Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБ-СЕРВІСУ ДЛЯ КОНВЕРТАЦІЇ ФАЙЛІВ ТА ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ

DEVELOPMENT OF A WEB SERVICE FOR FILE CONVERSION AND DIGITAL IMAGE PROCESSING

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-42
Мещерягіна Олександра Костянтинівна
_____
(підпис)
Керівник: асистент
Сачук Вікторія Олегівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«__» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович
_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«16» січня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Мещерягіна Олександра Костянтинівна
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка веб-сервісу для конвертації файлів та цифрової обробки зображень»
Керівник асистент Сачук Вікторія Олегівна
затверджені наказом закладу вищої освіти від «16» січня 2026 р. № 32/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «02» червня 2026 р.
3. Вихідні дані до роботи: опублікувати вітчизняні дослідження в сфері організації вебсервісів з конвертування та обробки цифрових файлів, технології веброзробки, принципи побудови клієнт-серверних систем, методи обробки цифрових файлів, безпеки, тестування
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповненння об'єкта проєктування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проєктованої системи, функціонально-структурна схема роботи об'єкта проектування

5. Перелік графічного матеріалу: 14 рисунків, презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Сачук В. О.*		
Теоретичне дослідження	*Сачук В. О.*		
Практична реалізація	*Сачук В. О.*		
Спеціальні розрахунки	*Сачук В. О.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Тулашвілі Ю. Й.*		

7. Дата видачі завдання *«16» січня 2026 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 17.02.2026 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 28.02.2026 р.*	
3	*Розробити функціональну схему роботи програмного продукту*	*до 12.03.2026 р.*	
4	*Описати засоби розробки об'єкта проектування*	*до 26.03.2026 р.*	
5	*Практична реалізація об'єкта проектування*	*до 30.04.2026 р.*	
6	*Провести спеціальні розрахунки*	*до 18.05.2026 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 02.06.2026 р.*	

Здобувач вищої освіти _____ Олександра МЕЩЕРЯГІНА

Керівник роботи _____ Вікторія САЧУК

АНОТАЦІЯ

Мещерягіна О. К. Розробка веб-сервісу для конвертації файлів та цифрової обробки зображень. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків, присвячена проєктуванню та розробці системи для обробки цифрових файлів. У межах початкового аналітичного етапу було виконано аналіз предметної області, наведено існуючі аналоги розробки та визначено основні засоби проєктування вебсервісу. Проєктування системи базується на специфікації вимог до розроблюваного сайту, обґрунтуванні вибору шляхів, технологій і засобів вирішення поставленого завдання, а також описується функціонально-структурна схема роботи об'єкта, разом із розробкою моделі бази даних. Технічна реалізація вебсервісу, клієнтської та серверної частини, здійснювалась за допомогою мови JavaScript, мови розмітки HTML та каскадної таблиці стилів CSS, що дозволило реалізувати процес конвертації, дії із файлами та цифрову обробку зображень, разом із інтеграцією бази даних на основі SQLite. Було проведено опис тестування та налагодження, разом із інструкцією для користування сервісом. Результати проведеного дослідження узагальнені у висновках, що підтверджують потребу та ефективність у сфері обробки цифрових файлів.

Ключові слова: вебсервіс, конвертація, JavaScript, SQLite, HTTPS, база даних, обробка цифрових файлів.

ANNOTATION

Oleksandra Meshcheriahina. Development of a web service for file conversion and digital image processing. Manuscript.

Bachelor's qualification paper in the study program «Computer Science». Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification paper consists of an introduction, four chapters, a conclusion, a list of references, and appendices, and is devoted to the design and development of a system for processing digital files. During the initial analytical phase, a domain analysis was performed, existing analogues of the development were presented, and the main tools for designing the web service were identified. The system design is based on the requirements specification for the website under development, a justification of the chosen approaches, technologies, and tools for solving the task at hand, and a description of the functional and structural diagram of the system's operation, along with the development of a database model. The technical implementation of the web service, including the client-side and server-side components, was carried out using JavaScript, HTML, and CSS, which enabled the implementation of the conversion process, file operations, and digital image processing, along with the integration of an SQLite-based database. A description of testing and debugging was provided, along with instructions for using the service. The results of the study are summarized in the conclusion, which confirms the need for and effectiveness of digital file processing.

Keywords: web service, conversion, JavaScript, SQLite, HTTPS, database, digital file processing.

ЗМІСТ

ВСТУП

В середовищі технологій щоденно зростає об’єм та кількість інформації, яку необхідно обробляти. Користувачі працюють з великою кількістю даних, що створює потребу у зручних інструментах їх обробки. Однією з поширених проблем є несумісність форматів документів та зображень, а також необхідність використання окремих програм для кожного типу завдання, що ускладнює робочі процеси та знижує продуктивність для масових користувачів. Крім того, для певних груп людей встановлення спеціалізованого програмного забезпечення є малоефективним для виконання лише однієї типової задачі, а також це часто вимагає додаткових ресурсів пристрою та часу на налаштування. У зв’язку з цим виникає потреба у створенні універсальних вебсервісів, які дозволятимуть виконувати низку процесів швидко та з найліпшим підходом.

Метою кваліфікаційної роботи бакалавра є розробка вебсервісу для конвертування файлів та базової обробки зображень, що враховує інтуїтивний дизайн, базові вимоги користувачів та швидкість процесів обробки.

Об’єктом дослідження є процес обробки цифрових зображень та конвертації файлів у вебсередовищі.

Предметом дослідження є методи, засоби та рішення для створення вебсервісу конвертації та обробки цифрових файлів.

Для досягнення поставленої мети роботи необхідно вирішити такі завдання:

– проаналізувати існуючі вебсервіси із подібним до розробки підходом, де в головний пріоритет стануть сайти для обробки зображень та конвертування файлів;

– сформулювати та визначити основні функціональні вимоги до розроблюваної системи;

– спроєктувати архітектуру вебсервісу з урахуванням гармонійної клієнтської та серверної взаємодії;

– розробити механізми завантаження, обробки та конвертації файлів, з урахуванням забезпечення стабільності серверу;

– реалізувати функції цифрової обробки зображень;

– побудувати безпеку, як для сайту, так і для користувачів, обробки та зберігання завантажених файлів;

– протестувати роботу веб-сервісу з подальшою оптимізацією його функціональності.

Результат проведеного дослідження та розробка були апробовані, описані та опубліковані у міжнародній науковій конференції «Сучасні інформаційні технології: від теорії до практики» (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану проблеми

Значна частина діяльності користувачів, в реальний час, пов’язана з обробкою цифрових файлів різних форматів. Документи, зображення, презентації та мультимедійні дані широко використовуються в освітній, професійній та повсякденній діяльності. У зв’язку з цим виникає необхідність конвертування файлів з одного формату в інший, оскільки поставлені задачі змінюються, а також є необхідність виконання базових операцій редагування зображень, які є частиною багатьох документів. Виходячи з даної проблеми, можна припустити, що певний відсоток користувачів стискаються з такою необхідністю не постійно, що призводить до не потреби встановлення спеціалізованого програмного забезпечення.

Вже існує та активно підтримується велика кількість програмних засобів і вебсервісів, що надають можливість конвертації файлів і обробки зображень у режимі онлайн. Такі рішення дозволяють користувачам виконувати необхідні операції без прив’язки до конкретного пристрою, використовуючи лише браузер і доступ до мережі Інтернет. Низка інструментів, зібрана з багатої кількості різнопрофільних застосунків, вже занесена до купи в один вебресурс – сайт [1].

Перед постановкою недоліків варто визначити об’єкти дослідженя.

Конвертація даних – це перетворення даних з одного формату в інший, що супроводжується збереженням основного логічно-структурного змісту інформації. У разі зміни будь-якого з елементів роботи, дані мають бути конвертовані, перш ніж їх можна буде використати в іншій області [2].

Обробка зображень – будь-яка форма обробки інформації, для якої вхідні дані представлені цифровим зображенням. Цифрова обробка зображень полягає у представленні зображень у вигляді даних, які можуть бути оброблені комп’ютером, з використанням математичних та логічних операцій [3].

Попри значну кількість існуючих рішень, стан проблеми характеризується низкою недоліків. По-перше, багато веб-сервісів мають обмежений функціонал, зосереджуючись лише на конвертації файлів або лише на обробці зображень, що змушує користувачів звертатися до кількох різних платформ. Такі сервіси не є недоліком, оскільки вони грають роль «замінника» застосунку, але, водночас, вони є проблемою питання, яке розглядається у багатофункціональному колі. По-друге, вебсервіси із невеликою базою перевантажені рекламою або містять складний інтерфейс, що ускладнює використання користувачем. Такий підхід ігнорує користувацький досвід, внаслідок чого жодна із сторін не здобуває користі [4]. По-третє, важливим аспектом залишається питання безпеки, оскільки користувачі змушені завантажувати власні файли на сторонні сервери. Це не тільки ставить довіру під сумніви, а й створює можливість ризиків витоку або несанкціонованого доступу до даних [5].

Повертаючись до питання користувацького досвіду, існують обмеження щодо розміру файлів, кількості операцій або швидкості обробки, що негативно впливає на ефективність використання таких сервісів [6]. Необхідно проаналізувати та скласти ідеальне рішення щодо всіх обмежень, створюючи баланс між комфортом та ефективністю. Особливо це актуально при роботі з великими зображеннями або документами складної структури.

