Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ СКЛЕЙКИ ВІДЕО З ГЕТЕРОГЕННИХ ЧАСТИН (ЗА РОЗДІЛЬНОЮ ЗДАТНІСТЮ ТА ФОРМАТОМ)
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-1
Дудник М.В
(прізвище, ініціали)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір В.П
(посада, прізвище, ініціали)

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дуднику Максиму Вячеславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для склейки відео з гетерогенних частин (за роздільною здатністю та форматом)

затверджена наказом університету від 19 травня 2025 року № 381
2. Термін подання здобувачем роботи до екзаменаційної комісії 28 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, офіційна документація та приклади використання бібліотеки FFmpeg.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд сучасних форматів відеофайлів та їх характеристик (роздільна здатність, кодек, частота кадрів тощо).
2. Аналіз існуючих інструментів для обробки та склейки відео.
3. Розробка алгоритму нормалізації гетерогенних відеофрагментів за параметрами.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність склейки відео з гетерогенних частин, постановка задачі, тестові відео.

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-13.04.25	
3	Аналіз літератури з досліджуваної проблеми	14.04.25-18.04.25	
4	Аналіз технічних засобів	19.04.25-25.04.25	
5	Розробка методу	26.04.25-30.04.25	
6	Програмна реалізація	01.05.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____проф. Машталір В.П_____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 64 с., 11 рис., 30 джерел.

СКЛЕЙКА ВІДЕО, ГЕТЕРОГЕННІ ВІДЕОФАЙЛИ, ВЕБЗАСТОСУНОК, FFMPEG, TYPESCIPT, NEXT.JS, ОБРОБКА ВІДЕО, НОРМАЛІЗАЦІЯ ПАРАМЕТРІВ, ВІДЕОКОДЕКИ.

Об’єктом роботи є процес автоматизованого об’єднання відеофрагментів з різними технічними параметрами у вебсередовищі.

Метою роботи є створення вебзастосунку, який дозволяє користувачам об’єднувати гетерогенні відео без втрати якості та потреби в технічних навичках.

У роботі проаналізовано сучасні інструменти склейки відео, обґрунтовано вибір технологій, спроєктовано клієнт-серверну архітектуру. Реалізовано модуль нормалізації, що узгоджує частоту кадрів, роздільну здатність, орієнтацію та інші параметри.

Результатом є вебзастосунок, який автоматично визначає технічні характеристики відео, обробляє фрагменти та формує єдиний відеофайл у форматі MP4. Застосунок протестовано на реальних прикладах з нестандартними відеоформатами.

Рішення орієнтоване на користувачів освітньої, медійної та корпоративної сфер, які потребують простого, стабільного та універсального інструменту склейки відео.

VIDEO MERGING, HETEROGENEOUS VIDEO FILES, WEB APPLICATION, FFMPEG, TYPESCRIPT, NEXT.JS, VIDEO PROCESSING, PARAMETER NORMALIZATION, VIDEO CODECS.

The object of the work is the process of automated merging of video clips with different technical parameters in a web environment.

The purpose of the work is to develop a web application that enables users to merge heterogeneous videos without quality loss and without requiring technical skills.

The paper analyzes current tools for video merging, justifies technology choices, and designs a client-server architecture. A normalization module has been implemented to align frame rates, resolutions, orientations, and other parameters.

As a result, a web application was created that automatically detects video properties, processes fragments, and produces a single MP4 video file. The application was tested on real-world examples with non-standard formats.

This solution targets users in educational, media, and business fields who require a simple, stable, and universal video merging tool.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (інтерфейс прикладного програмування)

CLI – Command-Line Interface (інтерфейс командного рядка)

FFmpeg – Free Fast Multimedia Platform (бібліотека для обробки аудіо- та відеофайлів)

SPA – Single Page Application (односторінковий вебзастосунок)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (досвід користувача)

SSR – Server-Side Rendering (рендеринг на стороні сервера)

H.264 – відеокодек, стандарт стиснення з втратами, широко використовуваний у форматі MP4

MP4 – MPEG-4 Part 14 (формат мультимедійного контейнера для зберігання відео та аудіо)

WebM – відкритий формат мультимедійного контейнера, орієнтований на веб

FPS – Frames Per Second (кадрів за секунду)

HD – High Definition (висока роздільна здатність, зазвичай 1280×720 пікселів)

Next.js – фреймворк для React з підтримкою SSR та генерації статичних сайтів

TypeScript – надбудова над JavaScript з підтримкою типізації

Node.js – середовище виконання JavaScript-коду на сервері

REST – Representational State Transfer (архітектурний стиль вебсервісів)

Codec – алгоритм кодування/декодування відео- або аудіоінформації

Padding – додавання полів (чорних смуг) для збереження пропорцій

Cropping – обрізання кадру для вирівнювання формату

Scaling – масштабування відео до потрібної роздільної здатності

ВСТУП

У XXI столітті цифрові технології відіграють ключову роль у розвитку освіти, медіа, маркетингу та комунікацій. Особливо значущим є поширення відеоконтенту, який став основним способом передачі інформації, навчання та розваг. Зі зростанням кількості відеоматеріалів, постає проблема об’єднання відеофрагментів, що мають різні технічні параметри.

На практиці користувачі часто стикаються з необхідністю склеїти відео, записані в різних умовах і на різне обладнання. Проте більшість наявних рішень або є надто складними для новачків, або мають значні функціональні обмеження. Це створює бар’єр для широкого кола користувачів, які прагнуть швидко отримати якісний результат без спеціальних технічних знань.

У цьому контексті актуальним стає створення простого у використанні вебзастосунку, що дозволяє автоматично нормалізувати параметри відеофрагментів і об’єднувати їх в один цілісний відеофайл, незалежно від початкових розбіжностей. Такий інструмент має працювати стабільно у браузері, не вимагати встановлення програмного забезпечення та бути доступним на будь-якому пристрої від комп’ютера до смартфона.

Актуальність даної роботи зумовлена потребою у доступному, універсальному та надійному рішенні для обробки гетерогенних відео, яке відповідає вимогам сучасного середовища. Вона поєднує технології веброзробки, автоматизації відеообробки та адаптивної нормалізації мультимедійного контенту.

1 АНАЛІЗ ЗАСТОСУНКІВ ДЛЯ СКЛЕЙКИ ВІДЕО З ГЕТЕРОГЕННИХ ЧАСТИН

1.1 Актуальність обробки та склейки відео з різними параметрами

У сучасному світі відео є ключовим інструментом комунікації, навчання, розваг і презентацій [1]. Щодня генеруються тисячі відеофрагментів, знятих на різні пристрої з різними технічними параметрами, роздільною здатністю, форматом, частотою кадрів, орієнтацією та якістю [2].

На практиці часто виникає потреба поєднати такі різнорідні відео в один цілісний ролик. Ця задача постає як перед фахівцями, так і перед звичайними користувачами, які не мають досвіду у відеомонтажі. Проблема ускладнюється тим, що існуючі інструменти або є складними для опанування (Adobe Premiere Pro, DaVinci Resolve), або мають функціональні обмеження (Clideo, Kapwing). Командні утиліти, як ffmpeg, забезпечують максимальну гнучкість, проте вимагають знань технічного рівня.

Окрему категорію становлять викладачі, студенти та фахівці дистанційного навчання, які часто працюють з фрагментами відео, записаними у різних умовах: аудиторіях, відеозв’язку, мобільних пристроях тощо.

У зв’язку з цим постає потреба в інструменті, який би дозволяв автоматично поєднувати відеофайли з різними параметрами без втрати якості та без потреби в складному налаштуванні.

З огляду на тенденції розвитку вебзастосунків та хмарних сервісів, актуальним є створення простого у використанні вебінструменту, який би забезпечував автоматизовану обробку гетерогенних відеофрагментів і формував стабільний результат, готовий до використання на будь-якому пристрої.

1.2 Аналіз існуючих програмних рішень для склейки відео

Професійні редактори відео залишаються одним із найпоширеніших інструментів для обробки відеоматеріалів, які потребують високої якості та налаштувань. Таке програмне забезпечення зазвичай орієнтоване як на фахівців зі сфери медіа, так і на досвідчених користувачів, які працюють з відео на регулярній основі. Вони дозволяють повноцінно редагувати відеофрагменти, змінювати параметри зображення, поєднувати кілька джерел та експортувати готові матеріали в різних форматах. Одним із найвідоміших прикладів такого типу програм є Adobe Premiere Pro [3]. Цей редактор широко використовується в кіноіндустрії, телебаченні та відеопродакшн-студіях. Він підтримує багатодоріжковий монтаж, має розвинуті засоби для корекції кольору, накладання ефектів і роботи зі звуком (рис. 1.1). Також Adobe Premiere дозволяє вручну налаштовувати технічні параметри кожного відеофайлу перед їх склеюванням, що є перевагою при роботі з різнорідними фрагментами.

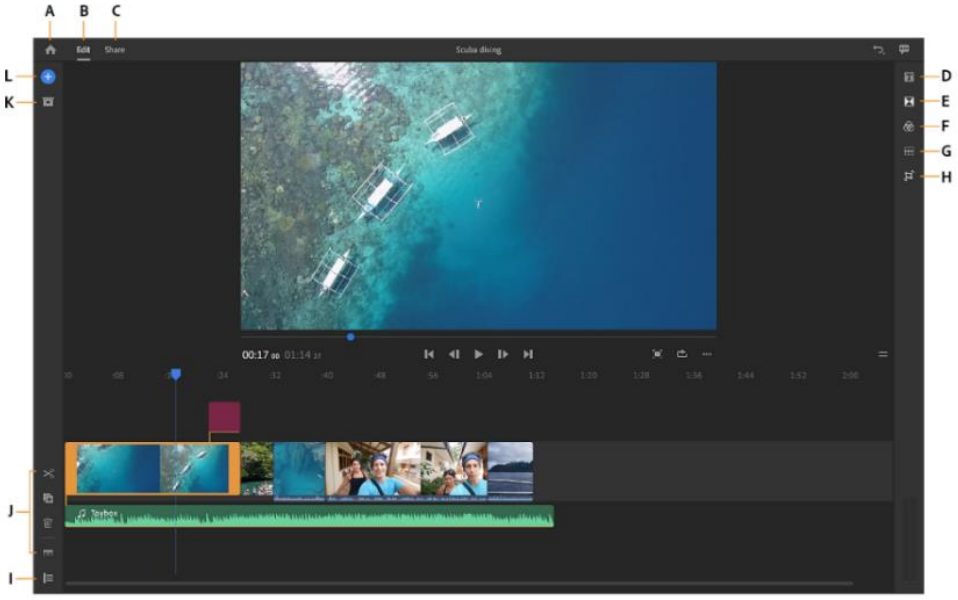

Рисунок 1.1 – Інтерфейс Adobe Premiere Pro [3]

Іншим популярним інструментом є DaVinci Resolve [4]. Він поєднує в собі можливості професійного відеомонтажу, високоякісної кольорокорекції та роботи зі звуком, що робить його одним із найпотужніших рішень. Програма має зрозумілий інтерфейс, що дозволяє працювати як початківцям, так і досвідченим користувачам. Однією з ключових переваг DaVinci Resolve є наявність безкоштовної версії, яка підтримує більшість основних функцій редагування і дозволяє створювати проєкти професійного рівня без фінансових витрат. Крім того, інструмент підтримує роботу з відеофайлами у високій роздільній здатності, і дозволяє ефективно обробляти гетерогенні видеофрагменту (рис. 1.2). Проте DaVinci Resolve потребує досить потужного комп’ютера з сучасною відеокартою та певного рівня навичок для ефективного використання всіх його можливостей. Для новачків це може стати певним бар’єром на початку роботи.

Рисунок 1.2 – DaVinci Resolve вікно монтажу з кількома відео й аудіодоріжками [4]

Також можлвио розглянути Shotcut як альтернативу [5]. Це безкоштовний відеоредактор з відкритим кодом, який можна використовувати на різних операційних системах. Shotcut підтримує базові операції: склеювання, обрізання, зміну розміру кадру, регулювання гучності

тощо. Його інтерфейс простіший у порівнянні з Adobe чи DaVinci, що робить його доступним для новачків (рис. 1.3). Проте функціональні можливості є обмеженими, а деякі завдання, пов’язані з обробкою відео з різними параметрами, доводиться виконувати вручну.

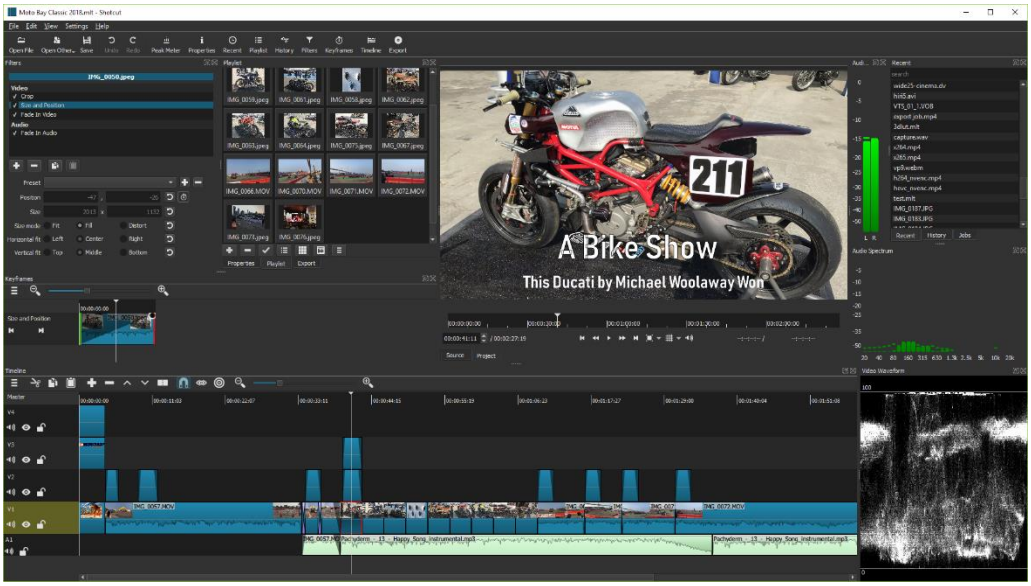

Рисунок 1.3 – Інтерфейс Shotcut [5]

Загалом, професійні редактори відео демонструють вражаючі можливості, але мають і ряд суттєвих обмежень. Вони потребують встановлення, налаштування, достатньо часу на опрацювання та знань для ефективного використання. Тому, незважаючи на їх потужність, подібні інструменти не завжди підходять для швидких чи одноразових задач об'єднання відео, особливо якщо користувач не має досвіду у відеомонтажі.

Окрему категорію засобів для обробки відео становлять інструменти, що працюють у середовищі командного рядка, або CLI (Command-Line Interface) (рис. 1.4). Вони зазвичай не мають графічного інтерфейсу та керуються шляхом введення текстових команд. Попри зовнішню складність, ці інструменти є надзвичайно гнучкими, швидкими та точними у виконанні окремих операцій над відеофайлами.

