відеофайлів;

– очищення EXIF metadata;

– очищення службових метаданих відео;

– зміну яскравості;

– зміну контрастності;

– зміну насиченості;

– налаштування RGB-каналів;

– зміну ширини та висоти медіафайлів;

– зміну частоти кадрів (FPS);

– регулювання прозорості watermark;

– запуск процесу обробки;

– перегляд результатів обробки;

– збереження результатів у системі;

– перегляд історії обробок;

– повторне відкриття результату;

– видалення записів історії;

– надсилання результатів через Telegram Bot API.

А.3.2 Умови експлуатації

Для коректного функціонування програмного забезпечення необхідні:

– підключення до мережі Internet;

– доступ до Telegram;

– активний Telegram Bot;

– встановлений Python 3.10 або новіше;

– встановлений FFmpeg;

– наявність бібліотек Flask, Pillow та Requests;

– база даних SQLite.

А.3.3 Вимоги до складу та параметрів технічних засобів

Для стабільної роботи програмного забезпечення мінімальні параметри системи повинні становит:

- процесор: 2-ядерний x86-64, частота від 2.0 ГГц;
- оперативна пам’ять: від 4 ГБ;
- накопичувач: від 10 ГБ SSD;
- операційна система: Windows 10/11 або Linux.

А.4 Порядок контролю та приймання

Починаючи з дати видачі завдання на дипломну кваліфікаційну роботу, протягом наступних 5 тижнів має бути забезпечене виконання роботи за календарним планом, що міститься в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом – строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів формується пояснювальна записка до дипломної кваліфікаційної роботи.

Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б

Текст програми

```
import json
import mimetypes
import os
import urllib.error
import urllib.request
import uuid
from pathlib import Path

from flask import Flask, jsonify, request, send_file
from werkzeug.exceptions import HTTPException, RequestEntityTooLarge
from werkzeug.utils import secure_filename

from backend.database import get_db, init_db, row_to_dict, upsert_user
from backend.media_processor import detect_media_type, process_media

BASE_DIR = Path(__file__).resolve().parent
UPLOAD_DIR = BASE_DIR / "uploads"
OUTPUT_DIR = BASE_DIR / "outputs"
WATERMARK_DIR = BASE_DIR / "watermarks"

app = Flask(__name__, static_folder="static", static_url_path="/static")
app.config["MAX_CONTENT_LENGTH"] = 250 * 1024 * 1024

@app.before_request
def ensure_database():
    init_db()

@app.errorhandler(RequestEntityTooLarge)
def file_too_large(_error):
```

```
    return jsonify({"error": "Файл завеликий. Спробуйте менше фото або відео."}), 413

@app.errorhandler(Exception)
def api_error(error):
    if not request.path.startswith("/api/"):
        if isinstance(error, HTTPException):
            return error
        raise error
    if isinstance(error, HTTPException):
        return jsonify({"error": error.description}), error.code
    return jsonify({"error": f"Помилка сервера: {error}"}), 500

@app.get("/")
def index():
    return send_file(BASE_DIR / "static" / "index.html")

@app.get("/favicon.ico")
def favicon():
    return "", 204

@app.get("/health")
def health():
    return jsonify({"status": "ok"})

@app.post("/api/users")
def save_user():
    user = upsert_user(request.json or {})
    return jsonify(user)
```

```
@app.post("/api/upload")
def upload_media():
    user = upsert_user(request.form)
    file = request.files.get("file")
    if not file:
        return jsonify({"error": "File is required"}), 400

    media_type = detect_media_type(file.filename)
    if media_type == "unknown":
        return jsonify({"error": "Unsupported file type"}), 400

    UPLOAD_DIR.mkdir(parents=True, exist_ok=True)
    original = secure_filename(file.filename)
    stored = f"{uuid.uuid4().hex}_{original}"
    upload_path = UPLOAD_DIR / stored
    file.save(upload_path)

    with get_db() as conn:
        cursor = conn.execute(
            """
            INSERT INTO media_files (user_id, original_filename, stored_filename,
media_type, upload_path)
            VALUES (?, ?, ?, ?, ?)
            """,
            (user["id"], original, stored, media_type, str(upload_path)),
        )
        media_id = cursor.lastrowid

    return jsonify({"id": media_id, "media_type": media_type, "filename": original})
```

```
@app.post("/api/watermark")
def upload_watermark():
    user = upsert_user(request.form)
    file = request.files.get("file")
    if not file:
        return jsonify({"error": "Watermark file is required"}), 400

    WATERMARK_DIR.mkdir(parents=True, exist_ok=True)
    original = secure_filename(file.filename)
    stored = f"{uuid.uuid4().hex}_{original}"
    path = WATERMARK_DIR / stored
    file.save(path)

    with get_db() as conn:
        cursor = conn.execute(
            "INSERT INTO watermarks (user_id, filename, file_path) VALUES (?, ?,
?)",
            (user["id"], original, str(path)),
        )
    return jsonify({"id": cursor.lastrowid, "filename": original})

@app.post("/api/process")
def process_uploaded_media():
    payload = request.json or {}
    user = upsert_user(payload.get("user", {}))
    media_id = int(payload.get("media_id", 0))
    options = payload.get("options", {})

    with get_db() as conn:
        media = conn.execute(
```

```
        "SELECT * FROM media_files WHERE id = ? AND user_id = ?", (media_id,
user["id"])
    ).fetchone()
    if not media:
        return jsonify({"error": "Media file not found"}), 404

    watermark = conn.execute(
        "SELECT * FROM watermarks WHERE user_id = ? ORDER BY id DESC
LIMIT 1", (user["id"],)
    ).fetchone()
    cursor = conn.execute(
        """
            INSERT INTO processing_tasks (user_id, media_file_id, status,
options_json)
        VALUES (?, ?, ?, ?)
        """,
        (user["id"], media_id, "processing", json.dumps(options)),
    )
    task_id = cursor.lastrowid

  try:
            watermark_path = watermark["file_path"] if watermark and
options.get("watermark") else None
    result_path = process_media(
        Path(media["upload_path"]),
        OUTPUT_DIR,
        media["media_type"],
        options,
        watermark_path,
    )
```

```
        with get_db() as conn:
            conn.execute(
                "UPDATE media_files SET result_path = ? WHERE id = ?",
                (str(result_path), media_id),
            )
            conn.execute(
                    "UPDATE processing_tasks SET status = 'done', updated_at =
CURRENT_TIMESTAMP WHERE id = ?",
                (task_id,),
            )
        return jsonify(
            {
                "task_id": task_id,
                "status": "done",
                "strip_metadata": bool(options.get("strip_metadata", True)),
                "preview_url": f"/api/preview/{task_id}",
                "download_url": f"/api/download/{task_id}",
                "send_url": f"/api/send/{task_id}",
            }
        )
    except Exception as exc:
        with get_db() as conn:
            conn.execute(
                """
                UPDATE processing_tasks
                        SET status = 'failed', error_message = ?, updated_at =
CURRENT_TIMESTAMP
                WHERE id = ?
                """,
                (str(exc), task_id),
```

```
            )
        return jsonify({"task_id": task_id, "status": "failed", "error": str(exc)}), 500


@app.get("/api/tasks/<int:task_id>")
def task_status(task_id):
    with get_db() as conn:
        task = conn.execute("SELECT * FROM processing_tasks WHERE id = ?",
(task_id,)).fetchone()
    if not task:
        return jsonify({"error": "Task not found"}), 404
    return jsonify(row_to_dict(task))


@app.get("/api/download/<int:task_id>")
def download_result(task_id):
    with get_db() as conn:
        row = conn.execute(
            """
            SELECT mf.result_path, mf.original_filename
            FROM processing_tasks pt
            JOIN media_files mf ON mf.id = pt.media_file_id
            WHERE pt.id = ? AND pt.status = 'done'
            """,
            (task_id,),
        ).fetchone()