Обробка зображень важлива, оскільки їх кількість складає майже половину ресурсів інтернету, як підкреслюється у статті, «значна кількість зображень передається через Інтернет у вигляді мініатюр, публікацій у соціальних мережах та інших елементів веб-сайтів. Зазвичай саме зображення займають найбільшу частину обсягу веб-сторінок, складаючи від 24 % до 58 % від загального розміру» [7]. Також в статті вказано, що «традиційні формати стиснення зображень без втрат, такі як PNG, дозволяють досягти коефіцієнта стиснення до зменшення пропускної здатності на 66 %, тоді як формати зі стисненням із втратами, такі як JPEG і WebP, дозволяють досягти зменшення пропускної здатності на 95 %» [7].

Підсумовуючи вищенаписане, можна сказати, що існуючі вебсервіси для конвертації файлів та обробки зображень мають низку суттєвих обмежень. Серед основних проблем є розділення функціоналу між різними платформами, перевантаженість інтерфейсів рекламою, а також обмеження щодо розміру файлів, швидкості обробки та кількості операцій, що знижує ефективність використання таких сервісів. Окремо виділяються питання безпеки, оскільки завантаження файлів на сторонні сервери створює ризики витоку даних і втрати контролю над інформацією. Внаслідок цього виникає потреба створення нового вебсервісу, який задовольнить всі потреби.

1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напряму та вибір методів рішення

Аналіз аналогічних вебсервісів є необхідним етапом у процесі розробки продукту, оскільки він дозволяє оцінити сучасний стан ринку та визначити основні тенденції у відповідній предметній області. Проведення такого аналізу дає змогу виявити існуючі рішення, їх функціональні можливості, а також сильні й слабкі сторони, що є необхідним етапом для обґрунтування доцільності створення нового веб-сервісу.

Першим об’єктом для аналізу стануть сервіси для конвертування файлів, серед них варто визначити онлайн-конвертер Zamzar [8], що підтримує понад 1200 типів форматів для обробки, а також має простий та інтуїтивний інтерфейс. Даний сайт має великий мінус, який характеризується необхідністю створення акаунту для завантаження конвертованих файлів, що в сучасному світі відлякує потенційних користувачів. Також безкоштовний тариф обмежується певною кількістю конвертацій на день, а розмір кожного файлу не має бути більше 1 МБ, що в свою чергу є занадто малим об’ємом для звичайних повсякденних документів. Такий тип сервісів націлений на утримування старих користувачів, які підтримують роботу сайту підписками, а також мають доступ до великої кількості форматів. Сайт створено у 2006 році, але він все одно залишається

популярним у списку аналогів, тому наявність застарілого дизайну та невигідних умов безкоштовного плану нівелюється утримуванням бази користувачів.

Останнім для розгляду об’єктом лишається CloudConvert [9], у порівнянні із попередніми інструментами, даний вебсервіс націлений на професійний підхід, він орієнтований на користувачів, що потребують розширених можливостей налаштування та інтеграції з іншими системами. На відміну від простих онлайн-конвертерів, він надає більш гнучкий контроль над параметрами обробки файлів. Вебсервіс дає вибір серед 200 форматів, має сертифікований захист даних, співпрацює із постачальниками програмного забезпечення для досягнення кращих результатів, а також надає можливість детального налаштування параметрів конвертації, таких як якість, розмір файлу, кодеки та інші технічні характеристики. Тут присутня наявність API для розробників, що дозволяє інтегрувати функції конвертації у сторонні веб-додатки та системи. Ліміт конвертацій в день складає всього 10 файлів, але сервіс нівелює таку можливість двома типами платної моделі – одноразова та підписка. Одноразова оплата передбачає купівлю необхідної кількості конвертацій, а підписка надає ширший спектр додаткових можливостей. Функціонал сайту також пропонує створення архівів, стиснення розміру певних типів файлів, а також їх об’єднання. Недоліком є те, що потребується більше розуміння параметрів обробки файлів.

Другим об’єктом для аналізу стане сервіс для обробки цифрових зображень. При завантаженні головної сторінки iLoveIMG [10] користувачу одразу пропонуються всі популярні та часто необхідні дії до файлів, що включає стиснення зображення, обрізка, зміна розміру, конвертація, тощо. Інтерфейс є простим та інтуїтивним, зосередженим на звичайного користувача. Сервіс також має підтримку масових операцій, а також роботу зі штучним інтелектом. Платна підписка сервісу надає доступ до інструментів ШІ, додаткових функцій сайту, а також збільшений обсяг операцій. Підписка не нав’язана, оскільки всі необхідні функції є безкоштовними, що передбачає позитивний користувацький досвід. Також варто зазначити ще один плюс даного вебсервісу – створення зображення будь-якого сайту. Сайт позиціонує себе як частину великої екосистеми, де кожна

окрема частинка відповідає своїй області роботи, це гарний аспект того, що зайвих функцій на сайті присутнім не буде, а відповідно – користувач не плутатиметься з інструментом.

Узагальнюючи аналіз аналогів можна зробити висновок, що існуючі веб-сервіси забезпечують достатній рівень базової функціональності, однак не всі з них поєднують простоту використання, універсального підходу до роботи з файлами, оскільки кожен сервіс спеціалізується на окремому типі задач, та розширені можливості в одному рішенні. Це підтверджує актуальність розробки нового веб-сервісу, який би об’єднував переваги існуючих систем та усував їх недоліки.

1.3 Аналіз і вибір засобів проектування вебсервісу

Від обраних технологій, інструментів та підходів залежить ефективність, продуктивність і масштабованість майбутньої системи. На етапі вибору засобів проєктування здійснюється дослідження актуальних програмних засобів, що використовуються для створення вебсайтів, а також визначення їх відповідності вимогам проєкту. При цьому враховуються як функціональні можливості інструментів, так і їх нефункціональні характеристики, зокрема швидкодія, надійність, безпека, зручність використання та можливість інтеграції з іншими сервісами. Врахування аналізу сумісності обраних технологій є доречним, оскільки даний підхід забезпечує стабільну взаємодію клієнтської та серверної частини.

Для реалізації поставленої мети було розроблено ядро вебсервісу за допомогою React, HTML, CSS, Node.js в парі із Express, JavaScript, HTTPS а також SQLite.

React [11] є клієнтською частиною розроблюваного вебсервісу та базується на компонентному підході, що передбачає розбиття інтерфейсу на незалежні частини – компоненти. Кожен компонент відповідає за окрему частину інтерфейсу, що спрощує розробку, тестування та підтримку коду. Бібліотека

використовує механізм, який дозволяє оновлювати без перезавантаження лише ті елементи сторінки, що змінилися. Завдяки зручній структурі коду та сумісності з іншими бібліотеками цей вибір стає найвигіднішим.

Мова розмітки HTML та каскадна таблиця стилів CSS у своєму поєднанні використовуються для створення структури інтерфейсу, зокрема форм завантаження файлів, кнопок взаємодії та інформаційних блоків, а також оформлюють інтерфейс, та забезпечують зручність використання. У парі вони забезпечують основу для побудови користувацького інтерфейсу вебсервісу.

JavaScript [12] дозволяє реалізувати динамічну поведінку вебсторінок, забезпечуючи взаємодію користувача з інтерфейсом у режимі реального часу. Завдяки цій мові можна обробляти події та здійснювати обмін даними з сервером.

Node.js [13] у поєднанні з Express [14] є популярним рішенням для створення серверної частини вебсторінок. Такий підхід дозволяє використовувати мову програмування JavaScript як на клієнтській, так і на серверній стороні, що значно спрощує процес розробки. Node.js – це середовище виконання JavaScript, яке працює поза браузером і дозволяє створювати серверні додатки. Воно базується на подієво-орієнтованій моделі вводу та виводу, що забезпечує високу продуктивність та обробку великої кількості одночасних запитів. Express є легким і гнучким фреймворком для Node.js, який спрощує розробку серверної логіки. Він надає готові інструменти для маршрутизації, обробки запитів і створення API.

HTTPS – це захищена версія протоколу HTTP, яка використовується для передачі даних між клієнтом і сервером у вебсередовищі. Основною особливістю HTTPS є використання криптографічного шифрування задля безпечної передачі даних та захисту від їх перехоплення. HTTPS базується на використанні протоколу Transport Layer Security, який надає шифрування даних під час передачі, перевірку цілісності даних, а також автентифікацію сервера через цифрові сертифікати.

SQLite [15] є реляційною системою керування базами даних, яка не потребує окремого серверного програмного забезпечення і працює всередині застосунку. На відміну від таких систем, як MySQL, вона не запускає окремий серверний процес, а зберігає всі дані у вигляді одного файлу на диску. Система підтримує мову запитів SQL, що дозволяє виконувати основні операції з даними, зокрема додавання, редагування, видалення та вибірку інформації.

Допоміжними програмними засобами стали бібліотеки, які реалізують потреби поставленої мети, що враховує конвертацію файлів, обробку зображень, захист від перевантаження серверу, маршрутизацію та роботу із запитами.

Використана у проєкті Sharp являє собою високопродуктивну бібліотеку для обробки зображень у середовищі Node.js, яка базується на нативній бібліотеці libvips. Вона використовується для обробки графічних файлів на серверній стороні без необхідності застосування сторонніх графічних редакторів. Завдяки наданому базовому набору трансформацій та оптимізацій файлів вона є ідеальним інструментом у побудові вебсервісу.

Бібліотека LibreOffice – це офісний пакет із відкритим вихідним кодом, який використовується для створення, редагування та конвертації офісних документів. Він є одним із найпоширеніших альтернативних рішень офісним пакетам і застосовується як у локальних, так і в серверних рішеннях.