Найвідомішим представником цього класу є ffmpeg [6]. Потужний фреймворк з відкритим кодом, який дозволяє виконувати майже всі можливі

маніпуляції з відео: склеювання, конвертацію, масштабування, витягування аудіо, зміни частоти кадрів тощо. ffmpeg активно використовується як в індивідуальних проєктах, так і в масштабних медіаплатформах.

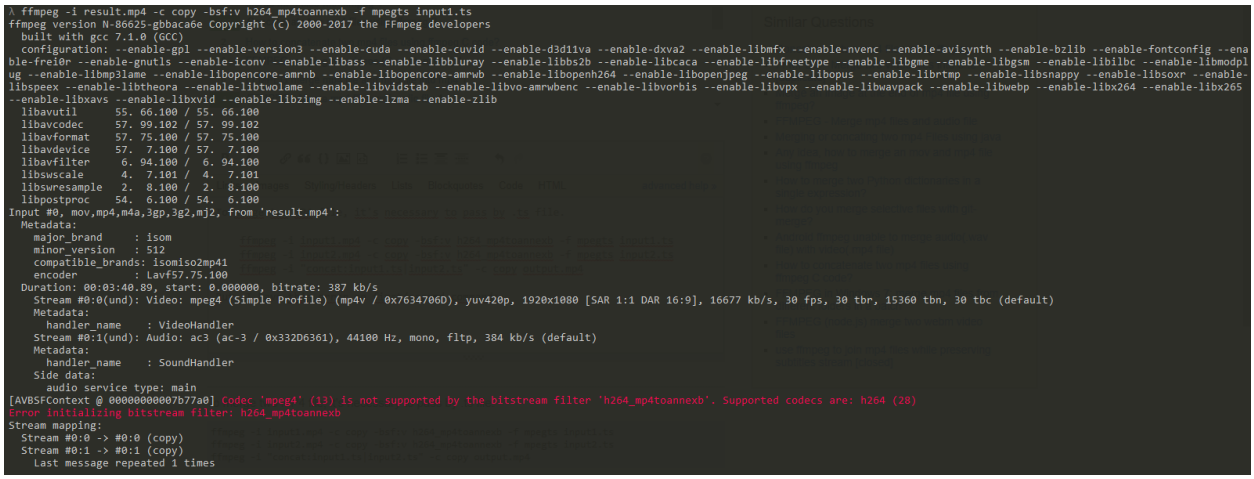

Рисунок 1.4 – Приклад введення команди Ffmpeg для склеювання двох відеофрагментів [6]

Одна з ключових переваг ffmpeg його універсальність. Інструмент підтримує величезну кількість форматів і кодеків, а також дозволяє користувачеві точно налаштувати вихідні параметри відео: роздільну здатність, частоту кадрів, співвідношення сторін тощо. Завдяки цьому ffmpeg є ідеальним рішенням для обробки відео з різними характеристиками.

Водночас, робота з ffmpeg має низку суттєвих недоліків, особливо для користувачів без технічного досвіду. Потрібно знати синтаксис команд, розуміти, як працюють відеопотоки, мати уявлення про кодеки, формати, аудіо й відеодоріжки. Будь-яка помилка у параметрах може призвести до невдалого результату або помилки під час виконання. Крім того, ffmpeg не забезпечує візуального зворотного зв’язку під час роботи. Користувач не бачить, як саме виглядатиме відео після обробки, поки воно не буде повністю згенероване. Це ускладнює процес для тих, хто звик до інтерактивного редагування.

У підсумку, інструменти командного рядка це потужне та точне рішення для автоматизованої обробки відео, особливо у випадках, коли потрібно виконати складну трансформацію чи пакетну обробку великої кількості файлів. Однак високий поріг входу та відсутність візуального інтерфейсу обмежують їхню доступність для широкого кола користувачів, які не мають спеціальних технічних знань.

З розвитком вебтехнологій усе більшої популярності набувають онлайн-сервіси, які дозволяють виконувати базові операції з відео прямо в браузері без встановлення програмного забезпечення та без складних налаштувань. Подібні інструменти орієнтовані переважно на звичайного користувача, який хоче швидко та зручно об’єднати кілька відеофрагментів в один файл.

Серед найбільш відомих сервісів можна виділити Clideo, Kapwing та VEED.io (рис. 1.5). Усі вони надають можливість завантажити відеофайли, розмістити їх у потрібному порядку, додати або видалити звук, обрізати зайві частини, а потім зберегти об’єднаний ролик у зручному форматі. Інтерфейси подібних сервісів зазвичай інтуїтивно зрозумілі, з елементами «drag and drop» та підказками для користувачів.

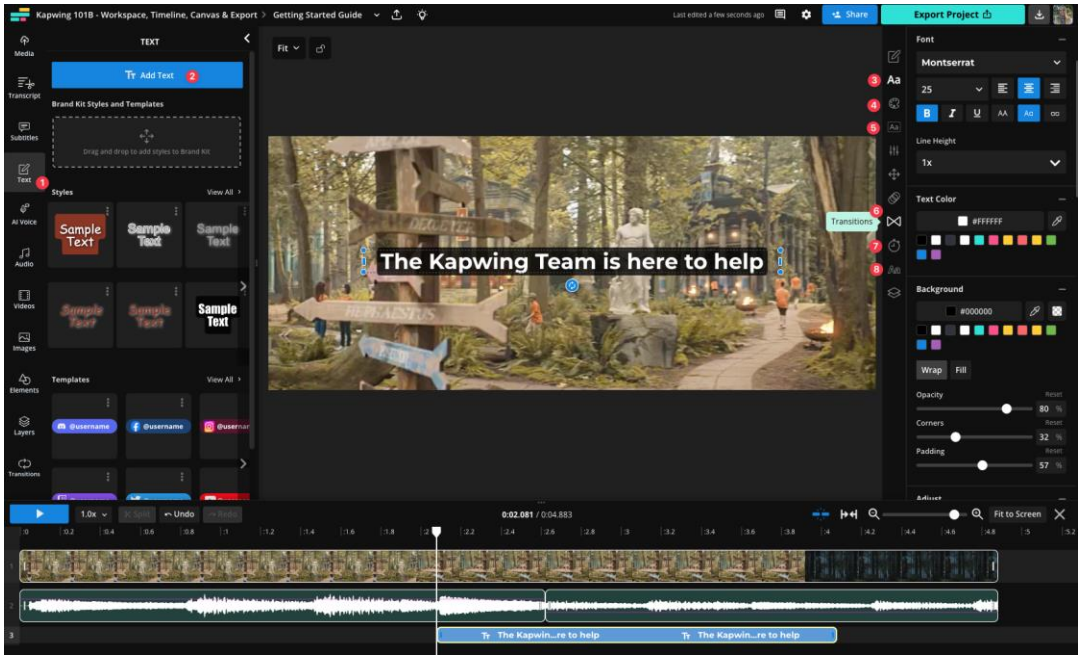

Рисунок 1.5 – Головна сторінка Kapwing [7]

Проте, попри зручність, онлайн-редактори мають і ряд обмежень, які варто враховувати. По-перше, обмеження за розміром файлів: безкоштовні версії часто не дозволяють завантажити великі відео або скорочують якість під час експорту. По-друге, наявність водяних знаків у підсумковому відео, якщо користувач не оформив підписку. Крім того, у більшості випадків доступна лише обмежена кількість форматів збереження, а технічні параметри обробки (роздільна здатність, частота кадрів, співвідношення сторін) змінюються автоматично, без детального контролю. Попри це, онлайн-сервіси залишаються важливою частиною екосистеми інструментів для роботи з відео. Вони ідеально підходять для простих і швидких задач, коли не потрібна глибока технічна обробка або робота з великою кількістю різноформатних фрагментів.

У випадках, коли користувач має справу з різними параметрами відеофрагментів, такими як роздільна здатність, орієнтація кадру чи співвідношення сторін, можливості вебсервісів можуть виявитися недостатніми. Це зумовлює потребу у створенні рішень, які поєднують зручність інтерфейсу з більшою гнучкістю у налаштуваннях і стабільністю в обробці нестандартних відеофайлів.

Для узагальнення порівняльного аналізу різних підходів до об’єднання відео доцільно розглянути ключові характеристики обраних інструментів у вигляді таблиці. Це дозволяє наочно оцінити рівень зручності, функціональності, доступності та технічних можливостей кожного рішення.

Таблиця 1.1 – Порівняння інструментів для обробки відео за перевагами та недоліками

Інструмент	**Переваги**	**Недоліки**
Adobe Premier Pro / Shotcut / DaVinci Resolve	- Потужний і професійний редактор - Підтримує всі формати	-Платний - Важкий для новачків - Потрібен потужний ПК
Clideo, Kapwing та VEED.io [7] [8] [9]	-Не треба нічого встановлювати - Простий інтерфейс - Швидкий старт	- Обмеження на розмір файлів - Не підходить для складних завдань - Бувають водяні знаки
ffmpeg	- Ідеально підходить для автоматизації - Працює з будь яками відео та їх форматами	- Важко розібратись без досвіду - Немає графічного інтерфейсу

З таблиці видно, що кожна категорія інструментів має свої сильні й слабкі сторони. Професійні редактори, як-от Adobe Premiere чи DaVinci Resolve, надають розширені можливості, але вимагають високого рівня підготовки та ресурсів. Командні інструменти, зокрема Ffmpeg, демонструють гнучкість і універсальність, однак складні для новачків. Онлайн-сервіси зручні у використанні, але мають функціональні обмеження та часто не здатні обробляти нестандартні відеофайли належним чином.

Все це створює підґрунтя для формування нових вимог до сучасних рішень, які поєднували б простоту використання з технічною адаптивністю.

У процесі об’єднання відеофрагментів з різними технічними характеристиками, роздільною здатністю, форматом, орієнтацією кадру, співвідношенням сторін чи частотою кадрів, виникає потреба в узгодженні цих параметрів для отримання стабільного відеофайлу. На практиці існує кілька основних моделей склейки, які можуть бути реалізовані в програмних рішеннях [10]:

– масштабування (resize). Усі відеофайли приводяться до однакового розміру кадру. Це дозволяє уникнути розривів на межі фрагментів, але може спричинити спотворення зображення, якщо вихідне співвідношення сторін було різним. Такий підхід активно використовується у більшості редакторів при ручному монтажі;

– додавання полів (padding). Для збереження співвідношення сторін без спотворення відео до нього додаються чорні смуги по краях (збоку або зверху/знизу), що дозволяє рівномірно вписати всі фрагменти в єдиний кадр. Цей метод широко застосовується у телевізійній трансляції та підготовці відео для соцмереж;

– обрізання кадру (cropping). Частина зображення відсікається, щоб привести фрагмент до потрібного формату. Такий варіант дозволяє зберегти формат вихідного ролика, але може втратити важливу частину зображення;

– вирівнювання за основним відео (master format). Один із відеофрагментів приймається як основа, до параметрів якого вирівнюються всі інші фрагменти. Це стосується не лише розміру, а й частоти кадрів, кодека, глибини кольору та формату звуку. Такий підхід дозволяє забезпечити цілісність структури фінального файлу;

– автоматичне узгодження параметрів (auto-normalization). Деякі інструменти здатні автоматично узгоджувати різні технічні характеристики відео під час склейки визначючи компроміс між якістю та відповідністю.

Застосування тієї чи іншої моделі залежить від мети, цільової аудиторії, формату платформи розміщення та технічних можливостей інструменту. У сучасних рішеннях, що орієнтовані на масового користувача, доцільно реалізовувати автоматичне визначення моделі або надавати користувачу можливість вибору серед кількох варіантів.

1.3 Вимоги до інструментів для склеювання відео з різними параметрами

З огляду на розглянуті у попередньому підрозділі приклади, можна сформувати перелік ключових вимог, яким має відповідати сучасний інструмент для об’єднання відеофрагментів, що мають відмінні технічні характеристики. У контексті активного використання відео у різних сферах від освіти до бізнесу потреба у таких рішеннях стає все більш актуальною.

Однією з основних вимог є підтримка відео з різними параметрами, зокрема відмінною роздільною здатністю, співвідношенням сторін, частотою кадрів, орієнтацією кадру та способом кодування. Інструмент має коректно обробляти ситуації, коли один фрагмент має горизонтальний формат, а інший вертикальний, або коли один файл коротший, інший довший, тощо. Важливо, щоб різнорідність вихідних матеріалів не призводила до помилок або втрати якості під час об’єднання.

Другою вимогою є автоматичне приведення всіх відеофрагментів до єдиного узгодженого формату. Це включає узгодження розміру кадру, типу відеопотоку, параметрів звуку та іншого. Такий підхід дозволяє користувачеві зосередитись на змісті, а не на технічних нюансах. Ідеальний інструмент має забезпечувати цей процес у фоновому режимі, без необхідності вручного втручання. Ще однією важливою властивістю є простота використання. Інтерфейс має бути інтуїтивно зрозумілим, а сам процес зведеним до кількох простих кроків: завантаження відео, визначення порядку фрагментів, запуск обробки та отримання готового результату. Відсутність потреби в попередній підготовці або технічній інструкції суттєва перевага для широкого кола користувачів.

Крім того, важливо забезпечити стабільність роботи інструменту, особливо при обробці великих файлів або значної кількості фрагментів. Навіть у разі завантаження важкого відео система не повинна зависати або завершувати обробку з помилкою. Надійність і передбачуваність результату

один із базових критеріїв якісного програмного рішення. Сучасне середовище використання цифрових інструментів також диктує потребу в платформеній незалежності. Оптимальним варіантом є вебзастосунок, який працює у браузері та не потребує встановлення. Це розширює коло потенційних користувачів від тих, хто працює з комп'ютера, до власників мобільних пристроїв або планшетів. Підтримка популярних браузерів і стабільна робота на різних операційних системах є важливою технічною вимогою.

Окремо варто зазначити сумісність із найпоширенішими форматами відео. Застосунок повинен підтримувати імпорт і експорт файлів у форматах, які найчастіше використовуються: MP4, WebM, AVI, MOV тощо. Це забезпечує гнучкість і зручність для кінцевого користувача.

1.4 Постановка задачі

Основна мета це створити простий і зручний вебзастосунок, який дозволяє об'єднувати різні відео в одне, навіть якщо вони мають різну якість, розмір, формат чи орієнтацію. Користувач зможе працювати з відео без спеціальних знань, а сам застосунок має стабільно працювати з файлами з різних джерел.

Об'єктом роботи є створення вебсайту для обробки та подальшого об'єднання відеофайлів з різними параметрами.

Метою роботи є розробка програмного забезпечення, яке дозволяє отримувати повну інформацію про відеофрагменти, приводити їх до спільного формату, не втрачаючи якості відео, та об'єднувати їх разом.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз інструментів, які використовуються для склейки відео;

– виявити переваги та недоліки у роботі цих інструментів;

– обґрунтувати вибір інструментів для реалізації клієнтської та серверної частин;

– розробити користувацький інтерфейс для завантаження відео, перегляду технічної інформації та запуску обробки;

– реалізувати серверну логіку, що забезпечує аналіз вхідних файлів, вирівнювання їх параметрів та об’єднання в один відеофайл;

– протестувати розроблене рішення на прикладах з реальними відео.