    if not row or not row["result_path"]:
        return jsonify({"error": "Result not found"}), 404
    suffix = Path(row["result_path"]).suffix or ".jpg"
    response = send_file(
        row["result_path"],
        as_attachment=True,
```

```
        download_name=f"variify_result_{task_id}{suffix}",
        mimetype=mimetypes.guess_type(row["result_path"])[0] or "application/octet-stream",
        max_age=0,
    )
    response.headers["Cache-Control"] = "no-store"
    response.headers["Access-Control-Expose-Headers"] = "Content-Disposition"
    return response

@app.post("/api/send/<int:task_id>")
def send_result_to_bot(task_id):
    payload = request.json or {}
    user = upsert_user(payload.get("user", {}))
    bot_token = os.getenv("TELEGRAM_BOT_TOKEN")
    if not bot_token:
        return jsonify({"error": "TELEGRAM_BOT_TOKEN is not configured"}), 400

    with get_db() as conn:
        row = conn.execute(
            """
            SELECT mf.result_path, mf.media_type, u.telegram_id
            FROM processing_tasks pt
            JOIN media_files mf ON mf.id = pt.media_file_id
            JOIN users u ON u.id = pt.user_id
            WHERE pt.id = ? AND pt.status = 'done' AND pt.user_id = ?
            """,
            (task_id, user["id"]),
        ).fetchone()

    if not row or not row["result_path"]:
```

```
        return jsonify({"error": "Result not found"}), 404

    result_path = Path(row["result_path"])
    if not result_path.exists():
        return jsonify({"error": "Result file is missing"}), 404

    ok, error = send_telegram_document(
        bot_token=bot_token,
        chat_id=row["telegram_id"],
        file_path=result_path,
        caption="Готовий результат Variify",
    )
    if not ok:
        return jsonify({"error": error}), 502
    return jsonify({"status": "sent"})

def send_telegram_document(bot_token, chat_id, file_path, caption):
    boundary = f"----VariifyBoundary{uuid.uuid4().hex}"
    mime_type = mimetypes.guess_type(file_path)[0] or "application/octet-stream"
    parts = []

    for name, value in {"chat_id": chat_id, "caption": caption}.items():
        part = (
            f"--{boundary}\r\n"
            f'Content-Disposition: form-data; name="{name}"\r\n\r\n'
            f"{value}\r\n"
        ).encode("utf-8")
        parts.append(part)

    file_header = (
```

```
        f"--{boundary}\r\n"
                        f'Content-Disposition: form-data; name="document"; filename="{file_path.name}"\r\n'
        f"Content-Type: {mime_type}\r\n\r\n"
    ).encode("utf-8")
    body = b"".join(parts) + file_header + file_path.read_bytes() + f"\r\n--{boundary}--\r\n".encode("utf-8")

    request_obj = urllib.request.Request(
        f"https://api.telegram.org/bot{bot_token}/sendDocument",
        data=body,
        headers={"Content-Type": f"multipart/form-data; boundary={boundary}"},
        method="POST",
    )

    try:
        with urllib.request.urlopen(request_obj, timeout=60) as response:
            result = json.loads(response.read().decode("utf-8"))
    except urllib.error.HTTPError as exc:
        return False, exc.read().decode("utf-8", errors="replace")
    except Exception as exc:
        return False, str(exc)

    if not result.get("ok"):
        return False, json.dumps(result, ensure_ascii=False)
    return True, ""

@app.get("/api/preview/<int:task_id>")
def preview_result(task_id):
    with get_db() as conn:
```

```
        row = conn.execute(
            """
            SELECT mf.result_path
            FROM processing_tasks pt
            JOIN media_files mf ON mf.id = pt.media_file_id
            WHERE pt.id = ? AND pt.status = 'done'
            """,
            (task_id,),
        ).fetchone()
    if not row or not row["result_path"]:
        return jsonify({"error": "Result not found"}), 404
    return send_file(row["result_path"], conditional=True)

@app.delete("/api/media/<int:media_id>")
def delete_media(media_id):
    payload = request.json or {}
    user = upsert_user(payload.get("user", {}))

    with get_db() as conn:
        media = conn.execute(
          "SELECT * FROM media_files WHERE id = ? AND user_id = ?", (media_id,
user["id"])
        ).fetchone()
        if not media:
            return jsonify({"error": "Media file not found"}), 404

        upload_path = media["upload_path"]
        result_path = media["result_path"]
        conn.execute("DELETE FROM processing_tasks WHERE media_file_id = ?",
(media_id,))
```

```
        conn.execute("DELETE FROM media_files WHERE id = ?", (media_id,))

    safe_unlink(upload_path, UPLOAD_DIR)
    safe_unlink(result_path, OUTPUT_DIR)
    return jsonify({"status": "deleted"})

def safe_unlink(path_value, allowed_dir):
    if not path_value:
        return
    path = Path(path_value).resolve()
    allowed = allowed_dir.resolve()
    if allowed not in path.parents:
        return
    if path.exists() and path.is_file():
        path.unlink()

@app.get("/api/history/<telegram_id>")
def user_history(telegram_id):
    with get_db() as conn:
        rows = conn.execute(
            """
            SELECT mf.id, mf.original_filename, mf.media_type, mf.created_at,
                pt.id AS task_id, pt.status, pt.updated_at, pt.options_json, mf.result_path
            FROM users u
            JOIN media_files mf ON mf.user_id = u.id
            LEFT JOIN processing_tasks pt ON pt.media_file_id = mf.id
            WHERE u.telegram_id = ?
            ORDER BY mf.id DESC
            LIMIT 30
            """,
```

```
            (str(telegram_id),),
        ).fetchall()
    history = []
    for row in rows:
        item = dict(row)
        options = {}
        if item.get("options_json"):
            try:
                options = json.loads(item["options_json"])
            except json.JSONDecodeError:
                options = {}
        item["strip_metadata"] = bool(options.get("strip_metadata", True))
                if item.get("task_id") and item.get("status") == "done" and
item.get("result_path"):
            item["preview_url"] = f"/api/preview/{item['task_id']}"
            item["download_url"] = f"/api/download/{item['task_id']}"
            item["send_url"] = f"/api/send/{item['task_id']}"
        item.pop("result_path", None)
        item.pop("options_json", None)
        history.append(item)
    return jsonify(history)

if __name__ == "__main__":
    init_db()
    app.run(host="0.0.0.0", port=5000, debug=False, use_reloader=False)
```