Також варто зазначити бібліотеки, які не є основними для проєкту, але без них робота вебсервісу може бути порушена. express-rate-limit – це проміжне програмне забезпечення для Node.js та Express, яке використовується для обмеження кількості запитів від одного користувача за певний проміжок часу. Його основною метою є захист вебсервісу від надмірного навантаження, зловмисних атак та неконтрольованого використання API. react-router-dom – це бібліотека для реалізації маршрутизації у веб-додатках, створених на основі React. Вона дозволяє організувати навігацію між різними сторінками або компонентами без повного перезавантаження сторінки. Axios забезпечує широкий набір функцій для роботи з HTTP-запитами, а саме:

– виконання запитів типу GET, POST, PUT, DELETE;

– передачу та отримання даних у форматі JSON;

– підтримку асинхронних операцій;

– обробку помилок запитів.

CORS – це механізм безпеки в браузерах, який контролює доступ веб-додатків до ресурсів, розташованих на іншому домені, протоколі або порту. Він визначає, чи може клієнтська частина вебсервісу звертатися до серверу, який знаходиться поза межами поточного походження, а також дозволяє серверу явно вказати, яким джерелам дозволено отримувати доступ до ресурсів, тобто до файлів та зображень користувача.

Helmet – це бібліотека для базового захисту серверів, він приховує технології, забороняє будь-яким іншим сайтам зі сторони вбудовувати ваш проєкт у фрейми, не дозволяє «вгадування» типу файлів та уникає міжсайтового скриптингу.

Multer є проміжним шаром, який забезпечує коректну обробку завантажених користувачем файлів. Він розділяє файл і текстові поля, зберігає файл у папку на сервері, після чого дозволяє передавати файли від клієнта до серверної логіки для подальшої обробки.

Середою для розробки було обрано Visual Studio Code, який є легким редактором вихідного коду, розроблений компанією Microsoft. Він задовільняє потреби для створення вебсервісу, оскільки працює із всіма вищезазначеними мовами програмування та бібліотеками, а також надає можливість встановлення розширень для полегшення розробки. Завдяки сучасному та простому середовищу розробка пришвидшилася у декілька разів.

Всі макети сторінок були розроблені за допомогою хмарного інструменту Figma. Він широко використовується серед розробників та є універсальним рішенням для розробки макетів сайтів. За допомогою Figma було створено макет головної сторінки, сторінки завантаження та обробки файлів, сторінку для створення палітри по фото, інтерфейс для редагування фото, сторінку для розділення фото по частинах, інтерфейс результатів конвертації та прототипи переходів між цими сторінками.

1.4 Постановка завдання на кваліфікаційну роботу бакалавра

Основною метою даної кваліфікаційної роботи є створення вебсервісу, завдяки якому буде реалізована можливість конвертації файлів різних типів, а також цифрова обробка зображень. Актуальність дослідження визначається необхідністю розробки максимально простого та швидкого інструменту для виконання повсякденних задач у цифровому та реальному світі, тим самим спрощуючи роботу звичайним користувачам, які матимуть доступ до цих інструментів онлайн, без опрацювання багатьох важких програмних забезпечень.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз сучасних проблем в області обробки цифрових файлів із опрацюванням недоліків та їх вирішення;

– визначити та опрацювати вимоги до вебсервісу, які відповідатимуть сучасним потребам користувачів та матимуть високий показник користувацького досвіду;

– розробити архітектуру системи, яка забезпечує гармонійну взаємодію клієнтської та серверної частини;

– обґрунтувати методи опрацювання даних, які відповідають вимогам до роботи із цифровими файлами;

– забезпечити інтуїтивний та легкий користувацький інтерфейс, який посилатиметься на максимальний комфорт у використанні користувачем;

– оцінити результати розробки тестуванням та визначити можливі варіанти реалізації для подальшої життєдіяльності сервісу.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

У процесі розробки систем важливим етапом є обґрунтований вибір програмних засобів, технологій та інструментів, які використовуються для реалізації проєкту. Внаслідок правильного вибору впроваджується ефективність розробки, продуктивність роботи системи та зручність її подальшого супроводу. У межах даного підрозділу обґрунтовується вибір засобів розробки, які були використані під час створення вебсервісу. Метою даного розділу є розкриття причин вибору конкретних технологій, їх переваг у контексті поставлених завдань, а також визначення їх відповідності вимогам розроблюваної системи.

React було обрано для побудови інтерфейсу користувача завдяки його компонентному підходу. Він дозволяє:

– створювати повторно використовувані UI-компоненти;

– забезпечує швидке оновлення інтерфейсу без перезавантаження сторінки;

– має ефективну роботу з станом.

Порівнянно з гіршими варіантами, які складніше масштабуються, переважно дублюють код, мають застарілий підхід, та мають погану архітектуру для систем, React є найкращим варіантом серед своїх конкурентів.

Для серверної логіки було обрано саме Node.js у парі з Express, оскільки вони:

– використовують одну мову програмування;

– мають високу продуктивність за рахунок асинхронної моделі;

– не мають складнощів із створенням REST API.

Аналоги в той самий час мають низку недоліків у вигляді повільнішого циклу розробки для задач у режимі реального часу, а також мають складнішу конфігурацію і більшу вагу системи.

Оскільки HTTP не забезпечує захисту даних і є неприйнятним для систем, що працюють з файлами користувачів, то на допомогу приходить «тунелювання» із використанням HTTPS. «Тунелювання» – це сучасний підхід, який дозволяє безпечно відкривати локальний або внутрішній сервер без прямого відкриття портів на роутері. Воно створює захищений «міст» між локальним сервером і зовнішнім Інтернетом, через який проходить весь трафік. Замість того, щоб відкривати порти, налаштовувати firewall та купувати сервер, такий спосіб просто довзоляє отримати публічний HTTPS URL, завдяки чому сервер немає відкритого доступу до IP, зменшується ризик атак, використовує SSL сертифікат та шифрує трафік. LocalTunnel – це інструмент, який дозволяє відкрити локальний вебсервер у публічному інтернеті без складного налаштування. Він не потребує реєстрації акаунта, налаштування серверів та конфігурації доменів, а також після запуску система отримує публічну HTTPS-адресу.

Sharp є однією з найефективніших бібліотек для обробки зображень у середовищі Node.js, і її вибір для вебсервісу конвертації файлів є обґрунтованим, оскільки:

– він працює на базі нативної бібліотеки libvips;

– не завантажує повністю зображення в пам’ять;

– використовує потокову обробку;

– легко інтегрується в стек Node.js.

На відміну від аналогів, які залежні від системних бібліотек, мають велике навантаження на центральний процесор та повільніші у Node.js сценаріях, Sharp є найвигіднішою позицією для розробки вебсервісу.

Для конвертації файлів у вебсервісі було обрано саме бібліотеку LibreOffice, яка є вільним офісним пакетом із відкритим вихідним кодом, який включає інструменти для роботи з документами. Дана бібліотека підтримує гарний вибір офісних форматів:

– DOCX;

– PDF;

– XLSX;

– PPT та PPTX.

Таким чином, обраний технологічний стек є оптимальним для реалізації вебсервісу, оскільки поєднує сучасні, продуктивні та безкоштовні рішення, які надають все необхідне для роботи повного циклу роботи системи – від інтерфейсу користувача до серверної обробки файлів. Використані технології взаємно доповнюють одна одну, що сприятливо відображається на роботі вебсервісу.

2.2 Функціонально-структурна схема роботи об’єкта проектування (блок-схеми алгоритмів, UML діаграми класів, діаграми компонентів)

Функціонально-структурна схема роботи об’єкта проєктування дозволяє представити загальну логіку функціонування системи, її основні компоненти та взаємозв’язки між ними. Така схема відображає, як саме організована робота вебсервісу, які модулі входять до його складу та як відбувається передача даних між ними. Завдяки схемам розробка пришвидшується у декілька раз та з’являється можливість побачити створену логіку розробки.

UML-діаграма класів – це представлення структури системи, яке показує, з яких класів вона складається, які в них є властивості, методи і як ці класи пов’язані між собою. Вона дозволяє наочно показати класи, їх атрибути, методи та взаємозв’язки між ними. Діаграма класів допомагає зрозуміти архітектуру програмного забезпечення ще до етапу реалізації, а також спрощує процес проєктування, розробки та подальшої підтримки системи. Backend об’єкту буде написано у процедурному стилі на базі фреймворку Express.js, а фронтенд базується на React, який використовує функціональні компоненти, тому доцільно буде побудувати концептуальну діаграму.

На рисунку 2.1 зображено концептуальну UML-діаграму із п’ятьма компонентами системи, де «private» є приватним доступом, а «public» – публічним.