2 ОБҐРУНТУВАННЯ ПРОГРАМНИХ РІШЕНЬ ДЛЯ СТВОРЕННЯ ЗАСТОСУНКУ ЗІ СКЛЕЙКИ ГЕТЕРОГЕННИХ ВІДЕО

2.1. Вибір архітектурного підходу

Системи, що працюють з відео різних параметрів, мають бути простими у використанні, стабільними та зручними для людей без досвіду в монтажі. Цільова аудиторія таких рішень це звичайні користувачі, які мають отримати швидкий і передбачуваний результат без необхідності в технічному налаштування. Особлива увага має приділятися зручності інтерфейсу, зведенню кількості дій до мінімуму та автоматичній обробці на всіх етапах: від завантаження до отримання готового відео.

Вебзасосунок є найбільш універсальним варіантом. Він не вимагає встановлення додаткових програм, не залежить від конкретної операційної системи, і забезпечує однакову доступність з ноутбука, смартфона чи планшета. Такий підхід особливо зручний у випадках, коли користувач має справу з кількома короткими відео або записами з різних пристроїв, які потрібно швидко об’єднати без втрати якості.

Сам браузер має технічні обмеження. При обробці великих відеофрагментів виникають складнощі, пов’язані з обсягом оперативної пам’яті, продуктивністю процесора та стабільністю роботи. Затримки в інтерфейсі, зависання або повна відмова обробки є наслідоком спроби виконати операції на клієнтському боці. Тому найкращим рішенням є перенесення важкої на серверну частину. Це дозволяє забезпечити однакову якість обробки незалежно від пристрою користувача, а також краще контролювати помилки та обробляти виняткові ситуації.

Ще одним важливим елементом є автоматизація. Ідеальний сценарій використання виглядає так: користувач обирає відеофайли, система самостійно визначає їх параметри, адаптує до єдиного стандарту, об’єднує та

повертає готовий результат. Це знижує бар’єр входу і значно покращує користувацький досвід, оскільки дозволяє зосередитися не на технічних аспектах, а на кінцевій меті в отриманні готового відео.

Не менш важливою є стійкість системи до помилок і нестандартних вхідних даних. У реальних умовах користувачі можуть завантажувати пошкоджені, обрізані, записані з мобільного або навіть горизонтально перевернуті відео. У таких випадках система має не завершувати роботу з помилкою, а коректно обробляти ситуацію або, щонайменше, зрозуміло повідомляти, що саме не так. Це сприяє підвищенню довіри до сервісу та зменшує ймовірність виникнення негативного користувацького досвіду.

Враховуючи вищезгадані вимоги, найоптимальнішим є використання класичної клієнт-серверної архітектури. Вона дозволяє чітко розділити відповідальність: клієнт відповідає за інтерфейс та взаємодію з користувачем, сервер відповідає за обробку, нормалізацію та збереження даних. У такій моделі застосунок реалізується як SPA (Single Page Application) [11], що значно прискорює інтерфейс і мінімізує обсяг трафіку. Усі взаємодії з сервером здійснюються через API, що дозволяє ефективно обробляти запити й отримувати результат у вигляді вже готового з’єднаного відео.

Серверна частина не лише обробляє відео, а й зберігає файли в тимчасовому сховищі, керує чергами та повертає посилання на результат. У разі зростання навантаження така архітектура може бути легко адаптована до хмарних сервісів або розділена на окремі компоненти, що спрощує масштабування.

Поєднання простого інтерфейсу з ефективною серверною обробкою створює збалансовану систему, яка одночасно є доступною для широкого кола користувачів і достатньо гнучкою для роботи з гетерогенними відеофрагментами.

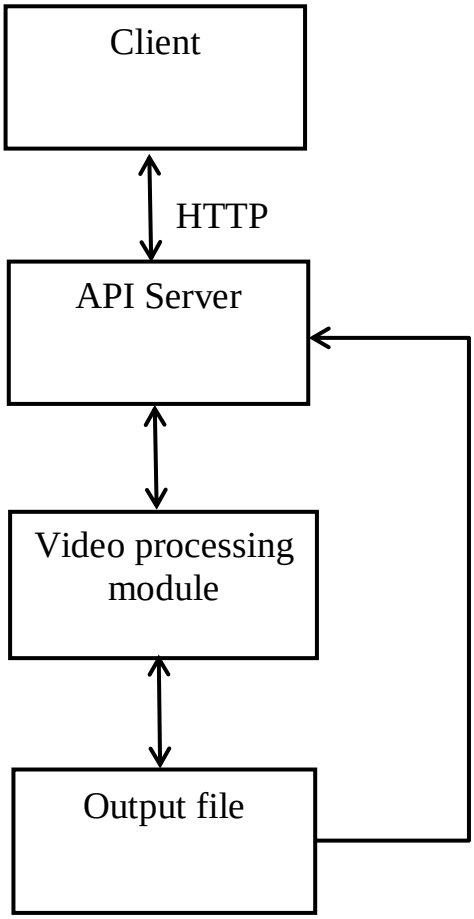

Рисунок 2.1 – Загальна схема взаємодії компонентів

Користувач завантажує відеофрагменти, які зберігаються на сервері. Далі запускається обробка: система нормалізує відео та формує підсумковий файл. Після завершення користувач отримує посилання або прямий доступ до результату.

Якщо файл буде пошкоджений або нестандартний, система має повернути зрозуміле повідомлення про помилку. Це покращує зручність використання та допомагає уникнути розчарування у користувача.

У випадках, коли одночасно надходить кілька запитів на обробку відео, система формує чергу виконання. Кожен запит додається до черги автоматично після завантаження файлів. Користувач отримує унікальний ідентифікатор сесії, за яким може відстежувати статус обробки: «в очікуванні», «виконується», «завершено» або «помилка».

Для забезпечення зворотного зв’язку використовується механізм періодичних запитів до API або WebSocket підключення, якщо воно підтримується клієнтом. Завдяки цьому користувач може бачити поточний стан обробки в режимі реального часу, не перезавантажуючи сторінку. У випадку збою, система повертає опис помилки, що дозволяє швидко зреагувати або повторити операцію.

Розглядалися й інші архітектурні варіанти. Наприклад, десктопний застосунок дає більше контролю, але потребує встановлення, оновлень і працює лише на конкретних операційних системах. Це не зручно.

Також тестувався варіант з обробкою у браузері через WebAssembly-версії ffmpeg. Але у реальних умовах такий підхід дає слабку стабільність, не справляється з великими файлами й має технічні обмеження.

Порівняно з цим, клієнт-серверна модель дозволяє балансувати навантаження, забезпечувати стабільність і залишатися простою для користувача. Вона підходить для вебзастосунку, який повинен безпомилково працювати з відео будь-якого формату і походження.

2.2. Визначення вимог до обробки відео

Серед ключових характеристик, які потребують обов’язкового узгодження перед об’єднанням відеофайлів, особливо важливими є роздільна здатність зображення, частота кадрів, співвідношення сторін, орієнтація відео, формат кодування та тип контейнера, в якому збережено файл. Як показано у [12], послідовна сегментація відео може ефективно виконуватися через просторовий поділ кадрів, що узгоджується з підходами до нормалізації в даному застосунку [13]. Система має розуміти, з яким саме матеріалом вона працює, та що з ним потрібно зробити, аби всі фрагменти взаємодіяли без конфліктів.

Система передбачає ряд механізмів для роботи з нестандартними або частково пошкодженими відеофрагментами.

У випадку, коли файл не містить відеодоріжки (лише аудіо), він автоматично виключається з обробки, а користувач отримує відповідне попередження.

Якщо контейнер відео пошкоджений або не зчитується, система не завершує роботу з помилкою, а переходить до обробки інших фрагментів, фіксуючи інцидент у логах.

При об’єднанні фрагментів різної тривалості коротші частини автоматично доповнюються порожнім відео або тишею, щоб синхронізувати тривалість усіх сегментів.

Такі виняткові ситуації не блокують виконання всієї задачі, а обробляються окремо, що підвищує стабільність системи і запобігає втраті прогресу через один помилковий файл.

Роздільна здатність відео визначає фізичний розмір кадру скільки пікселів по ширині та висоті матиме зображення. Якщо в одному фрагменті 1920×1080, а в іншому 1280×720, то без попереднього масштабування або вирівнювання зображення виглядатиме рвано: з розривами або різкою зміною чіткості. Це візуально дуже відчутно, навіть якщо глядач не має технічної підготовки. Особливо це стає помітним при переходах між фрагментами одне відео може виглядати «розтягнутим» або навпаки зменшеним у центрі чорного фону.

Не менш важливо враховувати частоту кадрів. Це технічний параметр, який показує, скільки кадрів відображається на екрані щосекунди. Якщо об’єднати відео з різною частотою можуть виникнути візуальні ривки, несинхронність руху або навіть неплавність зображення. У деяких випадках, якщо відео має нестандартну частоту (наприклад, 23.976 або 59.94 fps), то без нормалізації система обробки може неправильно оцінити таймінг або спробує «дотягнути» ці фрагменти до найближчого стандартного значення, що призведе до втрати точності синхронізації.

Співвідношення сторін це ще один важливий аспект. Відео може бути як стандартного формату 16:9, так і вертикального 9:16 або навіть квадратного (1:1). При об’єднанні таких фрагментів без попередньої адаптації часто виникають візуальні спотворення: одне з відео обрізається, інше розтягується або отримує чорні поля з боків. У випадку автоматичної обробки система повинна вміти або вирівнювати всі відео до одного співвідношення сторін, або правильно масштабувати та позиціонувати фрагменти, зберігаючи їхню геометрію.

Орієнтація кадру також впливає на кінцевий результат. Відео може бути зняте в портретному або ландшафтному режимі, і без правильного вирівнювання одне з них буде відображатися боком або з викривленням. Особливо це актуально для відео з мобільних пристроїв, де користувачі часто змінюють положення телефона під час зйомки. У такому випадку система повинна виявити реальну орієнтацію та привести всі фрагменти до єдиної логіки, вертикальної або горизонтальної.

Також, важливо сказати про формат кодування це набір технічних інструкцій, які визначають, як саме зображення та звук зберігаються у файлі. Якщо частина відео закодована за допомогою H.264, а інша VP9 або H.265, то система повинна або перекодувати всі фрагменти до одного формату, або забезпечити правильне зчитування та обробку під час злиття. Аналогічно із типом контейнера: файли у форматі MOV, MP4, AVI, WebM можуть зберігати інформацію по-різному, і без попереднього узгодження можуть виникнути конфлікти наприклад, відсутність звуку, неправильна тривалість або невідповідність між відео і аудіо матеріалами.

Усі ці технічні відмінності визначають якість і стабільність фінального результату. Саме тому попередня нормалізація це необхідний крок, що дозволяє перетворити набір фрагментів на єдиний, цілісний відеофайл. Наприклад також є можливість використовувати норму Кі Фана для обробки відеофрагментів, що дозволяє досягти більш плавних переходів між частинами із різними параметрами [14].

Окремо варто згадати про проблеми зі звуком. У деяких випадках аудіо може зникнути повністю після об’єднання, або бути розсинхронізованим з відеорядом. Це відбувається через різну частоту дискретизації, кількість каналів або відмінності у таймінгах звукових доріжок. Особливо складно в таких випадках стає виявити помилку на око, адже все здається «на місці» допоки не починаєш переглядати відео уважно або не стикаєшся з відмовою його відтворення.

З практики можна навести приклад, коли користувач намагається поєднати відео з GoPro (60 fps, 2.7K роздільність) з фрагментами, знятими на смартфон у вертикальному форматі (30 fps, 1080×1920). У результаті, без належної обробки, один з фрагментів втрачає чіткість, інший обрізається, а звук відстає на кілька секунд. Такі «дрібниці» легко перетворюють ідею зручного автоматичного монтажу на довгу ручну роботу з виправлення помилок.

Із цього випливає очевидний висновок: навіть при зовнішній схожості відеофайлів вони можуть бути технічно несумісними. А будь-яка система, яка претендує на автоматизацію склеювання, повинна передусім вирішувати ці проблеми ще до того, як користувач побачить результат.

Попри наявність загального переліку технічних параметрів, які потребують вирівнювання перед склеюванням відео, сам підхід до цієї обробки не є універсальним. У реальній практиці не існує одного «правильного» способу нормалізації, який би однаково добре працював у кожній ситуації. Система повинна не просто застосовувати заздалегідь визначений алгоритм, а вміти адаптуватися до конкретного набору вхідних файлів і умов їх поєднання.

На цьому етапі особливо важливо враховувати не лише технічні параметри, а й логіку фінального вигляду відео: наприклад, чи є основний фрагмент, до якого мають підлаштовуватися інші; чи очікує користувач збереження точних пропорцій, чи допускає обрізання; чи прийнятне використання полів або автоматичне масштабування.

Усе це формує своєрідну «динамічну модель нормалізації», коли система, аналізуючи вихідні матеріали, самостійно обирає найдоцільніший сценарій вирівнювання. Такий підхід може враховувати як типові стратегії (масштабування, обрізання, додавання полів тощо), так і більш гнучкі моделі, де рішення базується на комбінованих ознаках: від співвідношення сторін до положення об’єктів у кадрі.

Таким чином, уніфікація не повинна сприйматись як жорстка послідовність дій, а радше як адаптивний процес. Чим краще система «розуміє» контекст, тим менш помітним буде втручання обробки у візуальну структуру відео. Саме це і є основною вимогою: обробити не змінюючи сенсу, поєднати не зруйнувавши цілісність.

Успішне поєднання різнорідних відеофрагментів це не просто технічна задача, як може здаватися спочатку. Насправді тут важливо, щоби відео не виглядали зібраними нашвидкуруч, без системи. Бо інакше, навіть при нормальних технічних параметрах, усе одно буде видно різницю у чіткості, кольорах або рухах.

Якщо система справді нормалізує відео як треба, то кінцевий ролик повинен виглядати як один єдиний файл, ніби його монтував вручну досвідчений монтажер. Без того, щоб якась частина раптом ставала ширшою, інша розтягнутою, а ще якась темнішою або з дивними звуками. Це звучить просто, але на практиці, без попередньої обробки, такі дрібниці дуже псують загальне враження.

Нормалізація це щось типу невидимого монтажу: коли вона зроблена добре, то її взагалі не помітно. І це, мабуть, головна мета: не просто склеїти, а зробити це так, щоб усе виглядало як одне ціле, без зайвих питань чи потреби щось виправляти вручну потім.

Також важливо, щоб не виникало додаткових артефактів після обробки: смуг, які з’являються через неправильне вирівнювання, зміщення аудіо, надмірної компресії або втрати чіткості.

Користувач, переглядаючи фінальний ролик, не повинен помічати, де саме змінювався формат, частота кадрів чи співвідношення сторін. Якщо ж такі переходи стають очевидними значить, процес нормалізації не був достатньо точним або не адаптувався до вихідних даних.

Таким чином, успішна обробка це не лише відповідність певним технічним критеріям, а й досягнення візуальної та звукової цілісності. Саме на це повинна бути орієнтована вся система не на формальне об'єднання файлів, а на створення природного, «живого» результату, який не потребує додаткового редагування вручну.