ДОДАТОК В

Слайди презентації

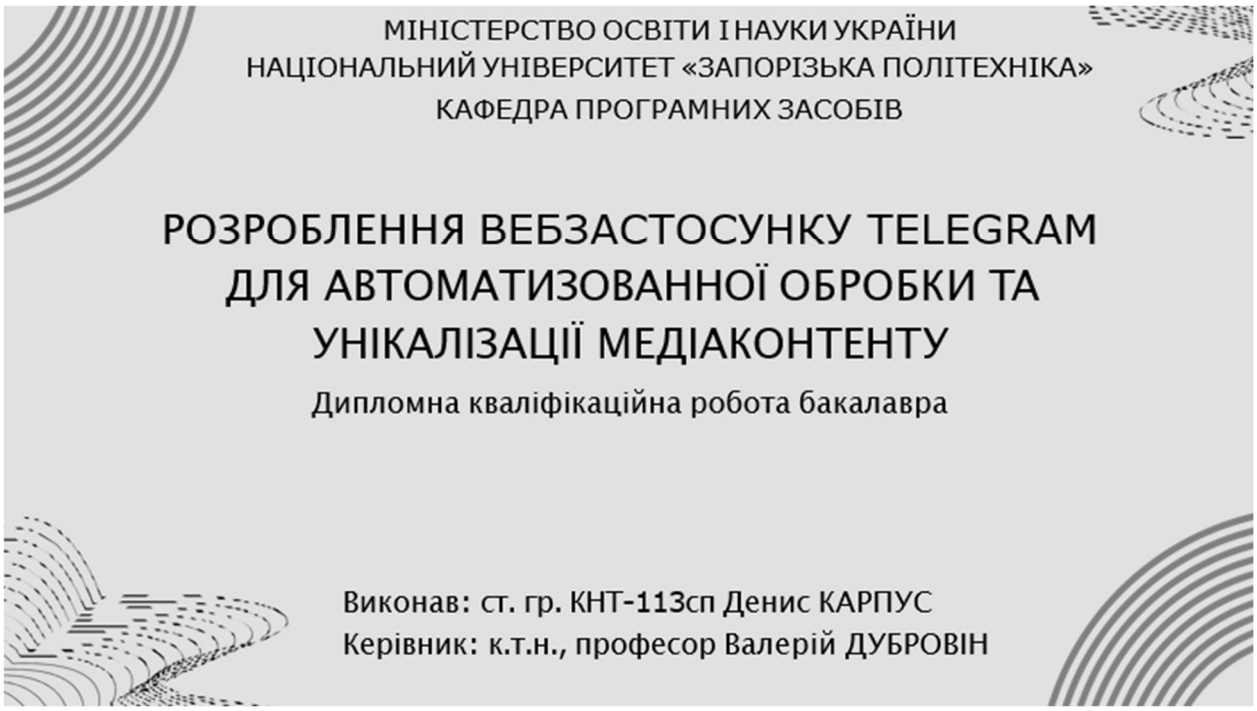


Рисунок В.1 – Слайд 1

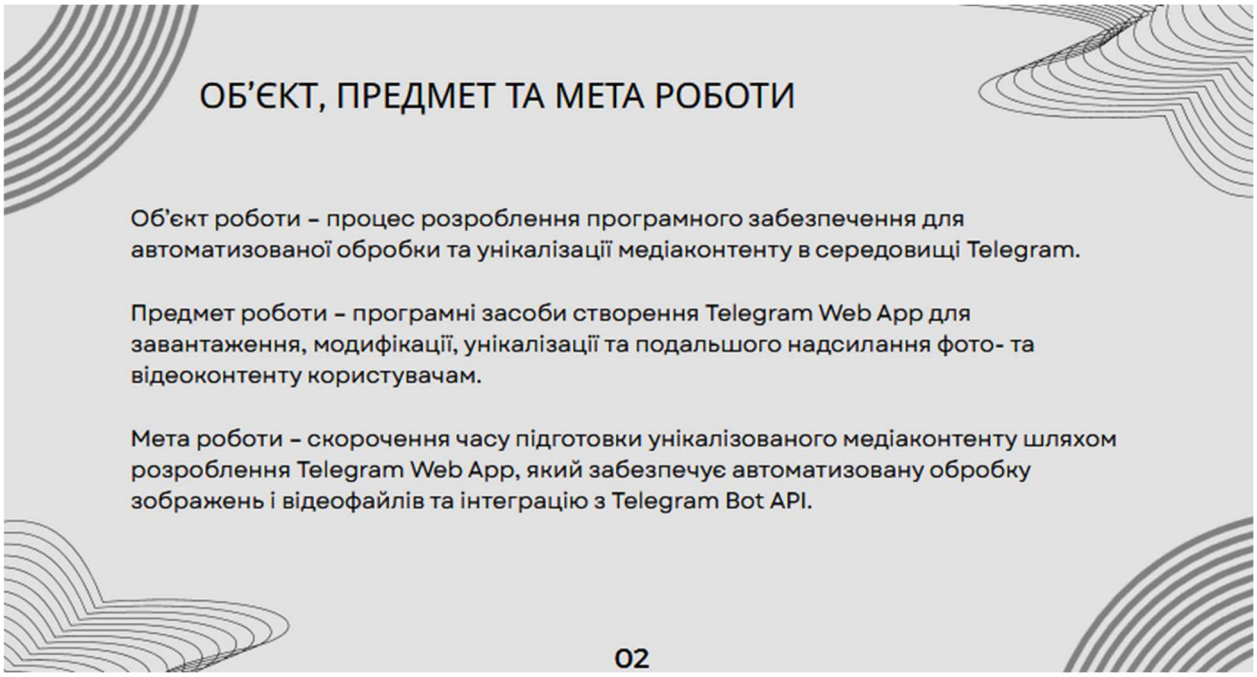


Рисунок В.2 – Слайд 2

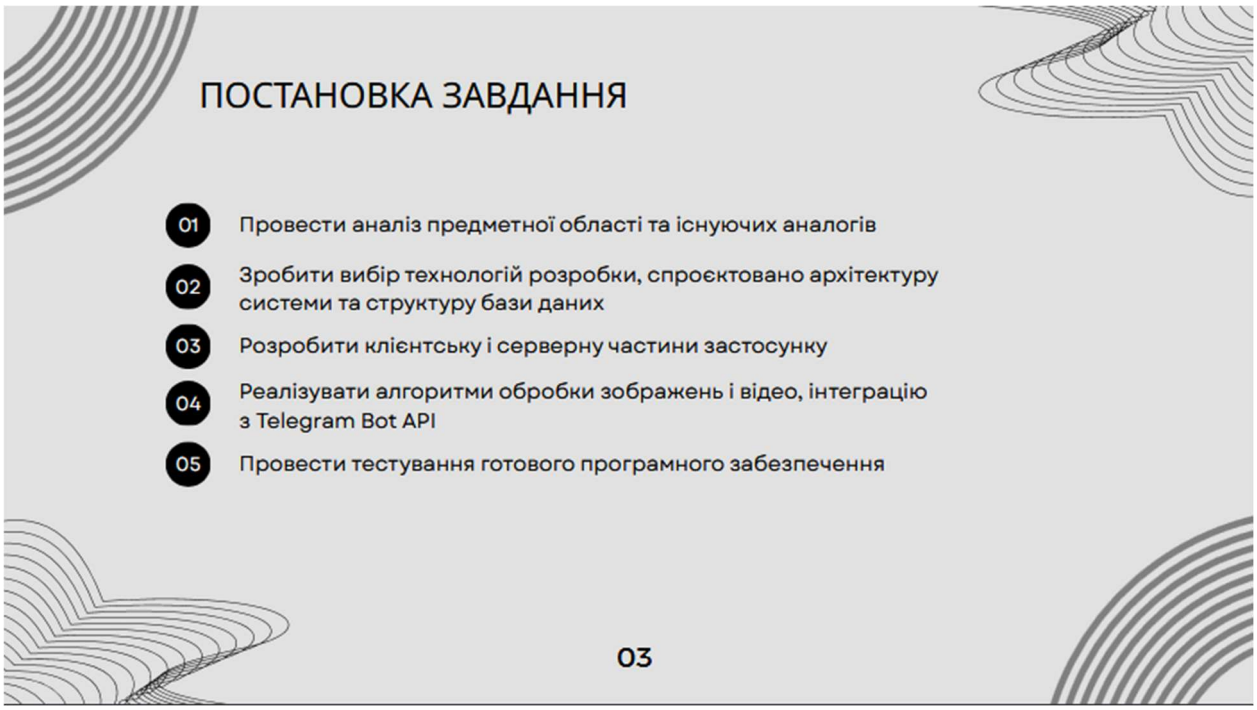


Рисунок В.3 – Слайд 3

Характеристика	Canva	Kapwing	Telegram Bots	Variify
Обробка зображень	+	+	+/-	+
Обробка відео	-	+	+/-	+
Додавання watermark	+	+	-	+
Очищення метаданих	-	-	-	+
Очищення EXIF metadata	-	-	+	+
Інтеграція з Telegram	+	+	+	+