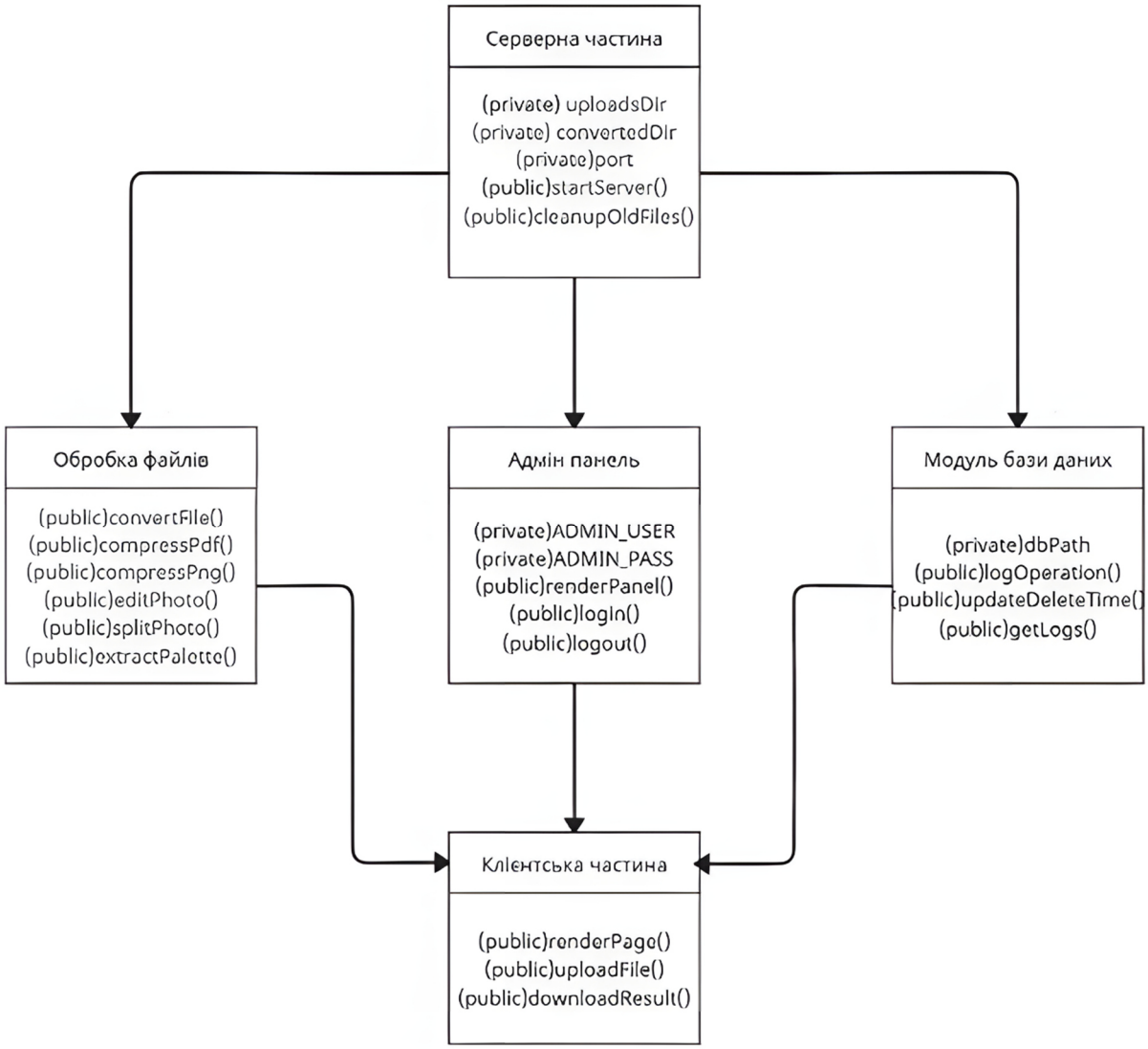

Рисунок 2.1 – Концептуальна UML-діаграма

Джерело: розроблено автором

Серверна частина відповідає за налаштування Express, ініціалізацію папок, запуск системи очищення файлів та прослуховування порту. Модуль бази даних приховує внутрішню реалізацію і дає контрольований доступ разом із SQLite, а також містить методи для запису нових логів та оновлює їх.

Блок обробки файлів містить бізнес-логіку програми, що включає обробники для використання бібліотек sharp, pdf-lib та libreoffice-convert для маніпуляцій з файлами.

Адмін-панель відповідає за автентифікацію та її показ. Блок клієнтської частини взаємодіє з сервером через HTTP-запити для відправки файлів та отримання результатів.

UML-діаграма компонентів використовується для відображення структури програмної системи на рівні її компонентів та зв’язків між ними, що показано на рисунку 2.2.

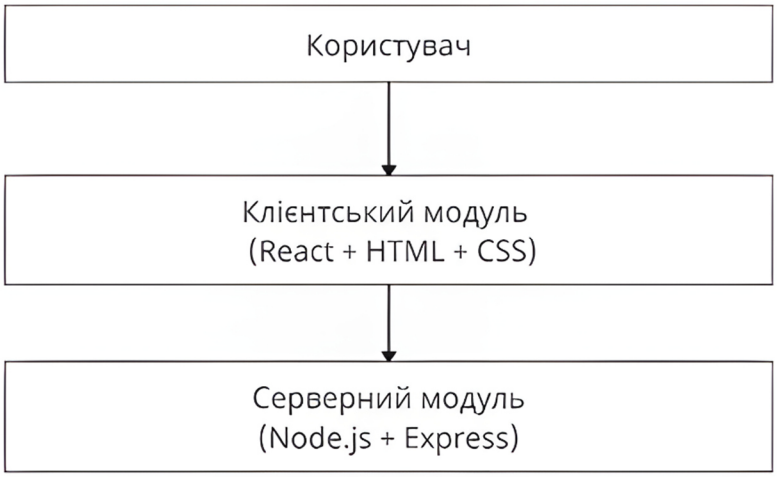

Рисунок 2.2 – UML-діаграма компонентів

Джерело: розроблено автором

На рисунку 2.3 розглядається процес роботи тунелювання, тобто процес заходу користувача на сайт, а також як працює серверна частина у даному випадку.

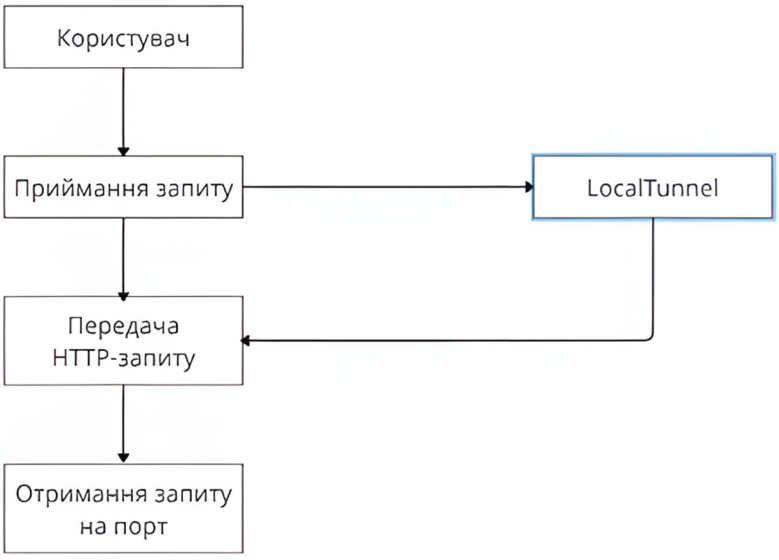

Рисунок 2.3 – Функціональна схема роботи тунелювання

Джерело: розроблено автором

Процес починається з того, що користувач переходить за посиланням на сайт, де його браузер робить DNS-запит. Сервери Localtunnel приймають цей запит і забезпечують перевірку безпеки. Сервер обробки Localtunnel шукає у своїй базі даних, який клієнт зараз тримає до нього підключений тунель з ім’ям домену, після чого сервер передає HTTP-запит всередину комп’ютера адміністратора. Тунельний клієнт на комп’ютері адміністратора отримує запит на локальний порт.

На рисунку 2.4 показано функціональну схему, в якій описано процес конвертації файлу у вебсервісі.

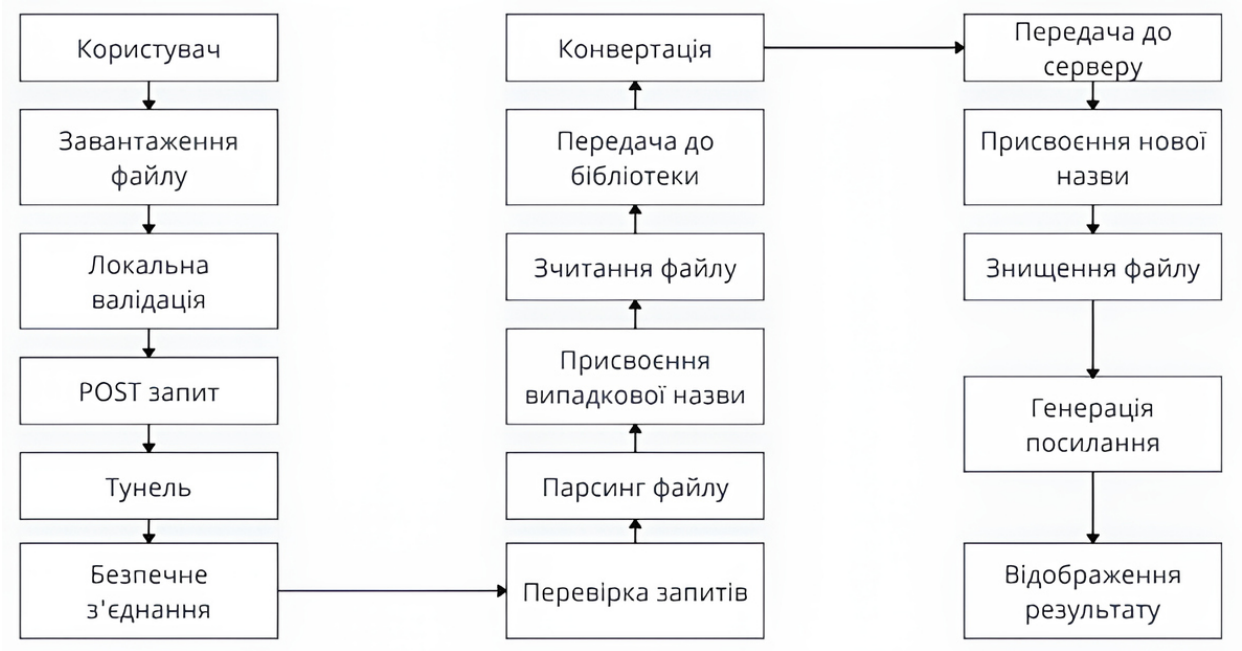

Рисунок 2.4 – Функціональна схема роботи конвертації

Джерело: розроблено автором

Першим чином користувач обирає файл або перетягує його у зону для завантаження файлу, після чого фронтенд перехоплює об’єкт файлу. Далі відбувається локальна валідація, де перевіряється розмір файлу, та якщо розмір перевищено, процес зупиняється. Здійснюється POST запит на частину backend, після чого запит іде через тунель Localtunnel. Запит спочатку проходить через Helmet, який дозволяє безпечне з’єднання, а express-rate-limit перевіряє, чи не робить відвідувач занадто багато запитів. Далі відбувається парсинг файлу, де