Щоб виконати описані вище вимоги до уніфікації відеофрагментів, у структурі серверної частини системи передбачено окремий модуль попередньої обробки відео (VideoProcessingModule). Його завдання полягає у підготовці всіх вхідних файлів до об'єднання: перевірці технічних параметрів, узгодженні ключових характеристик і забезпеченні сумісності між фрагментами.

Після завантаження файлів через API, система не переходить одразу до склеювання. Кожен відеофрагмент спочатку проходить аналіз: модуль визначає роздільну здатність, частоту кадрів, орієнтацію, співвідношення сторін, кодеки, тип контейнера та інші технічні деталі. Усе це співставляється з внутрішніми вимогами до фінального результату.

Далі модуль приймає рішення: чи можна з'єднати ці фрагменти без додаткових змін, чи потрібно провести нормалізацію. Якщо всі параметри вже узгоджені, система переходить до злиття одразу. Але якщо є розбіжності то запускається процес вирівнювання. Для цього використовується один із доступних підходів: масштабування, обрізання кадру або додавання полів. Вибір не базується лише на технічних умовах, враховується і загальна логіка композиції, наприклад, система може уникати обрізання, якщо в кадрі розпізнаються важливі об'єкти.

Трансформації виконуються як послідовні кроки всередині модуля. Це дозволяє уникати помилок, працювати з тимчасовими файлами, зберігати

оброблені фрагменти в кеш і не повторювати операції, які вже були виконані. Після завершення всі нормалізовані відео зберігаються окремо й передаються на фінальне склеювання.

Логіку роботи модуля зображено на рисунку 2.2, який демонструє послідовність етапів обробки. Окремо показано гілку оптимізованої обробки вона активується, коли всі параметри вже сумісні. Це допомагає мінімізувати навантаження на сервер і уникнути непотрібної перекодировки.

Завдяки такій структурі VideoProcessingModule вирішує ключову задачу забезпечує цілісність, стабільність і передбачуваний результат незалежно від того, на яких пристроях і з якими налаштуваннями були створені фрагменти. Саме він є технічною основою для реалізації головної ідеї проєкту автоматичного та зручного об’єднання гетерогенних відео в єдиний файл без зайвих зусиль з боку користувача.

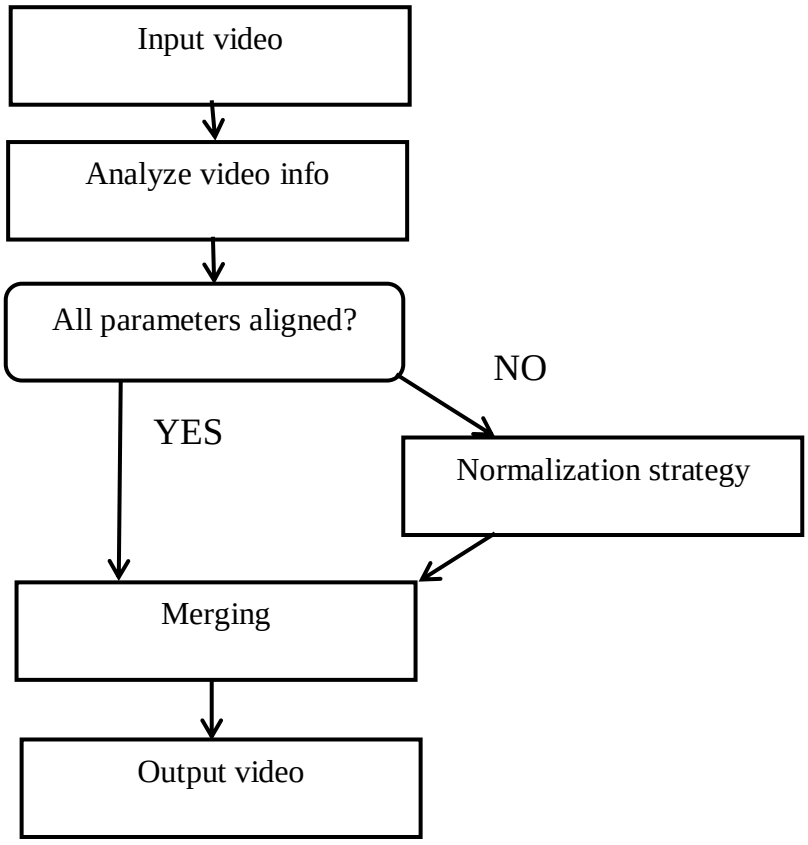

Рисунок 2.2 – Логіка роботи модуля попередньої обробки відео (VideoProcessingModule)

2.3. Формування вимог до формату вихідного відео

У процесі автоматичної обробки та склеювання відеофрагментів не менш важливим за нормалізацію вхідних даних є чітке визначення характеристик фінального результату. Якщо на виході користувач отримує відео, яке відтворюється з помилками, не відкривається на мобільному пристрої або має надто великий розмір, це повністю нівелює зусилля системи на попередніх етапах. Тому ще на етапі проєктування слід закласти вимоги до формату вихідного файлу, які забезпечують зручність перегляду, стабільність і сумісність з основними платформами. Попереднє виявлення змін у фрагментах дозволяє адаптивно налаштовувати обробку та уникати різких переходів у згенерованому відео [15].

На відміну від традиційного відеоредагування, де користувач сам обирає параметри експорту, у розроблюваній системі ці рішення приймаються автоматично. Це накладає додаткову відповідальність: обрані параметри повинні підходити для більшості користувачів без потреби додаткового конвертування чи ручного втручання. Формат вихідного відео має бути універсальним, легким для розповсюдження, придатним для завантаження в соціальні мережі та максимально незалежним від операційної системи або типу пристрою.

Важливим кроком у формуванні вихідного відео є визначення типу файлу, який буде збережено після обробки. Оскільки користувач очікує, що результат можна буде одразу переглянути, завантажити або надіслати, система повинна обирати формат, який гарантує максимально широкий спектр підтримки без додаткових конвертацій.

Таблиця 2.1 – Порівняльна характеристика форматів відеофайлів за основними параметрами

Формат	**Переваги**	**Недоліки**	**Сфера використання**
MP4 (H.264 + AAC) [16]	Універсальний, підтримується всіма пристроями, баланс якості та розміру	Втратне стиснення, немає підтримки прозорості	Соцмережі, YouTube, мобільні пристрої
WebM (VP8/VP9) [17]	Відкритий формат, добра компресія	Обмежена підтримка (особливо на Apple пристроях)	Вебсайти
MOV (Apple) [18]	Висока якість, підтримка альфа-каналу	Великий розмір, обмежена сумісність	Відеомонтаж (Final Cut), Mac
AVI [19]	Підтримує багато кодеків, мінімальне стиснення	Дуже великий розмір, застарілий	Архівне зберігання, спецзавдання
MKV [20]	Гнучкість, підтримка субтитрів, декількох аудіодоріжок	Не всі плеєри підтримують	Торренти, фільми, неофіційні релізи
H.265 (HEVC) [21]	Вища ефективність стиснення	Потребує потужного декодування, менше підтримки	4K-відео, стрімінгові сервіси

Для цього як основний формат контейнера було обрано MP4 на сьогодні це один із найпоширеніших стандартів у відеоіндустрії. Він підтримується майже всіма сучасними браузерами, операційними системами та мобільними пристроями. Крім того, MP4 забезпечує оптимальне стиснення даних без помітної втрати якості, що особливо важливо при роботі з великими файлами.

Усередині контейнера використовуються добре перевірені рішення: відеокодек H.264 (AVC) та аудіокодек AAC. Вони забезпечують високу якість при помірному бітрейті, дозволяють уникнути артефактів при стисканні та не створюють труднощів під час відтворення. Більше того, ці кодеки апаратно підтримуються більшістю пристроїв, що позитивно впливає на плавність перегляду навіть на слабких системах.

Важливо підкреслити, що вибір цих форматів продиктований не лише технічною доцільністю, а й практичним досвідом взаємодії кінцевого користувача з відеофайлом. Інші варіанти, зокрема WebM, MOV або формати з новішими кодеками на кшталт H.265, були відкинуті через обмежену підтримку, більшу вагу, або складність у подальшій обробці. Обраний формат це компроміс між якістю, стабільністю та універсальністю.

Окрім формату збереження, важливим компонентом є якісні характеристики відео: його роздільна здатність і частота кадрів. Ці параметри безпосередньо впливають як на візуальне сприйняття фінального ролика, так і на його розмір, швидкість передачі, плавність та загальну сумісність.

У якості цільового стандарту була обрана роздільність 1920×1080 пікселів (Full HD). Такий формат залишається найпоширенішим серед споживчих пристроїв і забезпечує хорошу деталізацію без надмірного навантаження на систему. У той час як 4K або 2.7K можуть виглядати більш сучасно, вони вимагають значно більшого обсягу пам’яті й можуть створювати проблеми при перегляді на слабших пристроях або через обмежене інтернет-з’єднання. Full HD, натомість, зберігає баланс між якістю та практичністю.

У деяких випадках, якщо користувач надає відео вищої якості, допускається збереження вихідної роздільності для уникнення додаткової декомпресії. Водночас для більшості ситуацій система застосовує масштабування до єдиного формату саме для того, щоб забезпечити стабільність і передбачуваність результату незалежно від вихідних фрагментів.

Частота кадрів встановлюється на рівні 30 кадрів на секунду (FPS) це значення є оптимальним для більшості сценаріїв використання: від перегляду в браузері до публікації в соцмережах. Більшість відео, знятих на смартфони, вже мають саме таку частоту. У випадках, коли вихідні відео мають іншу частоту (24, 25 або 60 FPS), система виконує автоматичну адаптацію до базового значення з урахуванням плавності руху. Такий підхід дозволяє уникати ривків, розсинхронізації та зменшує ризики несумісності при подальшому відтворенні.

Таким чином, фіксація роздільності та FPS у рамках чітко визначених значень не є обмеженням це інструмент забезпечення стабільної якості в умовах повної автоматизації процесу.

Формуючи вихідне відео, слід враховувати не лише якісні показники, а й обсяг генерованого файлу. Це особливо актуально для системи, яка працює у браузерному середовищі, де користувач очікує швидкого завантаження результату без потреби використання сторонніх сервісів чи додаткового програмного забезпечення.

Одним з ключових параметрів, що впливають на обсяг, є бітрейт кількість даних, які система передає для кожної секунди відео. Надто високий бітрейт призводить до великих файлів, які займають багато часу на обробку та передачу. Натомість надто низький викликає втрату якості, артефакти, змазування та втрату деталізації.

Для відео у форматі 1080p оптимальним значенням вважається бітрейт у межах 5 Мбіт/с. Це дозволяє зберігати чіткість зображення, зокрема у динамічних сценах, і водночас забезпечує помірний розмір файлу, придатний для швидкого скачування або передачі через мережу. У випадку меншої роздільності, система автоматично знижує бітрейт відповідно до заданих правил.

Компресія, що використовується в модулі обробки, є втратною (lossy) тобто частина даних свідомо відкидається для зменшення обсягу. Це типовий підхід у відеообробці: він дозволяє зберігати якість на рівні, прийнятному

для візуального сприйняття, при значному зменшенні розміру. Формати .mp4 та H.264 забезпечують гнучкі механізми компресії, які дозволяють досягти доброго результату без візуально помітних втрат.

Окрім основних параметрів, система враховує й загальну довжину ролика: якщо відео триває понад 5 хвилин, застосовується адаптивне коригування наприклад, незначне зниження бітрейту при переважно статичних сценах. Це дозволяє уникнути зайвого перевантаження сервера, не впливаючи помітно на якість.

Загалом, контроль над бітрейтом та розміром відео дає змогу досягти стабільного користувацького досвіду, при якому результат завжди передбачувано відкривається, завантажується та зберігає чіткість навіть у складних технічних умовах.

Окрім основних технічних характеристик таких як формат, роздільна здатність чи бітрейт система також формує низку додаткових параметрів, які забезпечують коректну поведінку файлу після завершення обробки. Ці деталі здебільшого залишаються непомітними для користувача, проте без них фінальний відеофайл може працювати нестабільно або відображатися некоректно на певних платформах.

По-перше, система забезпечує єдину орієнтацію кадру горизонтальну (landscape), як стандарт для переважної більшості екранів. Якщо вхідні відео мають вертикальну орієнтацію, вона або трансформується, або адаптується з дотриманням правильного співвідношення сторін і без викривлень. Це дозволяє уникнути ситуацій, коли фінальний ролик виглядає фрагментарно або непропорційно.

Також формується набір метаданих, серед яких: тривалість, розмір кадру, кількість кадрів, інформація про кодек і аудіотрек. Це критично важливо, оскільки саме ці дані використовують програвачі для відображення шкали часу, оцінки буферу та налаштування відтворення. Система також автоматично виставляє коректне співвідношення сторін (aspect ratio) та синхронізує відео з аудіо, уникаючи зміщень або накладень.

Щодо звукової частини, всі доріжки зводяться до стереоформату (2 канали) з частотою дискретизації 44.1 або 48 кГц. Такий підхід гарантує сумісність із більшістю медіаплеєрів і забезпечує якість, достатню як для голосу, так і для музичного супроводу. Якщо в оригінальних фрагментах присутні багатоканальні аудіо або різні мови, система залишає лише основний потік, що додатково зменшує обсяг файлу без втрати ключового контенту.

Окремо слід зазначити, що у фінальному файлі не передбачено обов’язкових субтитрів, заставок чи водяних знаків. Це свідомий вибір система фокусується на збереженні змісту, а не на додаткових елементах, які можуть відволікати або конфліктувати з форматами соціальних платформ (рис. 2.3).

Для ілюстрації зазначеного підходу нижче наведено приклад кадру з відео у розширенні 2K до нормалізації та відповідний кадр після приведення до формату Full HD із збереженням пропорцій (додаванням чорних полів) (рис. 2.4). Зображення є власноруч створеними в рамках реалізації проєкту.