Рисунок В.4 – Слайд 4

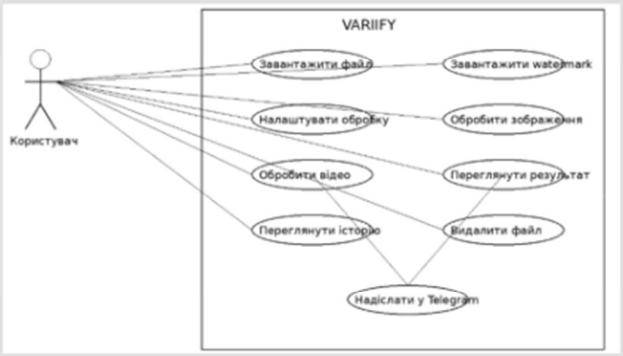
ПОРІВНЯННЯ ЗАСОБІВ РОЗРОБКИ

Критерій	Python	Java	C#	PHP
Простота розробки	Висока	Середня	Середня	Висока
Кількість бібліотек	Висока	Висока	Висока	Середня
Підтримка веброзробки	Висока	Висока	Висока	Висока
Робота з медіафайлами	Висока	Середня	Середня	Низька
Швидкість розробки	Висока	Середня	Середня	Низька

05

Рисунок В.5 – Слайд 5

USE CASE ДІАГРАМА СИСТЕМИ



06

Рисунок В.6 – Слайд 6

Рисунок В.7 – Слайд 7

Рисунок В.8 – Слайд 8

Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