запит потрапляє до бібліотеки Multer, що зберігає оригінальний файл у тимчасову папку сервера. Під час збереження Multer присвоює файлу випадкове унікальне ім’я, щоб уникнути конфліктів, якщо двоє людей одночасно завантажать файл із однаковою назвою. Модуль File System бере завантажений файл з папки і зчитує його в оперативну пам’ять. Конвертація починається із виклику бібліотеки libreoffice, яка звертається до фізично встановленої програми, після чого експортує файл у обраний користувачем формат. Результат конвертації повертається назад до серверу. Сервер створює нове ім’я для готового файлу, а таймер знищує файли через 5 хвилин. Node.js генерує два безпечні посилання на основі поточного протоколу і домену тунелю, після чого віддає їх частині frontend у форматі JSON-відповіді. React отримує цю відповідь та змінює свій стан, показуючи кнопку для завантаження файлу.

2.3 Створення моделі бази даних

Проєктування бази даних важливе, оскільки від неї залежить організоване збереження, швидке отримання та кероване оновлення інформації. База даних забезпечує структуроване зберігання інформації. Дані розділяються на сутності та поля, що дозволяє уникнути хаотичного накопичення файлів і спрощує їх обробку, а також вона дає можливість швидкого пошуку та вибірки завантажених та оброблених файлів.

База даних забезпечує цілісність і надійність даних. Завдяки правилам і обмеженням можна уникнути дублювання інформації або втрати важливих записів.

Інфологічна модель відображає предметну область, у проєкті ключовим об’єктом предметної області є операції обробки файлів. Дані операції мають такі властивості:

– унікальний ідентифікатор операції;

– оригінальну назву завантаженого файлу;

– назва файлу після обробки системою;

– тип виконаної операції;

– початковий формат файлу;

– формат після застосування операції;

– час завантаження на сервер;

– час успішного завершення операції;

– час автоматичного видалення файлу з сервера.

Логічна модель будується на основі реляційного підходу, тобто способом організації даних у вигляді таблиць. Вся інформація зведена в єдину таблицю operations_history, її структура складається із:

– id (INT);

– original_name (TEXT);

– processed_name (TEXT);

– operation_type (TEXT);

– from_format (TEXT);

– to_format (TEXT);

– upload_time (DATETIME);

– process_end_time (DATETIME);

– delete_time (DATETIME), а також містить тип NULL до моменту фізичного видалення файлу.

Фізична структура бази даних розміщена в одному файлі database.sqlite у директорії на стороні backend, де дані всередині файлу організовані за допомогою структур даних B-дерев, що забезпечує швидкий пошук та вставку нових записів без необхідності виділяти під базу даних окремі томи або процеси в операційній системі.

Оскільки архітектура зберігання логів буде денормалізовано для максимальної швидкості запису, ERD-діаграма складатиметься з однієї ключової сутності, в яку входять автоінкремент, оригінальна назва файлу, змінена системою назва файлу, назва операції, вхідний та вихідний формати, час завантаження та завершення операції, а також час видалення файлу. Таку діаграму можна побачити на рисунку 2.5.

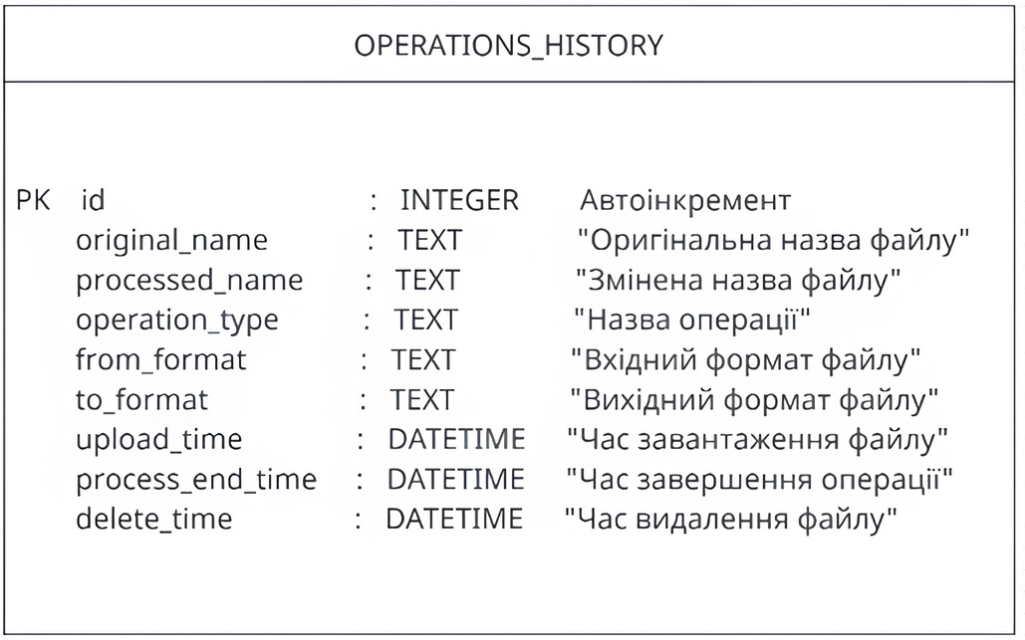

Рисунок 2.5 – ERD-діаграма

Джерело: розроблено автором

Сучасні системи керування базами даних найчастіше класифікують за типом підтримуваної моделі даних. У реляційних СКБД дані зберігаються у вигляді строго структурованих таблиць зі зв’язками між ними, прикладом таких баз даних є MySQL та SQLite.

Неструктуровані СКБД документо-орієнтовані та зберігають дані у JSON-подібних документах, наприклад, MongoDB. Також існують ключ-значення СКБД, що є швидкими сховищами в оперативній пам’яті. Графові СКБД існують для складних зв’язків. NewSQL СКБД поєднують масштабованість неструктурованих та транзакції реляційних баз даних.

Мотивацією вибору SQLite стало те, що в нього відсутні налаштування, що дозволяє обробляти файли «на льоту», оскільки налаштування важкого сервера було б надлишковим та ускладнило б розгортання проєкту.

Також свою роль зіграла локальність, оскільки SQLite є просто файлом, що забезпечує надшвидкий відгук, оскільки немає мережевих затримок між Node.js додатком і базою даних. Внаслідок даної локальності для резервної копії даних всієї історії операцій достатньо просто скопіювати один файл, що теж є мобільним випадком.

SQLite є реляційною, безсерверною СКБД. База даних підтримує принципи ACID, які гарантують надійну і правильну роботу з даними під час операцій, а саме:

– операція виконується повністю або не виконується взагалі;

– дані завжди залишаються коректними і відповідають правилам бази даних;

– одночасні операції не заважають одна одній та не «ламають» дані;

– після збереження дані не зникають навіть у випадку збою системи.

Динамічна типізація колонок, присутня в базі даних, передбачає, що структура даних не є жорстко фіксованою і може змінюватися або визначатися під час роботи програми, що актуально для колонки delete_time. Розмір бібліотеки складає менше 1 Мб, а також інтегрується безпосередньо в код програми.

Замість класичних драйверів типу ODBC, JDBC чи ADO.NET, у розроблюваному проєкті використовується нативна прив’язка за допомогою пакета sqlite3, що для Node.js є асинхронною обгорткою над оригінальним рушієм SQLite. Коли код робить запит на додавання логу, цей запит передається через прошарок C++ безпосередньо у двигун SQLite, та оскільки Node.js є однопотоковим, технологія доступу є суворо асинхронною. Сервер не зупиняє свою роботу в очікуванні запису даних на диск, натомість, SQL-запит ставиться в чергу циклу подій, і сервер одразу віддає сконвертований файл користувачеві, а запис у таблицю завершується у фоновому режимі через callback-функції.

РОЗДІЛ 3
РОЗРОБКА ВЕБСЕРВІСУ ДЛЯ КОНВЕРТАЦІЇ ФАЙЛІВ ТА ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ

3.1 Практична реалізація об’єкта проектування. Побудова блок-схем модулів програми

Розроблюваний вебсервіс використовує клієнтську та серверну частини для надання повного функціоналу користувачу. Взаємодія між клієнтською та серверною частинами реалізована за допомогою архітектурного стилю REST.