Рисунок 2.3 – Кадр до нормалізації (2K)

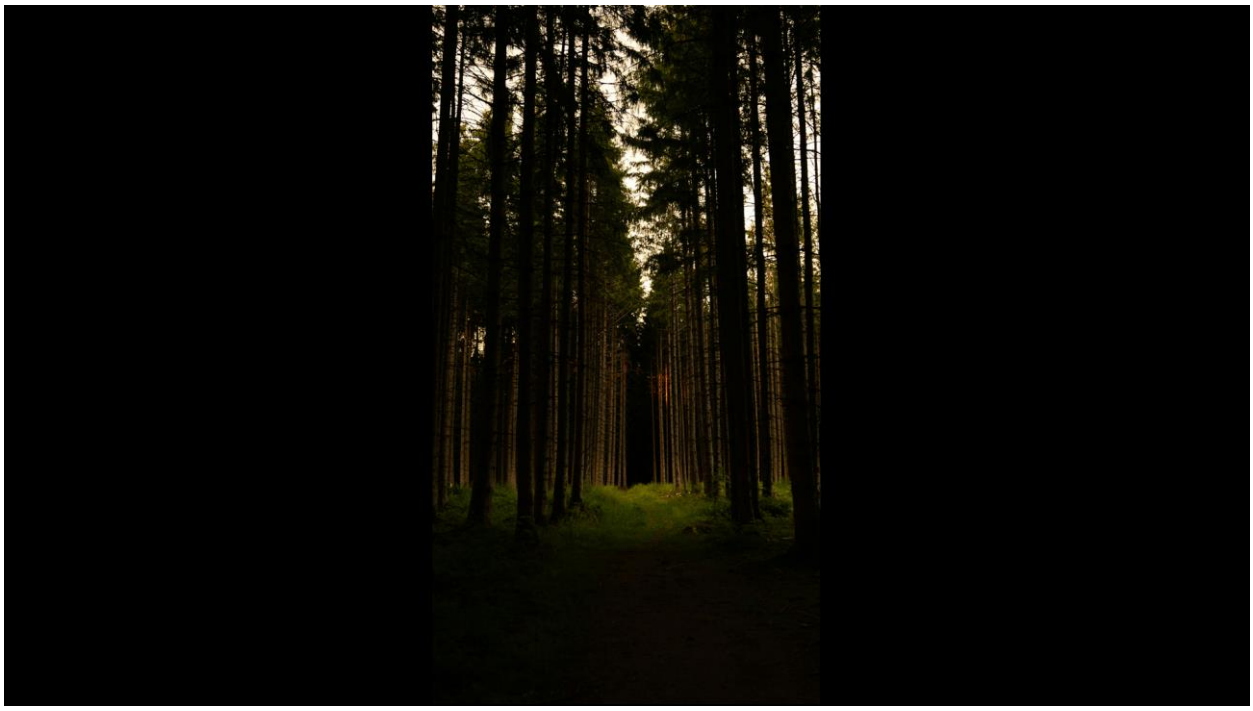

Рисунок 2.4 – Кадр після нормалізації до Full HD з додаванням чорних полів

У підсумку, всі встановлені вимоги до фінального відео спрямовані на досягнення головної мети створення стабільного, універсального та якісного результату, який не потребує жодних дій з боку користувача після завершення обробки. Завдяки цьому система виконує не лише технічну, а й функціональну роль: вона не просто поєднує відео, а забезпечує готовий до використання продукт, незалежно від вихідних умов зйомки, формату чи пристрою, на якому було створено фрагменти.

3 РЕАЛІЗАЦІЯ ПОСТАНОВКИ ЗАДАЧІ ТА ТЕСТУВАННЯ

3.1. Вибір інструментальних засобів для реалізації поставленої задачі

Для проєкту, особливо важливо підібрати такий технологічний стек, який забезпечить надійність, гнучкість, швидкість та простоту підтримки системи в майбутньому.

Під час розробки даного застосунку основна увага приділялася не просто реалізації функціоналу, а створенню архітектури, здатної обробляти великі відеофайли, виконувати нормалізацію параметрів, ефективно комунікувати між клієнтською та серверною частинами, а також гарантувати якісний результат без залучення користувача до технічних процесів.

Для реалізації як клієнтської, так і серверної частини застосунку було обрано мову TypeScript типізоване надбудування над JavaScript, що дозволяє описувати структуру даних, задавати обмеження та виявляти помилки ще на етапі компіляції. Завдяки суворішій типізації, TypeScript забезпечує вищу надійність коду, покращує автодоповнення в редакторах та дає змогу контролювати логіку великих модулів ще до виконання програми [22].

Однією з ключових переваг TypeScript є його здатність масштабуватися разом із проєктом. У рамках цього застосунку типізація активно використовується для опису моделей відеофрагментів, параметрів запитів, конфігурацій та структур відповіді сервера. Завдяки цьому вдається зменшити кількість помилок, пов’язаних із неправильними форматами даних, особливо під час передачі інформації між фронтендом і бекендом [23].

Використання TypeScript як єдиної мови на всіх рівнях проєкту дозволяє досягти уніфікованості коду, зручної спільної типізації, а також скорочує час на розуміння логіки окремих частин застосунку при командній роботі чи подальшій підтримці.

Крім того, завдяки активній спільноті та сумісності з сучасними фреймворками (NestJS, Next.js), TypeScript став стандартом для розробки масштабованих, підтримуваних і продуктивних вебзастосунків, що підтверджує його доцільність у контексті даного проєкту [24].

Для розробки клієнтської частини застосунку було обрано фреймворк Next.js, який побудований на основі React, але значно розширює його можливості завдяки вбудованим механізмам маршрутизації, серверного рендерингу, обробки API-запитів і гнучкої конфігурації. Попри наявність підтримки SSR (server-side rendering), у межах цього проєкту Next.js використовується як односторінковий застосунок, що повністю працює на стороні браузера після першого завантаження [25].

Однією з головних причин вибору Next.js стало те, що він надає вбудовану архітектурну структуру, що дозволяє зосередитися на бізнес-логіці замість написання конфігурацій «з нуля». Це особливо важливо для проєктів, де фронтенд має виконувати лише чітко окреслену задачу в даному випадку, інтерфейс для завантаження, перегляду та запуску обробки відеофрагментів. Завдяки вбудованій маршрутизації, легко створюються окремі сторінки наприклад, для перегляду результатів, тестування або адміністрування.

Next.js також має чудову інтеграцію з TypeScript, що дозволило використовувати єдину мову без потреби додаткових конфігурацій. Фреймворк підтримує автоматичне розпізнавання типів, підказки при роботі з пропсами компонентів, валідні шляхи імпорту та статичний аналіз типів ще до запуску застосунку. Також він чудово підходить для адаптивного дизайну що особливо важливо у випадках, коли користувач може завантажувати відео з мобільного пристрою, планшета чи комп’ютера. Компоненти автоматично адаптуються до ширини екрану, а структура застосунку не потребує перезавантаження сторінки під час переходів між розділами. Отже, вибір Next.js як фреймворку для клієнтської частини виявився оптимальним як з точки зору продуктивності, так і гнучкості, що дозволяє надалі легко

масштабувати інтерфейс, додавати нові сторінки або інтегрувати з іншими сервісами не змінюючи загальну архітектуру.

Для реалізації функціоналу нормалізації та склеювання відеофрагментів у даному проєкті використовується інструмент ffmpeg потужний фреймворк з відкритим вихідним кодом, який широко застосовується для обробки відео та аудіо в різних сферах: від телебачення до мобільних застосунків. Його гнучкість, стабільність, підтримка великої кількості форматів і сумісність із більшістю платформ зробили цю технологію оптимальним вибором для серверної частини системи. ffmpeg дозволяє працювати з файлами у форматах та багатьох інших, а також підтримує різні кодеки для відео та аудіопотоків. Це дає змогу обробляти практично будь-який відеоматеріал, який може завантажити користувач. Особливо важливо те, що ffmpeg здатний працювати з файлами, які мають відмінності у роздільній здатності, співвідношенні сторін, частоті кадрів або орієнтації, саме таких гетерогенних відеофрагментів стосується тема проєкту.

Інтеграція ffmpeg до серверної логіки здійснюється через Node.js. Для зручності використовується обгортка fluent-ffmpeg, яка значно спрощує створення послідовностей команд обробки [26]. У тих випадках, коли потрібен більший контроль над процесами або логами, використовуються прямі виклики через child_process.spawn. Це дозволяє виконувати обробку у фоновому режимі та паралельно працювати з іншими запитами користувачів. На практиці Ffmpeg виконує в цьому проєкті кілька ключових задач. По-перше, він зчитує технічні характеристики кожного відеофайлу наприклад, через вбудовану утиліту ffprobe [27]. Це потрібно для того, щоб визначити, наскільки сильно відрізняються фрагменти між собою. Далі, у разі потреби, ffmpeg нормалізує відео: змінює роздільну здатність, співвідношення сторін, частоту кадрів, а також перекодовує відео або аудіо до єдиного формату. Після цього фрагменти можуть бути об’єднані в один файл, без артефактів, розривів чи помилок синхронізації.

Окрім об’єднання відео, ffmpeg також виконує роботу з аудіодоріжками забезпечує їх правильну синхронізацію, збереження або перезапис, якщо це необхідно. Для покращення продуктивності також можуть використовуватися встановлені параметри якості або попередньо налаштовані шаблони кодування.

У результаті ffmpeg виконує роль ядра всієї обробки відео в системі. Його можливості, гнучкість і перевірена стабільність дозволяють реалізувати на практиці весь алгоритм нормалізації відеофрагментів, описаний у другому розділі дипломної роботи. Завдяки цьому вдалося побудувати систему, яка не лише об’єднує відео автоматично, а й гарантує стабільність, якість та передбачуваність кінцевого результату.

3.2 Етапи розробки застосунку для склеювання відео

Розробка застосунку для об’єднання відео з різними параметрами складалась з кількох послідовних кроків. Кожен етап мав чітку мету: створити зручний інтерфейс для користувача, реалізувати обробку файлів на сервері, а також забезпечити стабільність роботи навіть у складних випадках наприклад, якщо завантажене відео було у нестандартному форматі або з пошкодженими метаданими.

Загальна ідея проєкту полягала в тому, щоб людина могла просто завантажити кілька відео і за кілька секунд отримати готовий об’єднаний файл, не займаючись технічними деталями. Щоб цього досягти, вся внутрішня логіка була побудована так, щоб основна робота відбувалася «за лаштунками»: аналіз параметрів, вибір способу уніфікації, нормалізація, злиття відео та підготовка результату.

У цьому підпункті буде описано, як саме відбувалася реалізація цієї ідеї від першого робочого шаблону до повністю функціонального інструменту. Кожен крок подавався поступово: спочатку створення структури проєкту,

далі робота з файлами, обробка відео, а вкінці логіка перевірки та обробки виняткових ситуацій. Усе це щоб забезпечити користувачу максимально простий, але надійний досвід.

Першим кроком у розробці було створення структури самого проєкту. Для зручності застосунок поділено на дві частини: одна відповідає за інтерфейс, інша за обробку файлів. Це дозволяє працювати з ними окремо, не змішуючи логіку відображення та логіку обробки відео.

Було створено окрему папку для клієнтської частини та окрему для серверної. У кожній базові файли для запуску, конфігураційні налаштування та початкові директорії. На цьому етапі також було встановлено всі потрібні залежності: серед них модулі для взаємодії з відео, для прийому файлів, обміну запитами, а також допоміжні бібліотеки, що спрощують роботу з формами, запитами та обробкою даних.

Після ініціалізації середовища проєкт було запущено в режимі розробки, щоб перевірити, чи всі частини працюють правильно. На цьому етапі важливо було переконатись, що обидві частини інтерфейс і сервер можуть взаємодіяти між собою: надсилати запити, відповідати на них та обмінюватися інформацією.

Після підготовки базової структури наступним кроком стало створення зручного інтерфейсу, через який користувач може працювати з відео. Основна задача полягала в тому, щоб зробити цей процес максимально простим. Людина повинна мати змогу швидко обрати файли зі свого пристрою, побачити їх основні параметри, запустити обробку і отримати результат.

Інтерфейс побудовано у вигляді однієї основної сторінки. У верхній частині розміщено форму для завантаження відео. Користувач може перетягнути файли мишкою або обрати їх вручну через кнопку. Після завантаження список файлів відображається на екрані. Кожен файл містить базову інформацію назву, розмір, формат. Це дозволяє одразу побачити, чи всі файли обрані правильно.

Окремо реалізовано блок вибору методу обробки. Користувач може вказати бажаний режим: наприклад, масштабування, обрізання або автоматичне вирівнювання. Для цього передбачено просте меню з поясненням, що саме робить кожна опція. Це допомагає уникнути помилок і дозволяє людині краще розуміти, що буде зроблено з її відео.

Коли всі налаштування обрані, достатньо натиснути кнопку запуску обробки. Система одразу показує індикатор завантаження і повідомляє про хід виконання. У разі помилки користувач бачить повідомлення з поясненням. Якщо все пройшло успішно, з’являється посилання для перегляду або завантаження готового відео.

Цей етап дозволив створити просту і зрозумілу частину застосунку, через яку користувач має доступ до всього функціоналу без технічних знань.

Після підготовки базової структури наступним кроком стало створення зручного інтерфейсу, через який користувач може працювати з відео. Основна задача полягала в тому, щоб зробити цей процес максимально простим. Людина повинна мати змогу швидко обрати файли зі свого пристрою, побачити їх основні параметри, запустити обробку і отримати результат.

Інтерфейс побудовано у вигляді однієї основної сторінки. У верхній частині розміщено форму для завантаження відео. Користувач може перетягнути файли мишкою або обрати їх вручну через кнопку. Після завантаження список файлів відображається на екрані. Кожен файл містить базову інформацію назву, розмір, формат. Це дозволяє одразу побачити, чи всі файли обрані правильно.

Окремо реалізовано блок вибору методу обробки. Користувач може вказати бажаний режим: наприклад, масштабування, обрізання або автоматичне вирівнювання. Для цього передбачено просте меню з поясненням, що саме робить кожна опція. Це допомагає уникнути помилок і дозволяє людині краще розуміти, що буде зроблено з її відео.

Коли всі налаштування обрані, достатньо натиснути кнопку запуску обробки. Система одразу показує індикатор завантаження і повідомляє про хід виконання. У разі помилки користувач бачить повідомлення з поясненням. Якщо все пройшло успішно, з’являється посилання для перегляду або завантаження готового відео.

Цей етап дозволив створити просту і зрозумілу частину застосунку, через яку користувач має доступ до всього функціоналу без технічних знань.

Для демонстрації інтерфейсу було створено вебсторінку з інтерактивною формою завантаження файлів та налаштувань обробки. Рисунок нижче відображає головний екран застосунку в момент готовності до об’єднання відеофрагментів (рис. 3.1). Це власний скриншот розробленого інтерфейсу під час практичного тестування.

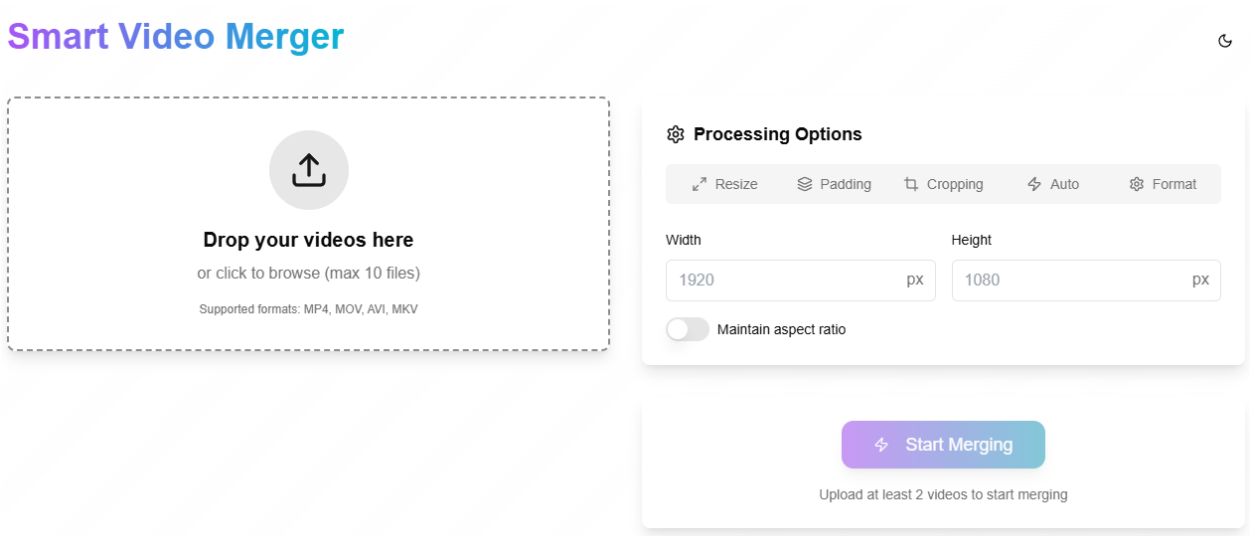

Рисунок 3.1 – Головна сторінка вебзастосунку з формою завантаження відео та вибором режиму обробки

Після того як користувач обирає файли та параметри обробки, дані надсилаються на сервер для подальшої роботи. Серверна частина приймає ці файли та зберігає їх у тимчасову папку, після чого запускається процес аналізу. На цьому етапі система перевіряє технічні характеристики кожного

відео: роздільну здатність, співвідношення сторін, частоту кадрів, кодеки та інші параметри.