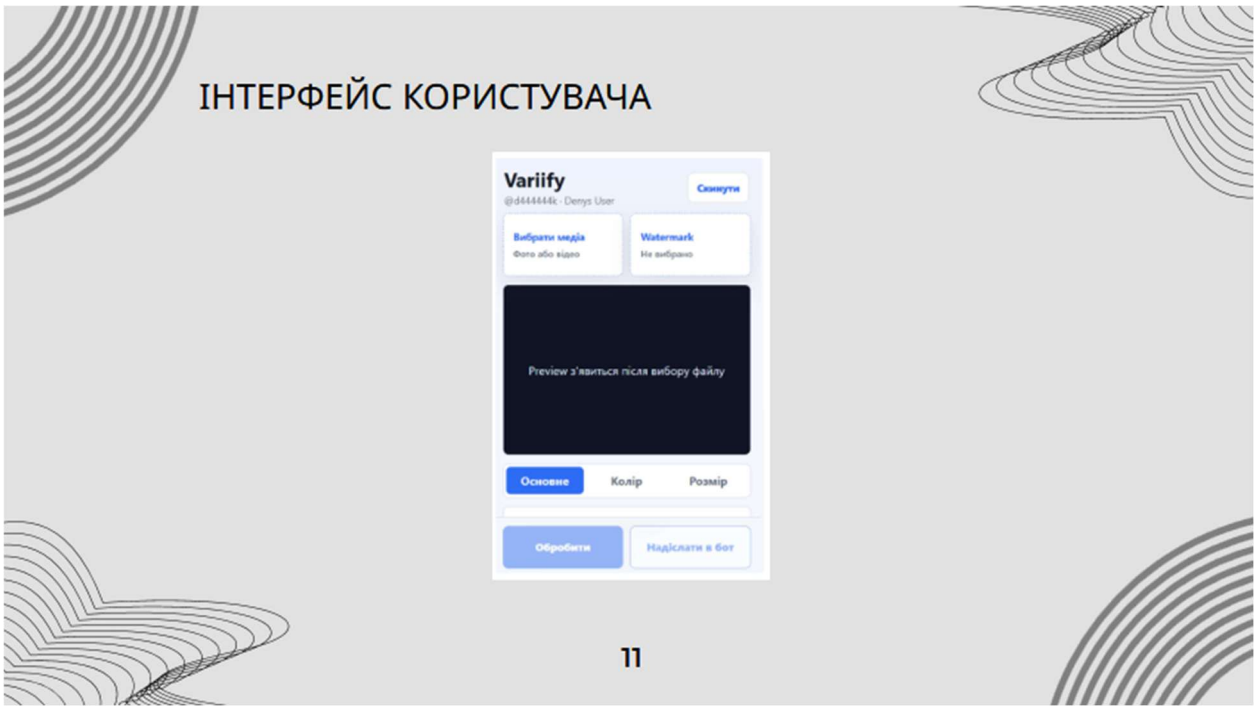


Рисунок В.11 – Слайд 11

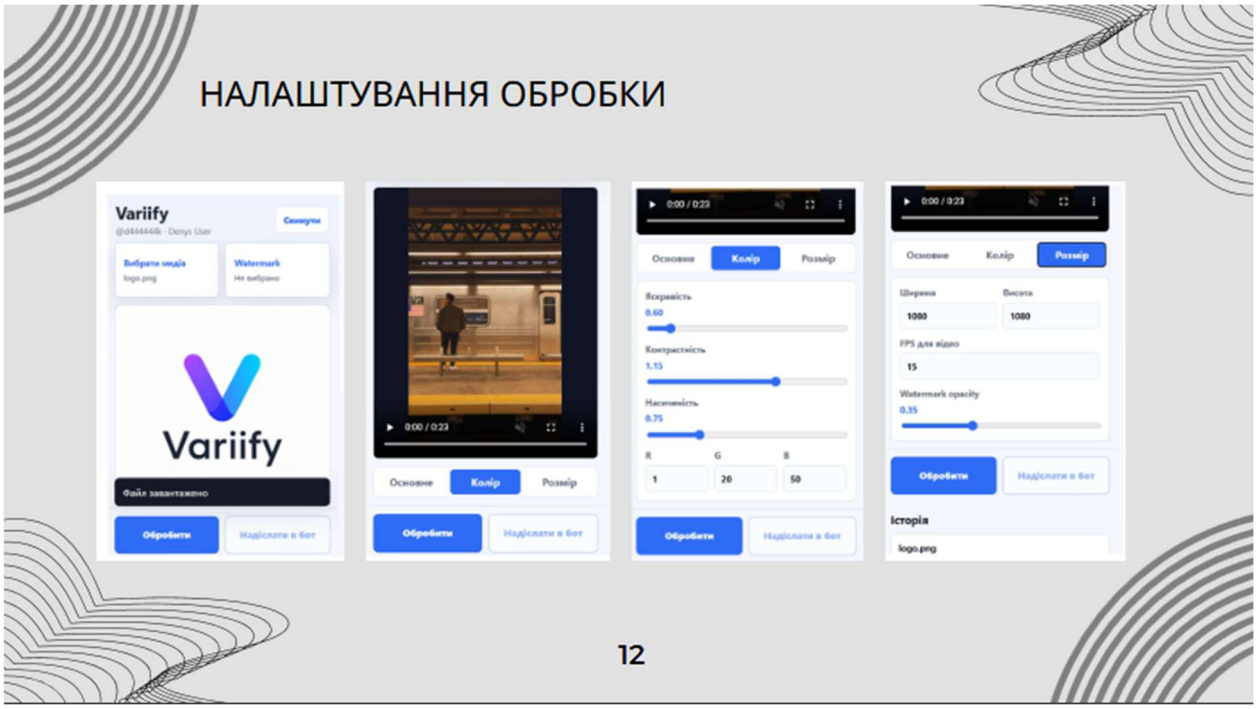


Рисунок В.12 – Слайд 12

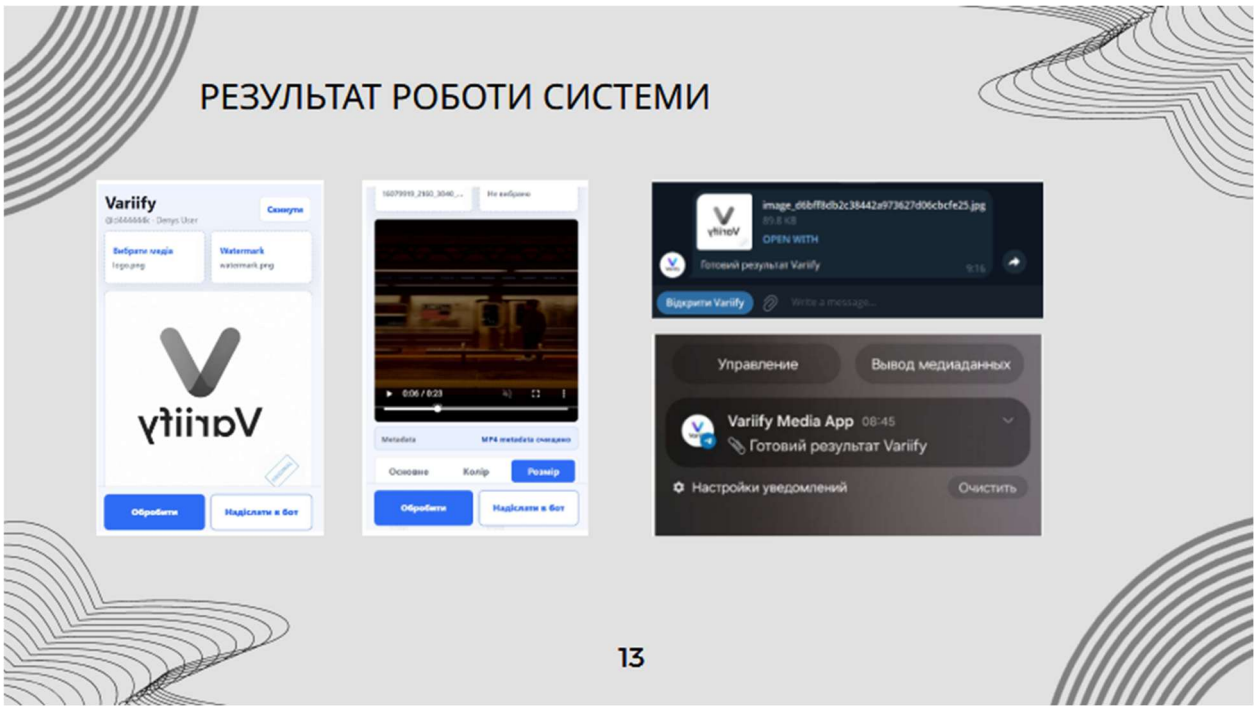


Рисунок В.13 – Слайд 13

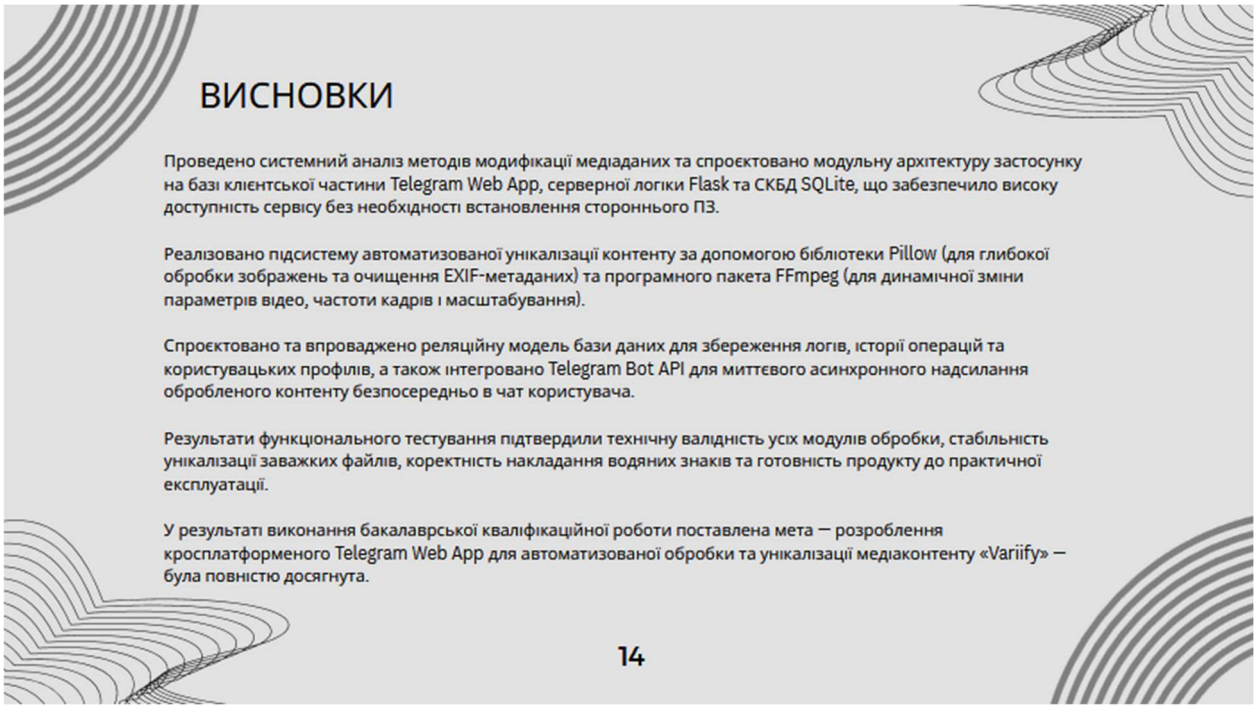


Рисунок В.14 – Слайд 14