Серверна частина, розподілена на кілька файлів, але головним є server.js. Даний файл створює і запускає сервер на Node.js за допомогою Express, який обробляє API, адміністрування, віддачу створених файлів, а також періодично очищає старі файли, що показано у лістингу коду у додатку Б.

Він починає реалізацію з підключення модулів path для роботи з файловими шляхами, express для створення сервера, helmet для базового захисту HTTP-заголовків, cors для дозволу кросдоменних запитів, а також власних утиліт і маршрутів для API та адмін-панелі. Після цього створюється екземпляр застосунку Express і вмикається режим trust proxy, щоб коректно визначати IP клієнта. Далі підключається middleware helmet, але з обмеженням для ресурсів між доменами, щоб не блокувати завантаження ресурсів.

Після цього дозволяються CORS-запити і додаються парсери для JSON і URL-encoded даних, що дозволяє серверу приймати дані від клієнта у стандартних форматах. Маршрути організовані так, що все, що стосується адміністративної частини, обробляється через /api/admin, а основні API-запити через /api. Далі йде окрема логіка для шляху /converted, який використовується для доступу до конвертованих файлів, де сервер перевіряє, чи існує файл у директорії convertedDir, і якщо так, то повертає його користувачу. При цьому він формує правильний заголовок, щоб браузер завантажував його з коректною назвою, включаючи підтримку UTF-8 для імен файлів. Якщо файл не знайдено, запит передається далі. Далі налаштовується віддача фронтенду з папки dist,

тобто зібраного застосунку. Для всіх маршрутів, які не починаються з /api або /converted, сервер повертає index.html, що дозволяє працювати маршрутизації на стороні клієнта. Окремо запускається функція cleanupOldFiles, яка видаляє застарілі файли одразу при старті сервера, а також повторюється кожну хвилину через setInterval, щоб автоматично очищати сервер від тимчасових файлів. В кінці сервер запускається на порту 5000.

Файл api.js реалізовує всі маршрути, це інтерфейс взаємодії між клієнтом та сервером. Тут описується конфігурація, обробка завантаження файлів, конвертація, стиснення розміру файлів, обробка зображень, реалізація палітри по фото, розділення фото по частинах, а також завантаження файлів. У лістингу 3.1 показано початкову конфігурацію серверної частини.

Лістинг 3.1 – Початкова конфігурація серверної частини

```
const {
  libre, sharp, uploadsDir, convertedDir, rgbToHex
} = require("../utils");
const { logOperation, updateDeleteTime } = require("../db");

const IMAGE_FORMATS = ["jpg", "jpeg", "png", "webp", "gif", "avif",
"tiff", "bmp", "ico"];
const upload = multer({
  dest: uploadsDir,
  limits: { fileSize: 35 * 1024 * 1024 }
});
const apiLimiter = require("express-rate-limit")({
  windowMs: 15 * 60 * 1000,
  max: 100,
  message: { error: "Занадто багато запитів з цієї IP-адреси, спробуйте
пізніше." }
});
```

кінець лістингу 3.1

На початку з файлу utils імпортуються допоміжні інструменти та змінні, де libre використовується для конвертації документів через програму LibreOffice, sharp – для обробки та редагування зображень, uploadsDir і convertedDir містять шляхи до директорій для збереження завантажених і оброблених файлів, а функція rgbToHex призначена для перетворення кольорів із формату RGB у HEX. Далі з модуля db підключаються функції logOperation та updateDeleteTime,

які використовуються для запису інформації про виконані операції у базу даних та оновлення часу автоматичного видалення файлів після завершення обробки. Після цього створюється масив IMAGE_FORMATS, що використовується для визначення, чи є файл графічним зображенням, щоб застосувати до нього відповідну обробку. Наступним чином відбувається налаштування бібліотеки Multer для завантаження файлів на сервер. У параметрі dest вказується директорія uploadsDir, куди тимчасово зберігатимуться завантажені файли.

Також задається обмеження максимального розміру файлу – 35 мегабайтів, що дозволяє захистити сервер від надмірного використання ресурсів. Наприкінці налаштовується механізм обмеження кількості запитів за допомогою бібліотеки express-rate-limit. Змінна apiLimiter створює захист від великої кількості звернень до API з однієї IP-адреси. У даному випадку користувач може виконати не більше 100 запитів протягом 15 хвилин. Якщо ліміт перевищено, сервер повертає повідомлення про помилку із текстом про надто велику кількість запитів.

Лістинг, описаний у додатку В, реалізовує обробку завантаження файлів. Тут відбувається реалізація функції handleFileUpload, яка відповідає за завантаження файлів на сервер та перевірку можливих помилок під час цього процесу. На початку функція викликає метод upload.single("file"), який налаштований через бібліотеку Multer для прийому одного файлу з поля форми file. Після цього виконується перевірка наявності помилок під час завантаження. Якщо виникає помилка типу MulterError, код перевіряє її причину. У випадку перевищення максимального розміру файлу сервер повертає HTTP-відповідь зі статусом 413 та повідомленням про перевищення допустимого розміру 35 МБ. Далі перевіряються невідомі помилки, які не належать до Multer. У такому випадку сервер повертає статус 500, що означає внутрішню помилку сервера.

Якщо файл відсутній, сервер повертає помилку зі статусом 400 та повідомленням про те, що файл не завантажено. Ім’я файлу перетворюється у кодування utf8 для правильного відображення спеціальних символів у назвах файлів. Наприкінці у змінну req.uploadTime записується поточний час завантаження файлу.

Конвертація різних типів файлів описується у лістингу, наведеному у додатку Г. Тут реалізовується основний маршрут вебсервісу для конвертації файлів. Маршрут /convert приймає POST-запит, після чого з файлу витягуються його початкова назва, тимчасове ім’я, формат та шлях до файлу на сервері. Далі формується назва та шлях для нового конвертованого файлу. У блоці try сервер визначає розширення вихідного файлу та перевіряє, чи є початковий і цільовий формати графічними форматами, використовуючи масив IMAGE_FORMATS.

Після цього описано декілька сценаріїв конвертації, якщо вихідний та цільовий формати однакові, файл просто копіюється без змін, якщо виконується конвертація зображення в інше зображення, використовується бібліотека Sharp, якщо ж виконується конвертація зображення у PDF-документ, то воно зчитується у буфер пам’яті.

Якщо формат зображення не підтримується бібліотекою pdf-lib, воно попередньо перетворюється у PNG через бібліотеку Sharp. Потім створюється новий PDF-документ, до нього додається сторінка розміром із зображення, після чого саме зображення вставляється у PDF. У випадку, якщо обробляються документи різних форматів за допомогою LibreOffice. Файл зчитується у пам’ять, після чого викликається libre.convertAsync, яка виконує конвертацію документа у потрібний формат.

Після успішної обробки сервер записує інформацію про операцію у базу даних за допомогою функції logOperation. Далі налаштовується автоматичне видалення тимчасових і конвертованих файлів через 5 хвилин за допомогою setTimeout. Після видалення також оновлюється час видалення у базі даних через функцію updateDeleteTime. Сервер повертає інформацію про успішне завершення операції, посилання для завантаження готового файлу та посилання для його попереднього перегляду. У випадку виникнення помилки тимчасовий файл видаляється, а клієнту повертається повідомлення про помилку під час конвертації.

Лістинг, описаний у додатку Д, створює маршрут /compress-pdf, який відповідає за стиснення PDF-документів на сервері. Маршрут приймає POST-

запит і використовує middleware apiLimiter для обмеження кількості запитів, а також handleFileUpload для завантаження PDF-файлу на сервер. На початку з об’єкта req.file отримуються початкова назва файлу, його тимчасове ім’я та шлях до завантаженого файлу. Далі до тимчасового файлу додається розширення .pdf за допомогою fs.renameSync, оскільки деякі бібліотеки коректно працюють лише з файлами, які мають відповідне розширення. Після цього формується назва нового стисненого файлу та шлях до нього у директорії convertedDir. У блоці try файл зчитується у буфер пам’яті через fs.readFileSync. Далі бібліотека pdf-lib завантажує PDF-документ за допомогою PDFDocument.load. Потім створюється новий PDF-документ compressedDoc, у який копіюються всі сторінки з початкового документа через метод copyPages. Кожна сторінка додається до нового документа окремо. Після цього виконується збереження PDF із параметрами useObjectStreams: true та addDefaultPage: false.

Лістинг коду, який описано у додатку Е, реалізує декілька модулів для роботи із зображеннями: стиснення PNG-файлів, редагування фотографій та розділення зображень на частини. Перший маршрут /compress-png відповідає за стиснення PNG-зображень. Після отримання файлу сервер формує назву нового файлу та шлях до нього. Для стиснення використовується бібліотека Sharp, яка виконує повторне збереження PNG із параметрами quality: 80 та compressionLevel: 9.

Після завершення обробки інформація про операцію записується у базу даних через функцію logOperation. Другий маршрут /edit-photo реалізує функції редагування фотографій. Після завантаження файлу сервер отримує параметри редагування з тіла запиту у форматі JSON. Користувач може змінювати формат файлу, розміри зображення, виконувати поворот фото, дзеркальне відображення, змінювати яскравість і насиченість кольорів, застосовувати чорно-білий фільтр, змінювати контрастність, підвищувати різкість та обрізати зображення.

Для всіх операцій використовується бібліотека Sharp, яка формує послідовність трансформацій над зображенням. Після застосування всіх змін результат зберігається у новий файл. Третій маршрут /split-photo реалізує

функцію розділення зображення на декілька частин. Користувач може обрати тип поділу – горизонтальний або вертикальний, а також на яку кількість частин його розділити (2, 3, 4). Після отримання параметрів сервер визначає ширину та висоту зображення через метод metadata() бібліотеки Sharp.