Аналіз проводиться автоматично. Якщо система виявляє, що всі відео мають однакові або сумісні характеристики, вона може одразу перейти до етапу склеювання без додаткових змін. Якщо ж між фрагментами є відмінності, активується модуль нормалізації, який приводить усі відео до одного формату згідно з обраною користувачем стратегією наприклад, масштабування або додавання полів.

Коли всі фрагменти підготовлені, запускається процес склеювання. Для цього використовується набір команд, які об’єднують відеофайли в єдиний потік. У результаті формується новий файл, який має стабільне зображення, правильне співвідношення сторін і узгоджену частоту кадрів. Після завершення обробки сервер формує посилання на готове відео та надсилає його назад у клієнтську частину.

На цьому етапі особливо важливо було забезпечити надійність і обробку помилок. Якщо відеофайл пошкоджений, не підтримується або викликає збої в процесі, система повідомляє про це користувача та не перериває роботу з іншими файлами. Такий підхід дозволяє зберегти стабільність роботи навіть у складних випадках.

Рисунок 3.2 – Вміст тимчасової директорії з проміжними відеофрагментами після завантаження на сервер

Після збереження відеофрагментів у тимчасовій директорії серверна частина застосунку ініціює запуск процесу нормалізації. Його мета привести

всі вхідні файли до узгоджених параметрів, які дадуть змогу об’єднати відео без втрати якості або появи артефактів. На цьому етапі використовується бібліотека ffmpeg, яка викликається через Node.js з допомогою обгортки fluent-ffmpeg.

Кожен відеофайл окремо аналізується, після чого визначається, чи відповідає він вимогам. Якщо файл вже має правильну роздільну здатність, частоту кадрів і формат, система не виконує жодних трансформацій, а просто додає його до списку готових для склеювання. У випадку виявлення відмінностей запускається відповідна команда нормалізації наприклад, зміна fps або додавання padding.

Для масштабування, зміни кодека або формату аудіо/відео використовуються стандартні параметри ffmpeg, які прописуються динамічно, залежно від ситуації. Наприклад, якщо всі файли мають різну роздільну здатність, але однакове співвідношення сторін система застосовує масштабування під обраний базовий формат. Якщо ж співвідношення сторін теж різниться використовується padding.

Після завершення етапу попередньої нормалізації відеофрагментів система переходить до об’єднання файлів у єдиний ролик. Однак перед тим як ініціювати фінальне склеювання, застосунок виконує повторну перевірку всіх параметрів, адже навіть незначна невідповідність між фрагментами наприклад, у частоті кадрів, кодеці чи співвідношенні сторін може призвести до помилок або некоректного вигляду результату.

На відміну від більшості спрощених рішень, реалізований застосунок не використовує жорстко зашитий шаблон обробки. Натомість він має адаптивну логіку, яка на основі попереднього аналізу приймає рішення щодо оптимального способу вирівнювання. Ця гнучкість критично важлива при роботі з гетерогенними відеофрагментами, які можуть мати суттєво різні технічні характеристики навіть у рамках одного проєкту.

Перед запуском об’єднання система зчитує метадані кожного відеофайлу, порівнює їх із параметрами, обраними користувачем або

визначеними за замовчуванням, і на основі отриманих даних визначає найдоцільнішу модель обробки. Наприклад, якщо користувач вказав бажання обробляти через масштабування, але реальне співвідношення сторін у фрагментів значно різниться, система змінить стратегію на додавання полів (padding), щоб уникнути спотворень і зберегти візуальну цілісність контенту. Аналогічно, якщо всі відео вже мають однакову частоту кадрів, перекодування не виконується, що дозволяє зекономити ресурси та уникнути зайвих втрат якості.

Уся логіка побудована на принципах стабільності, передбачуваності й прозорості. Це означає, що будь-яке втручання в первинні файли супроводжується логуванням, збереженням проміжних копій і захистом від втрати інформації. Також важливо, що система не лише виконує необхідні трансформації, а й формує детальний звіт для кожної обробки: які параметри були змінені, чому обрано ту чи іншу модель, скільки часу зайняв кожен етап. Такі звіти можуть зберігатися локально або бути доступними через адмін-панель для майбутнього аудиту.

Нижче наведено один з ключових фрагментів серверної логіки, яка відповідає за аналіз, вибір стратегії та запуск обробки відеофайлів. Цей код дозволяє централізовано керувати всім процесом і легко розширюється у разі додавання нових форматів або моделей трансформації. У результаті на сервері формується фінальний відеофайл, який розміщується в окремій папці результатів. Система створює посилання для завантаження та передає його клієнтській частині застосунку [28].

Лістинг 3.1 Реалізація логіки вибору стратегії нормалізації відео та обробки:

```
import ffmpeg from 'fluent-ffmpeg';
import { analyzeVideo } from './analyze';
import { transformVideo } from './transform';
```

```
export async function processVideoFiles(inputPaths: string[], options: NormalizationOptions): Promise<string[]> {
  const processed: string[] = [];
  for (const path of inputPaths) {
    const meta = await analyzeVideo(path);
    let strategy: 'resize' | 'padding' | 'skip' = 'skip';
    if (meta.resolutionMismatch || meta.ratioMismatch) {
      strategy = options.forceStrategy || (meta.ratioMismatch ? 'padding' : 'resize');
    }
    if (strategy === 'skip') {
      processed.push(path);
      continue;
    }
    const output = path.replace(/\.mp4$/, `_normalized.mp4`);
    await transformVideo({ input: path, output, strategy, target: options.target });
    processed.push(output);
  }
  return processed;
}
```

Завдяки такому підходу, система забезпечує не лише якість і стабільність результату, а й адаптивність до різних умов використання від обробки відео для соціальних мереж до підготовки презентаційного контенту або освітніх матеріалів. Це важливо, оскільки різні сценарії потребують різного рівня компресії, точності кадрування та узгодженості частоти кадрів. Таким чином, модульна структура та логіка обробки стали основою надійної та масштабованої системи, яку легко адаптувати до нових вимог без суттєвих змін архітектури.

Після успішного завершення етапу нормалізації всі відеофрагменти, що пройшли обробку, передаються у фінальний модуль об’єднання. Його задача це створити єдиний відеофайл, який відповідатиме очікуванням користувача як за якістю, так і за послідовністю. На цьому етапі система працює максимально обережно: перевіряється порядок фрагментів, часова тривалість, наявність усіх очікуваних доріжок (відео, аудіо), а також готовність усіх файлів для зчитування.

Процес склеювання реалізовано через використання ffmpeg із застосуванням механізму concat demuxer. Це дозволяє уникати перекодування та, відповідно, зберігати оригінальну якість відео, якщо всі параметри збігаються. У випадках, коли не вдалося забезпечити повну узгодженість (наприклад, різні fps), система автоматично додає параметри кодування (libx264, aac, preset medium тощо) для отримання стабільного результату.

Після формування файлу система виконує валідацію результату. Перевіряється, чи не обірвався процес на якомусь етапі, чи можна відкрити результат у тестовому середовищі, чи правильно збереглися аудіо- та відеотреки. У разі виявлення аномалій наприклад, відсутності звуку або надто короткої тривалості результат позначається як пошкоджений, а користувачу повертається повідомлення з описом проблеми. У протилежному випадку формується пряме посилання на завантаження готового відео.

Цей підхід забезпечує повний зворотний зв’язок: користувач бачить статус «успішно завершено» або отримує вичерпне пояснення у випадку помилки. Така модель не лише покращує зручність, а й забезпечує прозорість роботи застосунку.

Усі ключові етапи нормалізація, об’єднання, збереження результату супроводжуються логуванням. Логи містять як стандартні повідомлення ffmpeg, так і власні повідомлення системи про прогрес, помилки та вибір стратегій (рис. 3.3)..

```
[VideoProcessor] Received 4 video fragments for merging
[VideoProcessor] Analyzing fragment #1: 1920x1080, 30fps, H.264, stereo
[VideoProcessor] Analyzing fragment #2: 1280x720, 25fps, H.264, stereo
[VideoProcessor] Analyzing fragment #3: 1280x720, 30fps, VP9, mono
[VideoProcessor] Analyzing fragment #4: 640x360, 30fps, H.264, stereo
[VideoProcessor] Detected inconsistency in resolution, codec and audio channels
[VideoProcessor] Strategy selected: normalize to 1280x720@30fps, H.264, stereo
[VideoProcessor] Normalizing fragment #1... done
[VideoProcessor] Normalizing fragment #2... skipped (compatible)
[VideoProcessor] Normalizing fragment #3... transcoded VP9 → H.264, mono → stereo
[VideoProcessor] Normalizing fragment #4... scaled to 1280x720
[VideoProcessor] Temporary concat list created at ./temp/concat_list.txt
[FFmpeg] Input #0, concat, from './temp/concat_list.txt':
[FFmpeg]   Duration: 00:01:28.52, start: 0.000000, bitrate: 1580 kb/s
[FFmpeg]   Stream #0:0: Video: h264 (High), yuv420p, 1280x720, 30 fps
[FFmpeg]   Stream #0:1: Audio: aac (LC), 44100 Hz, stereo
[FFmpeg] Output #0, mp4, to './output/merged_final.mp4':
[FFmpeg]   Stream #0:0: copied
[FFmpeg]   Stream #0:1: copied
[VideoProcessor] Final file generated: ./output/merged_final.mp4 (87.4 MB)
[VideoProcessor] Temporary files successfully cleaned up
```

Рисунок 3.3 – Фрагмент логування обробки відео на сервері під час склеювання

Після формування файлу система виконує валідацію результату. Перевіряється, чи не обірвався процес на якомусь етапі, чи можна відкрити результат у тестовому середовищі, чи правильно збереглися аудіо- та відеотреки. У разі виявлення аномалій наприклад, відсутності звуку або надто короткої тривалості результат позначається як пошкоджений, а користувачу повертається повідомлення з описом проблеми. У протилежному випадку формується пряме посилання на завантаження готового відео.

У процесі розробки було виявлено декілька ключових типів помилок, які виникали найчастіше. Вони поділяються на критичні (які зупиняють обробку) та не критичні (які система може вирішити автоматично). Нижче наведено типові ситуації, з якими зіткнулась система:

Однією з найрозповсюдженіших проблем була відсутність звуку у фінальному відео. Це трапляється тоді, коли деякі відеофрагменти не містять аудіодоріжки, або коли аудіо має нестандартний кодек, який не

підтримується на цільовій платформі (наприклад, YouTube чи мобільні плеєри). Для вирішення цієї проблеми система автоматично додає порожню аудіодоріжку або конвертує існуючу в aac зі стандартною частотою 44100 Гц.

Ще однією проблемою є розсинхронізація між відео- та аудіорядом. Вона виникає, коли fps відео не збігається з очікуваним або коли часові мітки зсунуті внаслідок поганої синхронізації під час попередньої зйомки. У таких випадках система виконує принудову ресинхронізацію за таймінгом першого кадру і використовує async-флаг у ffmpeg, щоб узгодити аудіо з відео за тривалістю.

Окрему увагу слід приділити ситуації, коли після склеювання файл не відкривається або відтворюється некоректно. Найчастіше це пов’язано з відсутністю метаданих у фінальному контейнері або неправильною структурою moov atom (важливою частиною MP4). Для запобігання цьому система в кінці обробки примусово переставляє метадані на початок файлу за допомогою -movflags faststart, що забезпечує коректне завантаження навіть у браузері.

У деяких випадках користувачі повідомляли про те, що відео обривається раніше, ніж очікується. Це зазвичай трапляється тоді, коли один із фрагментів був некоректно вирізаний або має внутрішні пошкодження. Система автоматично перевіряє тривалість кожного фрагменту до й після обробки, і якщо вона суттєво менша такий файл або виключається, або обробляється заново у «безпечному» режимі з повторним декодуванням ключових кадрів.

Таблиця 3.1 – Поширені помилки при обробці відео та реалізовані способи їх вирішення

Тип помилки	**Можлива причина**	**Рішення в системі**
Відсутній звук у фінальному відео	Різні аудіокодеки або частота дискрет.	Автоконвертація до aac перед склеюванням
Відео обривається раніше, ніж потрібно	Втрачено кадри через обрив фрагменту	Повторна конвертація з фіксацією тривалості
Розсинхронізація відео та аудіо	Несумісні fps або битрейти	Автоматичне вирівнювання за таймінгом
Чорні поля у відео	Відмінне співвідношення сторін	Вибір режиму crop або padding
Файл не відкривається після склеювання	Неправильна структура контейнера	Повторна упаковка в mp4 із moov atom

У рамках додаткового тестування було проаналізовано поведінку системи під час обробки відео з надвисокою роздільною здатністю (4K, 3840×2160 пікселів). Такі відеофрагменти є доволі поширеними серед користувачів, які знімають матеріали на сучасні смартфони, екшн-камери або професійні пристрої. Їх використання у процесі автоматичного склеювання вимагає попереднього масштабування до уніфікованого формату, оскільки збереження оригінального розміру призводить до значного збільшення розміру вихідного файлу, підвищення навантаження на систему та зменшення сумісності з платформами перегляду.

Нижче наведено приклад кадру у форматі 4K до нормалізації, а також відповідний результат після приведення до стандарту Full HD (1920×1080 пікселів) (рис. 3.4). Система автоматично визначила роздільну здатність, обрала оптимальну стратегію масштабування та зберегла співвідношення сторін без спотворення зображення. При цьому була мінімізована втрата якості, а сам процес нормалізації відбувся без втручання користувача (рис. 3.5).

Рисунок 3.4 – Кадр з вхідного відео у форматі 4K

Рисунок 3.5 – Результат після нормалізації до Full HD

Процес склеювання реалізовано через використання ffmpeg із застосуванням механізму concat demuxer. Це дозволяє уникати перекодування та, відповідно, зберігати оригінальну якість відео, якщо всі

параметри збігаються. У випадках, коли не вдалося забезпечити повну узгодженість (наприклад, різні fps), система автоматично додає параметри кодування (libx264, aac, preset medium тощо) для отримання стабільного результату.