Далі обчислюються координати областей обрізання для кожної частини. Якщо обрано поділ на чотири частини, зображення автоматично розділяється на чотири рівні області. В інших випадках виконується вертикальний або горизонтальний поділ на задану кількість фрагментів. Після розрахунку координат кожна частина вирізається за допомогою методу extract() бібліотеки Sharp та зберігається окремим файлом. Для кожного створеного фрагмента формується посилання для завантаження та попереднього перегляду. Інформація про операцію також записується у базу даних.

Модуль витягування домінантних кольорів із зображення описано у лістингу коду у додатку Ж. Тут описано маршрут /extract-palette, де на початку з об’єкта req.file отримуються дані про завантажений файл. Далі з параметра req.body.options зчитуються налаштування користувача, зокрема необхідність створення зображення з палітрою, формат результату та кількість кольорів для аналізу.

Якщо кількість кольорів не вказана, за замовчуванням використовується п’ять кольорів. У блоці try бібліотека Sharp виконує масштабування зображення до розміру count × 1 піксель. Це дозволяє отримати набір основних кольорів, які найбільш часто зустрічаються на зображенні. Після цього зображення перетворюється у raw buffer, де кожні три значення відповідають компонентам RGB окремого кольору. Код формує масив кольорів, у якому кожен колір зберігається у двох форматах: RGB та HEX. Для перетворення RGB у HEX використовується функція rgbToHex. У результаті створюється палітра кольорів, яка повернена користувачу у вигляді JSON-даних. Після цього перевіряється параметр generateImage.

Якщо користувач обрав створення графічного зображення палітри, сервер формує новий файл із візуалізацією кольорів. Створюється окреме полотно, на

якому для кожного кольору генерується прямокутна смуга відповідного кольору. За допомогою Sharp палітра приєднується до правої частини початкового зображення методом extend, після чого обидва зображення об’єднуються.

Маршрут /download/:file відповідає за завантаження оброблених файлів із сервера та описаний у лістингу 3.2.

Лістинг 3.2 – Завантаження оброблених файлів

```
router.get("/download/:file", (req, res) => {
  const file = req.params.file, filePath = path.join(convertedDir, file);
  let suggestedName = req.query.name || file;
  try { suggestedName = decodeURIComponent(suggestedName); } catch (e) {
}

  if (fs.existsSync(filePath)) {
    return res.download(filePath, suggestedName);
  } else {
    res.status(404).send("File not found or expired");
  }
});

module.exports = router;
```

кінець лістингу 3.2

З параметрів запиту отримується назва файлу, після чого за допомогою модуля path формується повний шлях до файлу у директорії convertedDir, де зберігаються всі конвертовані файли. Створюється змінна suggestedName, яка містить ім’я файлу, що буде показане користувачу під час завантаження. Якщо у параметрах запиту передано власну назву через req.query.name, використовується саме вона, інакше застосовується початкова назва файлу.

Після цього виконується спроба декодування назви файлу за допомогою decodeURIComponent. Якщо декодування викликає помилку, вона ігнорується, а сервер продовжує роботу. Далі код перевіряє, чи існує файл у файловій системі за допомогою fs.existsSync. Якщо файл знайдено, сервер викликає метод res.download, який надсилає файл користувачу як завантаження та автоматично встановлює потрібні HTTP-заголовки. При цьому використовується змінна suggestedName, щоб користувач отримав файл із коректною та зрозумілою

назвою. Якщо файл не існує або вже був автоматично видалений із сервера, клієнту повертається HTTP-відповідь зі статусом 404 та повідомленням «File not found or expired». Наприкінці виконується module.exports = router, що експортує створений маршрутизатор Express для подальшого підключення в основному серверному файлі застосунку.

Також варто зазначити файл utils.js, який відповідає за утилітний модуль серверної частини, даний принцип описано у лістингу коду, який знаходиться у додатку И. Спочатку тут підключаються основні модулі: fs і path для роботи з файловою системою, libreoffice-convert для конвертації документів, sharp для обробки зображень і база даних db. Після цього для бібліотеки Sharp вимикається кешування через sharp.cache(false), щоб зменшити використання пам’яті при великій кількості обробок зображень. Далі задається шлях до виконуваного файлу LibreOffice. Це потрібно, щоб сервер міг конвертувати документи через встановлений застосунок LibreOffice на пристрої. Бібліотека працює через Promise-варіант, щоб було зручніше використовувати async/await у коді сервера. Далі створюються дві робочі директорії: uploads (для тимчасово завантажених файлів) і converted (для результатів обробки). Якщо ці папки не існують, вони автоматично створюються.

Задається константа CLEANUP_AGE_MS, яка визначає час життя файлів у вигляді 5 хвилин. Функція rgbToHex перетворює кольори з формату RGB у HEX. Це використовується при створенні палітри. Функція cleanupOldFiles виконує автоматичне очищення серверу від старих файлів, де спочатку визначається поточний час, потім перевіряються обидві директорії uploads і converted, а наприкінці для кожного файлу обчислюється скільки часу він існує.

Якщо були видалені файли, виконується оновлення бази даних, де у таблиці operations_history оновлюється поле delete_time для відповідних записів. Наприкінці всі функції та змінні експортуються через module.exports, щоб їх можна було використовувати в інших частинах сервера. Це дозволяє централізовано керувати обробкою файлів, конвертацією, шляхами та очищенням усього проєкту.

Адмін-панель серверу містить сторінку для входу, а також сторінку із таблицею, в якій описано проведені операції. У лістингу коду, описаному у додатку К, показано логіку входу на адмін-панель. Тут задаються константи ADMIN_USER і ADMIN_PASS, які містять логін і пароль адміністратора. Функція isAuthenticated існує для захисту маршрутів. Вона перевіряє кеш запиту і шукає там значення admin_auth=loggedIn. Якщо такий кеш присутній, користувач вважається авторизованим і йому дозволяється доступ до наступного кроку. Якщо ні, то сервер повертає помилку «401 Unauthorized». Маршрут /check-auth дозволяє перевірити, чи авторизований користувач. Він читає кеш і повертає JSON-відповідь з полем authenticated: true або false. Маршрут /login відповідає за вхід адміністратора. Він перевіряє логін і пароль із тіла запиту. Якщо дані правильні, сервер встановлює кеш admin_auth=loggedIn з параметрами безпеки HttpOnly, обмеженням шляху та часом життя 1 година. Маршрут /logout видаляє кеш авторизації і повертає підтвердження успішного виходу.

Реалізація сторінки адмін-панелі наведена у додатку Л. Тут після отримання запиту сервер спочатку перевіряє наявність файлів у директоріях uploadsDir і convertedDir. Якщо папки існують, він зчитує їхній вміст за допомогою fs.readdirSync та отримує списки всіх завантажених і вже оброблених файлів. Якщо директорії відсутні, замість цього повертаються порожні масиви. Далі оголошується функція getLogs, яка виконує SQL-запит до таблиці operations_history, вибираючи останні 200 записів, відсортованих за спаданням ідентифікатора ID. Після виконання запиту результат зберігається у змінній logs. У stats передається кількість файлів у кожній директорії, а також самі списки завантажених і конвертованих файлів. logs містить історію операцій із бази даних. Якщо під час виконання будь-якої частини коду виникає помилка, вона перехоплюється у catch, і сервер повертає відповідь зі статусом 500 та текстом помилки.

Кожна сторінка сайту має однаковий дизайн, тому далі буде описано принцип розробки області для завантаження файлів у лістингу 3.3.

Лістинг 3.3 – Область для завантаження файлів

```
   <div className={`dropzone ${isDragActive ? 'active' : ''}`}
         onDragOver={handleDragOver}
         onDragLeave={handleDragLeave}
         onDrop={handleDrop}
         style={{marginTop: "20px"}}>
   <div className="dropzone-image-placeholder">
   <ArrowDown size={80} color="black" strokeWidth={3} /></div>
   <button className="select-file-btn" onClick={triggerSelect}> Вибрати
PNG файл </button>
   <p className="dropzone-text">або перетягніть PNG сюди</p>
   <input
         type="file"
         ref={fileInputRef}
         style={{ display: 'none' }}
         onChange={handleFileChange}
         accept=".png"/>
```

кінець лістингу 3.3

Основний контейнер div з класом dropzone створює зону для перетягування файлів. До нього динамічно додається клас active, якщо користувач перетягує файл над цією областю isDragActive. Це дозволяє візуально підсвічувати зону. Також тут підключені обробники подій onDragOver, onDragLeave і onDrop, які відповідають за логіку перетягування файлів у React. Всередині зони знаходиться іконка стрілки вниз ArrowDown, яка візуально підказує користувачу, що сюди можна завантажити файл. Далі розміщена кнопка «Вибрати PNG файл», яка викликає функцію triggerSelect, відкриває системне вікно вибору файлу, використовуючи прихований <input type="file">. Під кнопкою відображається текст-підказка «або перетягніть PNG сюди», який пояснює альтернативний спосіб завантаження через перетягування. Сам <input> прихований, він активується через кнопку або програмно через fileInputRef.

Подія onChange={handleFileChange} спрацьовує, коли користувач вибирає файл через діалог. Атрибут accept=".png" обмежує вибір лише PNG-файлами, цей параметр змінюється залежно від того, на якій сторінці користувач зараз знаходиться, щоб уникнути помилок при обробці.