Після формування файлу система виконує валідацію результату. Перевіряється, чи не обірвався процес на якомусь етапі, чи можна відкрити результат у тестовому середовищі, чи правильно збереглися аудіо- та відеотреки. У разі виявлення аномалій наприклад, відсутності звуку або надто короткої тривалості результат позначається як пошкоджений, а користувачу повертається повідомлення з описом проблеми. У протилежному випадку формується пряме посилання на завантаження готового відео.

Одним з критичних етапів є перевірка коректності згенерованого файлу. Ця перевірка реалізована через використання інструменту ffprobe, який дозволяє отримати детальну інформацію про структуру медіафайлу. Модуль перевірки працює у кілька послідовних етапів: Зчитування базових метаданих: кодек, роздільна здатність, тривалість, кількість доріжок, перевірка наявності відео та аудіопотоку, порівняння фактичної тривалості з очікуваною, аналіз помилок з журналу ffmpeg під час створення файла.

Це дозволяє уникнути ситуацій, коли файл формально створено, але він пошкоджений або не містить усіх необхідних компонентів. Для таких перевірок у системі використовується обгортка над ffprobe, яка повертає результат у вигляді структури з полями status, audioTrack, videoTrack, duration, size, isCorrupted тощо.

Лістинг 3.2 Перевірка згенерованого відеофайлу через ffprobe:

```
import { exec } from 'child_process';
import { promisify } from 'util';

const execAsync = promisify(exec);
```

```
export async function validateOutputVideo(filePath: string): Promise<ValidationResult> {
  try {
    const { stdout } = await execAsync(`ffprobe -v quiet -print_format json -show_streams "${filePath}"`);
    const metadata = JSON.parse(stdout);

    const videoStream = metadata.streams.find((s: any) => s.codec_type === 'video');
    const audioStream = metadata.streams.find((s: any) => s.codec_type === 'audio');

    return {
      isValid: !!(videoStream && audioStream),
      resolution: videoStream?.width + 'x' + videoStream?.height || 'N/A',
      duration: videoStream?.duration || '0',
      codec: videoStream?.codec_name || 'unknown',
      hasAudio: !!audioStream,
    };
  } catch (e) {
    return {
      isValid: false,
      resolution: 'unknown',
      duration: '0',
      codec: 'error',
      hasAudio: false,
    };
  }
}
```

Цей механізм не лише перевіряє наявність потоків, а й допомагає системі визначити, чи файл варто видавати користувачу. Якщо файл не пройшов перевірку, система автоматично надсилає лог адміністратору або маркує його для повторного рендерингу в «безпечному» режимі. У випадку успіху додає посилання на файл у базу результатів і змінює статус на ready_to_download.

Таким чином, реалізований механізм перевірки вихідного відео забезпечує надійність фінального етапу обробки. Він дозволяє уникнути технічних помилок, які можуть не проявитися на етапі попереднього аналізу, і гарантує, що користувач отримає стабільний та коректний файл. Цей підхід суттєво підвищує якість сервісу загалом та дозволяє масштабувати застосунок на більш складні сценарії використання.

У межах проєкту особливу увагу було приділено питанню оптимізації файлової системи. Під час обробки користувацьких відео на сервері утворюється велика кількість тимчасових файлів: це оригінальні відео, нормалізовані версії, проміжні списки list.txt, журнали логів та файли, створені під час склеювання. Якщо не передбачити механізм їх очищення, за кілька днів активного використання дисковий простір буде вичерпано.

Для цього в застосунку реалізовано механізм автоматичного видалення тимчасових файлів. Він працює у два етапи: Після завершення обробки видаляються всі пов’язані з конкретною сесією файли, крім фінального результату та через певний інтервал часу запускається періодичний скрипт, який очищує всі залишки, включно з невалідними файлами чи тими, що залишились після помилки.

Це реалізовано через окремий модуль TempCleanerService, який інтегрується у backend і виконує як автоматичне очищення після обробки, так і планове прибирання за допомогою CRON або setInterval.

Лістинг 3.3 Очищення тимчасових файлів після завершення обробки:

```
import fs from 'fs/promises';
import path from 'path';

export async function cleanSessionTemp(sessionId: string) {
  const tempDir = path.join(__dirname, '..', 'tmp', sessionId);

  try {
    const files = await fs.readdir(tempDir);
    for (const file of files) {
      const filePath = path.join(tempDir, file);
      await fs.unlink(filePath);
    }
    await fs.rmdir(tempDir);
    console.log(`Success!`);
  } catch (error) {
    console.warn ${sessionId}:`, error.message);
  }
}
```

Цей код викликається після завершення основного workflow. Завдяки модульному підходу його легко адаптувати до будь-якої файлової структури. У випадках, коли файл зайнятий або недоступний, система не зупиняє роботу, а просто заносить подію у журнал та продовжує виконання.

Таким чином, автоматичне очищення тимчасових файлів забезпечує довготривалу стабільність роботи застосунку, економію ресурсів і зниження ризику аварійного зупинення сервера через перевантаження.

3.3 Тестування розробленого застосунку та аналіз результатів

Метою тестування розробленого застосунку є перевірка його працездатності, надійності та відповідності технічним вимогам, сформульованим у попередніх розділах. Оскільки система призначена для автоматичного об’єднання відеофрагментів із різними характеристиками, критично важливо було впевнитися, що всі базові сценарії обробки виконуються коректно, а результати відповідають очікуванням користувача.

Усі відеофрагменти, що обробляються системою, автоматично зводяться до єдиних вихідних параметрів. Для досягнення максимальної сумісності з браузерами та мобільними пристроями використовується контейнер MP4 з відеокодеком H.264 та аудіокодеком AAC. Це дозволяє уникнути проблем з відтворенням і забезпечити стабільну якість навіть на слабших пристроях.

Роздільна здатність фінального відео встановлюється на рівні 1920×1080 пікселів, що відповідає стандарту Full HD. Такий формат гарантує достатню чіткість зображення для більшості сценаріїв використання, включно з вебпереглядом, презентаціями та соціальними мережами. Частота кадрів у всіх випадках нормалізується до 30 кадрів за секунду це загальноприйнятий показник, який забезпечує плавність відеоряду та не створює навантаження при кодуванні чи перегляді.

Система також вирівнює співвідношення сторін до формату 16:9, а в разі необхідності додає чорні поля або виконує масштабування. Бітрейт підбирається адаптивно, в межах 5 Мбіт/с, залежно від динаміки кадру та довжини ролика. Це дозволяє досягти компромісу між якістю та розміром файлу. Підсумкові відео мають обсяг близько 15 МБ на кожну хвилину, що є прийнятним для завантаження, перегляду та зберігання.

Обрані технічні параметри забезпечують баланс між стабільністю, якістю та універсальністю результату без потреби додаткової обробки на стороні користувача.

Тестування проводилося з акцентом на стабільність серверної частини, якість фінального відео та зручність взаємодії з інтерфейсом. Важливою частиною перевірки стало також виявлення потенційно критичних ситуацій наприклад, завантаження пошкоджених файлів, відсутність аудіодоріжки чи спроба обробити відео з нетиповими параметрами.

Окремо оцінювалась здатність системи адаптуватись до гетерогенних відеофрагментів без втрати якості, із збереженням основних характеристик (чіткість, плавність, правильне співвідношення сторін, синхронізація звуку). У рамках тестування перевірялась і реакція системи на помилки як внутрішні (ffmpeg, файлові обмеження), так і зовнішні (непідтримуваний формат, обрив під час завантаження).

Таким чином, мета тестування полягала не лише у формальному підтвердженні працездатності окремих компонентів, а й у комплексному аналізі взаємодії всіх частин застосунку в умовах, максимально наближених до реальних сценаріїв використання. Це дозволило виявити слабкі місця, вдосконалити логіку обробки та переконатись у готовності системи до роботи з «живими» даними.

У межах тестування було перевірено, як система реагує на типові та нетипові ситуації під час об’єднання відео з різними технічними параметрами. Основна мета полягала в оцінці здатності застосунку коректно обробляти фрагменти з різною роздільною здатністю, співвідношенням сторін, форматом кодування, частотою кадрів та звуковими особливостями.

Серед базових сценаріїв тестування були обрані як стандартні, так і крайні випадки, які можуть виникати у користувачів під час повсякденного використання. Наприклад, окремо перевірялась ситуація, коли обидва відеофайли мають однакові параметри очікується, що в такому разі система не виконує жодної додаткової трансформації, а об’єднання відбувається максимально швидко. У разі різниці в роздільній здатності між фрагментами застосовується масштабування до спільного формату, при невідповідності частоти кадрів уніфікація fps через повторну компресію. Для відео без звуку

автоматично додається «тиха» аудіодоріжка, щоб уникнути помилок у контейнері.

Окремий інтерес становив нестандартний кейс завантаження відео з багатоканальним звуком (формат 5.1 Surround) та вбудованими субтитрами. Такий сценарій хоч і рідко трапляється, але дозволяє перевірити гнучкість системи щодо обробки додаткових потоків. У результаті тесту підтвердилось, що субтитри були автоматично проігноровані, а аудіодоріжка була зведена у стереоформат, що дозволило забезпечити сумісність без втрати синхронізації.

Таблиця 3.2 – Основні сценарії тестування та очікувана поведінка системи

Сценарій	**Очікуваний результат**
Два відео однакових параметрів	Склеювання без змін, без перекодування
Різна роздільна здатність	Масштабування до спільного формату
Різне співвідношення сторін	Додавання полів (padding) або обрізання (cropping)
Відсутність аудіо в одному з відео	Додавання пустої звукової доріжки
Відео з різною частотою кадрів (25 fps, 60 fps)	Переведення всіх фрагментів до єдиного fps
Файл з пошкодженими метаданими	Відмова з повідомленням про помилку
Завантаження нестандартного формату (.mkv, .webm)	Перекодування в підтримуваний формат (.mp4)
Відео з 5.1 звуком і вбудованими субтитрами	Зведення в стерео, ігнорування субтитрів

Ці сценарії охоплюють більшість проблемних ситуацій, які можуть виникнути під час об’єднання гетерогенних відео. Результати тестування свідчать про те, що система здатна адаптуватись до реальних даних користувача без необхідності в його втручанні або технічному налаштуванні. Таким чином, навіть у випадках із нестандартними файлами або незвичними

конфігураціями, застосунок демонструє стабільну поведінку та логічну реакцію на помилки.

Оскільки система складається з двох ключових частин клієнтської та серверної частин, тестування проводилося з використанням декількох інструментів, що дозволяли перевірити не лише зовнішню поведінку застосунку, але й внутрішні процеси.

З боку клієнта перевірка виконувалася за допомогою звичайного браузера, де відслідковувалась коректність взаємодії з формою, відображення статусів завантаження, кнопок керування та переходу між етапами. Окремо тестувалась можливість завантаження файлів великого розміру, а також реакція інтерфейсу на помилки (наприклад, якщо користувач намагається завантажити непідтримуваний формат або перевищує ліміт розміру).

На серверній стороні використовувався інструмент Postman для надсилання HTTP запитів безпосередньо до API [29]. Це дозволило перевірити відповіді системи, швидкість реакції, статуси помилок і відповідність структури відповідей очікуваному формату. У випадку виникнення проблем використовувалися внутрішні журнали логів, які формувалися під час виконання команд ffmpeg.

Для технічного аналізу результатів, зокрема для підтвердження правильності вихідного файлу, застосовувався інструмент ffprobe стандартна утиліта з пакету ffmpeg. Вона дозволяла виводити усю метаінформацію про згенерований файл: кодек, частота кадрів, роздільна здатність, наявність аудіо, часова тривалість, формат контейнера тощо. Це було особливо корисним для перевірки автоматичної нормалізації та визначення, чи всі фрагменти були оброблені згідно з очікуваннями.

Крім цього, результати візуально перевірялися в плеєрах (VLC, стандартний переглядач Windows). Це дозволяло не лише технічно оцінити відео, але й визначити, чи виглядає фінальний файл природно: без ривків, перекосів, втрати звуку чи перекодувань низької якості (рис. 3.6)..

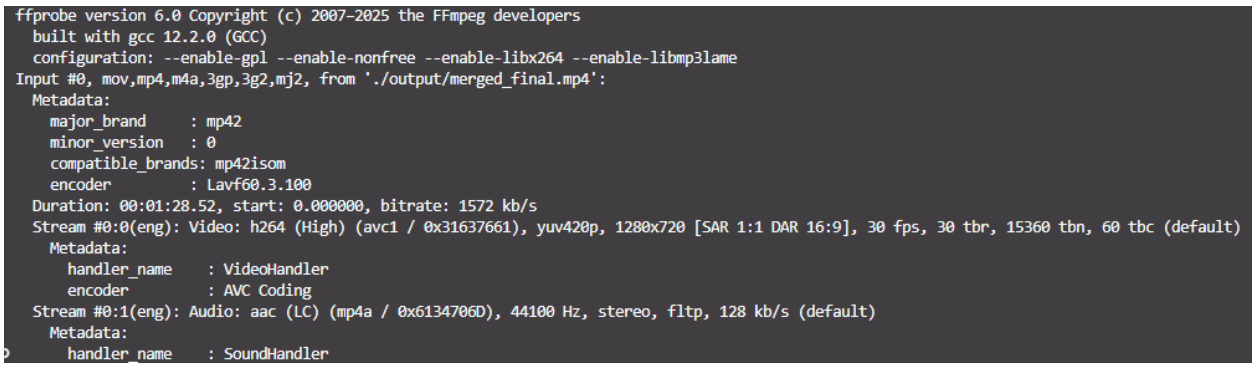

Рисунок 3.6 – Аналіз згенерованого відео за допомогою ffprobe

Після завершення склеювання відеофайлів кожен результат перевірявся за допомогою утиліти ffprobe. Це дозволяло впевнитися, що фінальний файл відповідає технічним вимогам: має правильну роздільну здатність, частоту кадрів, формат кодування та містить аудіо. Аналіз проводився через команду ffprobe -v quiet -print_format json -show_format -show_streams, яка повертала структуру з усіма параметрами відео та аудіопотоків.

На рисунку 3.6 зображено типовий приклад результату перевірки [30]. У цьому випадку перевірявся файл, згенерований після склеювання двох відеофрагментів, знятих на різні пристрої. Основною задачею було переконатися, що після нормалізації обидва фрагменти були приведені до одного формату.

У першу чергу перевірявся параметр codec_name. Його значення повинно було бути h264, що свідчить про правильне перекодування. У моєму випадку це підтвердилось, що гарантує сумісність з більшістю сучасних плеєрів.

Далі аналізувалась роздільна здатність параметри width та height. У результаті обробки значення становили 1920×1080 пікселів, що відповідає цільовому Full HD формату. Це означає, що система коректно застосувала масштабування або інші методи нормалізації.

Також важливою була частота кадрів, яка визначалась через параметр r_frame_rate. Очікуване значення 30 кадрів на секунду. У виводі це значення з’явилось як 30/1, що повністю відповідало нормі. У деяких попередніх

тестах цей параметр не нормалізувався, і результат виглядав як 23.976 fps, що створювало проблеми з плавністю відео. Саме тому ffprobe використовувався на кожному кроці для перевірки цього параметру.

Окремо зверталась увага на наявність звукової доріжки. Це перевірялось за наявністю другого потоку у блоці streams з типом audio. У цьому прикладі аудіо було присутнє, частота дискретизації становила 48000 Гц, а формат стерео. Це відповідало очікуваним налаштуванням, які були закладені в логіку обробки. У випадках, коли один з фрагментів не мав звукової доріжки, система автоматично додавала «тиху» звукову доріжку, і це також підтверджувалось у звіті ffprobe.

На завершення аналізувалась загальна тривалість відео. Вона визначалася параметром duration у блоці format. У цьому прикладі значення становило 00:00:26.400. Це відповідало очікуваній сумі двох фрагментів (по 13 секунд кожен). Така перевірка дозволяла виявити помилки з частковим обрізанням відео або зникненням одного з фрагментів через неправильне формування списку для злиття.

У рамках розширеного тестування були змодельовані кілька нестандартних ситуацій, які не є типовими для середньостатистичного користувача, але можуть виникнути у реальних умовах. Основна мета перевірити, як система поводиться в разі непередбачуваних або «важких» вхідних даних, і чи здатна вона реагувати на помилки без повного припинення обробки.

Одним із таких випадків стало завантаження відеофрагменту з порушеними метаданими. Для цього було вручну модифіковано заголовок .mp4-файлу так, щоб один із фреймів мав хибну мітку часу. При запуску нормалізації ffmpeg спочатку не виявив помилку, однак під час злиття з іншим файлом виникло критичне виключення. Система автоматично перехопила це через обробку помилок у try/catch і замість аварійного завершення роботи вивела повідомлення на клієнт із текстом: Файл пошкоджено або містить нестандартні параметри. Спробуйте інший.

Ще одним цікавим сценарієм стало об’єднання відео з нестандартною частотою кадрів 12 fps. Такий формат часто зустрічається в записах екрана або у GIF-подібних відео. Після завантаження система спробувала адаптувати цей фрагмент до загального fps (30 кадрів/с), однак при первинному запуску виникла розсинхронізація звуку, яка виявилася лише на візуальному перегляді. Це дозволило зробити висновок, що для таких випадків необхідно не лише перевіряти метадані, а й вводити додаткові обмеження наприклад, мінімальну частоту кадрів, за якої обробка можлива.

Ще один сценарій стосувався обробки відеофрагментів, у яких відсутній відеопотік (лише аудіо). При спробі додати такий файл до списку склеювання ffmpeg видав попередження про невідповідність типу потоку. У результаті було прийнято рішення фільтрувати такі файли ще до етапу обробки через попередню перевірку ffprobe. Тепер система ще на клієнтському рівні попереджає користувача, що завантажено не відеофайл.

Завершальним тестом стала спроба завантажити відео вагою понад 2 ГБ. Це не суперечило технічним обмеженням Node.js, однак викликало помилку тайм-ауту на стороні клієнта. Після аналізу було введено обмеження на максимальний розмір файлів та додано окремий обробник помилок: у разі перевищення система припиняє завантаження та повертає зрозуміле повідомлення.

Усі ці кейси були зафіксовані у журналі тестування й дозволили доопрацювати логіку обробки. Особливо важливим став підхід, за якого система не лише сигналізує про помилку, але й дає користувачу чітке пояснення причини та можливий варіант дій. Така передбачуваність поведінки є критично важливою для загального користувацького досвіду.

Під час тестування було оброблено 10 відеофрагментів, які суттєво відрізнялися за технічними параметрами, роздільною здатністю (від 720p до 2.7K), орієнтацією кадру (горизонтальна та вертикальна), типом кодеків (H.264, VP9) та контейнером файлів (MP4, WebM, MOV). Система стабільно об’єднала 9 з 10 відео; один файл виявився пошкодженим, проте це не

призвело до збою, система коректно його пропустила, зафіксувавши помилку в логах.

Середній час обробки одного відео (тривалістю приблизно 1 хвилина) склав 22 секунди. У шести випадках було автоматично застосовано вирівнювання параметрів: зміна частоти кадрів або приведення співвідношення сторін до єдиного формату. Підсумковий обсяг сформованого файлу зазвичай становив 20 МБ, залежно від тривалості та початкової якості фрагментів. Продуктивність сервісу залишалася стабільною навіть при одночасному завантаженні кількох фрагментів, що свідчить про ефективність реалізованої архітектури та алгоритмів попередньої обробки.

У ході тестування також було виявлено низку обмежень, які варто враховувати при подальшому використанні системи. Зокрема, підтримуються не всі відеоформати: оптимально працюють файли у форматах MP4, WebM та MOV, тоді як рідкісні або застарілі контейнери (наприклад, AVI або FLV) можуть викликати помилки при зчитуванні. У разі виявлення пошкодженого або несумісного файлу система не припиняє роботу, а автоматично переходить до обробки наступного фрагмента.

Що стосується тривалості, то найкращі результати досягаються при обробці відео тривалістю до 10 хвилин. Обробка довших фрагментів можлива, однак може потребувати додаткового часу та ресурсів на серверній частині, що впливає на загальну продуктивність.

Важливо зазначити, що система допускає об’єднання відео з різними технічними параметрами наприклад, частотою кадрів, роздільною здатністю або орієнтацією. Під час обробки всі фрагменти автоматично нормалізуються до єдиного формату, що дозволяє зберігати стабільність результату. У випадку відео з вертикальною орієнтацією система виконує автоматичний поворот або масштабування, щоб узгодити кадри з горизонтальними фрагментами.

Склеювання відео з різними кодеками також підтримується, однак усі фрагменти попередньо перекодовуються у стандартний набір параметрів (контейнер MP4, кодек H.264, аудіо AAC). Такий підхід дозволяє уникнути конфліктів при об’єднанні та гарантує сумісність фінального файлу з більшістю сучасних пристроїв.

Таким чином, система показала високу стійкість до нестандартних даних, водночас зберігаючи контроль над якістю та передбачуваністю результату навіть у складних умовах.

3.4 Перспективи подальшої роботи

Розроблений вебзастосунок виконує основну задачу склеювання гетерогенних відеофрагментів із попередньою нормалізацією технічних параметрів. Проте під час реалізації та тестування було виявлено кілька напрямів, які можуть бути розвинені в майбутньому для підвищення зручності, стабільності й функціональності системи.

Першим перспективним кроком є впровадження функціоналу обрізання (trim) відеофрагментів перед склеюванням. У багатьох сценаріях користувач не потребує об’єднання повного відео, а лише його частин. Надання можливості задавати часові діапазони для кожного фрагмента дозволить створювати більш точні та адаптовані результати без потреби у зовнішньому монтажі.

Іншим напрямом розвитку є автоматичне виявлення дублікатів або повторюваних кадрів. У випадках, коли користувач випадково завантажив однакові файли, система могла б сигналізувати про це або запропонувати виключити зайве. Це особливо актуально при роботі з великою кількістю коротких відео, наприклад, записів із камер спостереження.

Також доцільним є впровадження модуля автоматичної кольорокорекції. Фрагменти відео, зняті в різних умовах освітлення або на

різні пристрої, можуть суттєво відрізнятись за яскравістю, контрастністю та балансом білого. Додавання функції, що аналізує колірний профіль і вирівнює основні параметри зображення, дозволить зробити результат візуально цілісніше і привабливішим.

У контексті масштабованості цікавим напрямом є інтеграція з хмарними середовищами обробки (наприклад, AWS Lambda, Cloud Run або подібні). Це дозволить переносити обчислювальні навантаження з локального сервера на більш гнучку та масштабовану інфраструктуру, зменшуючи час обробки та підвищуючи стабільність при великій кількості користувачів.

З технічної точки зору також розглядається можливість збереження історії обробок або профілів налаштувань для повторного використання. Це дозволить постійним користувачам зберігати улюблені шаблони або швидко повторювати попередні дії без повторного налаштування параметрів.

Ще однією цікавою ідеєю є додавання режиму попереднього перегляду результату перед склеюванням. Це дозволить користувачу побачити, як виглядатиме фінальний файл, оцінити вирівнювання фрагментів, наявність чорних полів або потенційних спотворень, і за потреби змінити модель уніфікації.

З боку інтерфейсу можна розширити функціонал клієнтської частини: додати індикатори прогресу обробки, відображення технічних характеристик файлів у зручному вигляді, завантаження декількох файлів drag-and-drop або інтеграцію з хмарними сховищами для завантаження відео напряму.

Таким чином, розроблений застосунок має потенціал до подальшого розвитку як в технічному, так і функціональному плані. Деякі з можливих напрямків удосконалення включає в себе розширення підтримки відеоформатів, інтеграцію індикатора виконання, або ж можливість попереднього перегляду об’єднаного результату перед збереженням.

Крім того, можливо розглянути оптимізацію взаємодії з користувачем. Наприклад, спрощення інтерфейсу, або адаптація під мобільні пристрої, що підвищить зручність при роботі.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було розроблено вебзастосунок для автоматизованої склейки відеофрагментів із різними технічними параметрами. Рішення поєднує сучасні підходи до обробки відео, клієнт-серверну архітектуру, а також зручний інтерфейс для користувача, який не має спеціальних технічних знань.

У процесі реалізації проведено детальний аналіз існуючих інструментів професійних редакторів, командних утиліт і онлайн-сервісів, що дозволило обґрунтовано сформулювати вимоги до майбутнього застосунку. Було розглянуто різні моделі нормалізації відео та запропоновано адаптивний підхід до приведення фрагментів до єдиного формату.

Серверну частину застосунку реалізовано на Node.js із використанням бібліотеки FFmpeg для обробки відео, а клієнтську, на фреймворку Next.js із використанням мови TypeScript. Було впроваджено механізми завантаження відеофрагментів, аналізу їх параметрів, нормалізації та формування цілісного відеофайлу у форматі MP4.

Застосунок дозволяє об’єднувати відео з різною роздільною здатністю, орієнтацією, частотою кадрів та форматом збереження. Весь процес є автоматизованим, інтуїтивно зрозумілим і не потребує встановлення додаткового програмного застосунку. Результат формується на сервері, а користувач отримує посилання на готовий файл. Система протестована на різних конфігураціях вхідних даних і продемонструвала стабільну роботу та високу якість обробки.

Розроблений вебзастосунок може бути використаний у сфері освіти, створення відеоконтенту, цифрових презентацій і в корпоративному середовищі, де потрібна швидка обробка гетерогенних відео.

У роботі також окреслено можливі напрями подальшого розвитку: реалізація черг обробки, підтримка додаткових форматів і кодеків,

масштабування у хмарному середовищі, впровадження акаунтів користувачів та історії проєктів.

Узагальнюючи, виконання цієї роботи дозволило на практиці застосувати знання в галузях веброзробки, обробки відео, оптимізації UI/UX та побудови продуктивних серверних рішень. Проєкт демонструє, як сучасні вебтехнології можуть забезпечити просту, надійну та доступну автоматизацію складних мультимедійних процесів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cisco. Visual Networking Index: Forecast and Trends, 2017–2022. URL: https://www.cisco.com/c/en/us/solutions/collateral/serviceprovider/visualnetworking-index-vni/white-paper-c11-741490.html (дата звернення 07.04.2025).

2. Statista. Number of videos uploaded to YouTube per minute 2023. URL: https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/ (дата звернення 07.04.2025)

3. Adobe. Premiere Pro – Industry-standard video editing software. URL: https://www.adobe.com/products/premiere.html (дата звернення 07.04.2025)

4. Blackmagic Design. DaVinci Resolve 18 – Professional Editing, Color,

5. Effects and Audio Post. URL: https://www.blackmagicdesign.com/products/davinciresolve (дата звернення 11.04.2025)

6. Shotcut. Free, Open-source, Cross-platform Video Editor. URL:https://shotcut.org/ (дата звернення 12.04.2025)

7. ffmpeg. Documentation – Video Encoding, Muxing, and Filters. URL: https://ffmpeg.org/documentation.html (дата звернення 13.04.2025)

8. Kapwing. Collaborative Online Video Editor. URL: https://www.kapwing.com/ (дата звернення 14.04.2025)

9. VEED.io. Online Video Editor. URL: https://www.veed.io/ (дата звернення 17.04.2025)

10. Clideo. All-in-one online video tools. URL: https://clideo.com/ (дата звернення 17.04.2025)

11. ffmpeg Wiki. Heterogeneous media merging (Concat, Re-encoding, etc.). URL: https://trac.ffmpeg.org/wiki/HowToConcatenate (дата звернення 18.04.2025)

12. REST API Tutorial. RESTful API Design and Best Practices. URL: https://restfulapi.net/ (дата звернення 20.04.2025)

13. Mashtalir, S., Mashtalir, V. (2016) Sequential temporal video segmentation via spatial image partitions. Proceedings AVSS.

14. Mashtalir, S., Mashtalir, V. (2020) Spatio-Temporal Video Segmentation.

15. Mashtalir S., Lendel, D. (2024) Video Fragment Processing by Ky Fan Norm

16. Mashtalir S., Lendel, D. (2024) Video Pre-Motion Detection by Fragment Processing.

17. MPEG. The Moving Picture Experts Group – Standards for video and audio compression. URL: https://mpeg.chiariglione.org/ (дата звернення 22.04.2025)

18. WebM Project. Open, royalty-free media file format for the web. URL: https://www.webmproject.org/ (дата звернення 23.04.2025)

19. Apple Developer. AVFoundation Framework – Working with audiovisual media on Apple platforms. URL: https://developer.apple.com/documentation/avfoundation (дата звернення 24.04.2025)

20. Microsoft. AVI RIFF File Reference – Technical documentation for handling AVI files. URL: https://learn.microsoft.com/en-us/windows/win32/directshow/avi-riff-file-reference (дата звернення 25.04.2025)

21. Matroska. Technical specifications for the Matroska multimedia container format. URL: https://www.matroska.org/technical/elements.html (дата звернення 30.04.2025)

22. Wikipedia. High Efficiency Video Coding. URL: https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding (дата звернення 01.05.2025)

23. Mozilla. JavaScript Documentation – Comprehensive reference for JavaScript language and APIs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 03.05.2025)

24. TypeScript. Official Documentation – Type-safe superset of JavaScript. URL: https://www.typescriptlang.org/docs/ (дата звернення 05.05.2025)

25. NestJS. Official Documentation – A progressive Node.js framework for building scalable applications. URL: https://docs.nestjs.com/ (дата звернення 05.05.2025)

26. Next.js. Documentation – The React Framework for Production.. URL: https://nextjs.org/docs (дата звернення 06.05.2025)

27. fluent-ffmpeg. A fluent API to FFmpeg for Node.js. URL: https://github.com/fluent-ffmpeg/node-fluent-ffmpeg (дата звернення 07.05.2025)

28. ffmpeg. ffprobe Documentation – Tool for media stream analysis. URL: https://ffmpeg.org/ffprobe.html (дата звернення 11.05.2025)

29. Node.js. Stream API Documentation – Working with streaming data in URL: https://nodejs.org/api/stream.html (дата звернення 11.05.2025)

30. Mozilla. Using Fetch – Guide to the Fetch API for network requests. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення 11.05.2025)

31. Jest. A delightful JavaScript testing framework. URL: https://jestjs.io/docs/getting-started (дата звернення 11.05.2025)