У додатку М описано сторінку із показом обробленого файлу та відповідної кнопки для його завантаження. Це умовний рендеринг, який показує

різні екрани інтерфейсу залежно від стану процесу обробки файлу. Перша частина перевіряє стан state === "converting". Якщо файл зараз обробляється, користувач бачить екран завантаження. Він складається зі спінера, який візуально показує, що йде процес, і текстового повідомлення «Йде стиснення PNG... Зачекайте будь ласка.» Друга частина активується, коли обробка завершена і результат готовий.

Всередині блоку result-state є контейнер для попереднього перегляду зображення. У ньому показується зображення через тег <img>, яке завантажується з previewUrl. До нього застосовуються стилі, щоб воно коректно масштабувалося і не виходило за межі контейнера objectFit: "contain". Поруч знаходиться блок download-container, який містить основні дії для користувача.

Перша – це кнопка «Завантажити файл», яка веде за адресою downloadUrl і дозволяє завантажити оброблений файл. Нижче відображається попереджувальний текст, який повідомляє про автоматичне видалення завантаженого вмісту. Також є кнопка «Стиснути інший файл», яка викликає функцію resetPage і повертає користувача назад.

Базові глобальні стилі сайту описані у додатку Н. Спочатку через :root оголошуються CSS-змінні, які використовуються як централізована система кольорів у всьому проєкті. Тут визначаються основні кольори інтерфейсу: темно-сірий фон --gray-bg, основний акцентний зелений колір теми --theme-red, білий і чорний текст, а також світлий фон. Далі йде універсальний селектор *, який скидає стандартні відступи браузера (margin: 0, padding: 0) і встановлює box-sizing: border-box. Селектор body задає основний шрифт сторінки Segoe UI, встановлює білий фон сторінки, чорний колір тексту і базову висоту рядка line-height: 1.6 для покращеної читабельності. Клас .app-container використовується як головний контейнер застосунку. Він зроблений flex-контейнером із вертикальним напрямком і мінімальною висотою на весь екран, щоб сторінка завжди займала всю висоту вікна браузера. Клас .main-content визначає основну частину сторінки. Він також використовує flex-структуру, розтягується на доступний простір і має вертикальні відступи, щоб контент не припадав до країв.

Для зберігання історії операцій та моніторингу активності системи була обрана система керування базами даних SQLite. Блок-схему методики реалізації бази даних показано на рисунку 3.1.

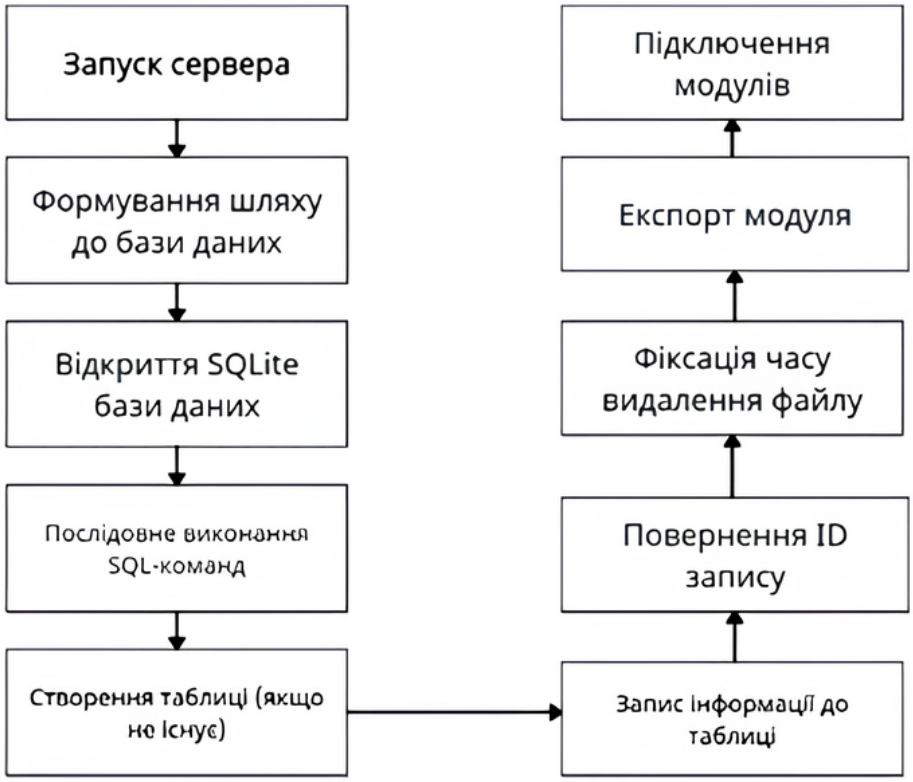

Рисунок 3.1 – Методика реалізації бази даних

Джерело: розроблено автором

Після запуску сервера, підключаються модулі path і sqlite3, де через path формується шлях до файлу бази даних database.sqlite. Далі створюється об’єкт бази даних db, який відкриває або створює SQLite-файл. Після цього використовується db.serialize(), щоб гарантувати послідовне виконання SQL-команд. Усередині створюється таблиця operations_history, якщо вона ще не існує.

Таблиця зберігає всю історію операцій з файлами. Вона містить такі поля: унікальний ідентифікатор id, назву оригінального файлу, назву обробленого файлу, тип операції, початковий і кінцевий формат, час завантаження, час завершення обробки та час видалення файлу. Далі оголошується функція logOperation. Вона використовується для запису нової операції в базу даних.

Функція приймає всі ключові параметри операції з файлом і повертає Promise, оскільки робота з базою є асинхронною. Всередині виконується SQL-запит INSERT INTO operations_history, який додає новий запис у таблицю. Після успішного виконання повертається this.lastID, що є ID щойно створеного запису.

Якщо виникає помилка, вона передається через reject. Наступна функція updateDeleteTime оновлює час видалення файлу. Якщо id відсутній, функція одразу завершується. Інакше виконується SQL-запит UPDATE, який записує час видалення у поле delete_time для відповідного запису. Наприкінці модуль експортує об’єкт бази даних db, функцію logOperation для створення записів і функцію updateDeleteTime для оновлення часу видалення.

Основним елементом бази даних є таблиця operations_history, яка містить інформацію про кожну проведену операцію з файлами. В цій структурі знаходиться id: INTEGER унікальний ідентифікатор запису; original_name: TEXT початкова назва файлу, завантаженого користувачем; converted_name: TEXT назва обробленого файлу на сервері; operation_type: TEXT тип операції (convert, compress, edit, split, palette); source_format: TEXT вхідний формат файлу; target_format: TEXT вихідний формат; upload_time: DATETIME час завантаження файлу; process_end_time: DATETIME час завершення обробки; deleted_at: DATETIME мітка часу, коли файл був видалений із сервера.

Основними SQL-запитами є запит на створення таблиці, запит на додавання нового запису, що виконується після кожної успішної операції для фіксації логів, запит на оновлення статусу видалення, що виконується автоматично через 5 хвилин після обробки, коли спрацьовує механізм очищення папок, а також запит на отримання історії для адмін-панелі.

3.2 Тестування та налагодження вебсистеми

Тестування вебсайту – це необхідна частина розробки, оскільки воно дозволяє перевірити коректність роботи всіх функцій системи до того, як її побачать користувачі. Невелика помилка в коді може призвести до некоректної

обробки даних, втрати файлів, збоїв у роботі сервера або повної недоступності сервісу. Тому тестування допомагає виявити такі проблеми ще на етапі розробки, коли їх виправлення не матиме великих зусиль.

Автоматизовані тести, які інтегруються безпосередньо в коді, дозволяють перевіряти окремі функції або модулі системи без необхідності ручного тестування кожного разу, що полегшує роботу для проєктів із зростаючим функціоналом.

Для тестування розроблюваного вебсервісу було розроблено декілька важливих автоматизованих тестів. У лістингу 3.4 описано тест на перевірку функції перетворення.

Лістинг 3.4 – Тест на перевірку функції перетворення

```
console.log("Модульне тестування функцій.");
  try {
    const hex = rgbToHex(255, 0, 0);
    assert.strictEqual(hex, "#ff0000", "Помилка: rgbToHex(255,0,0) має
повертати #ff0000");
    const hexBlack = rgbToHex(0, 0, 0);
    assert.strictEqual(hexBlack, "#000000", "Помилка: rgbToHex(0,0,0) має
повертати #000000");
    console.log("Unit-тести пройдено успішно!");
  } catch (err) {
    console.error("Unit-тест провалено:", err.message);
  }
```

кінець лістингу 3.4

Цей код виконує просте модульне тестування функції rgbToHex, яка відповідає за перетворення кольорів із формату RGB у HEX. Це необхідна частина для роботи із інструментами, які працюють із зображеннями, а також для створення палітри. В блоці try відбувається перевірка роботи функції на двох контрольних прикладах. У першому випадку функція викликається з параметрами 255, 0, 0, що відповідає червоному кольору, і перевіряється, чи результат дорівнює рядку #ff0000. У другому випадку перевіряється перетворення чорного кольору 0, 0, 0 у #000000. Для перевірки використовується assert.strictEqual, який порівнює фактичний результат із очікуваним і викликає помилку, якщо вони не збігаються.

У лістингу 3.5 описано тест на перевірку директорій.

Лістинг 3.5 – Тест на перевірку директорій

```
console.log("\nПеревірка файлової системи");
  try {
    assert.strictEqual(fs.existsSync(uploadsDir), true, `Папка
${uploadsDir} відсутня!`);
    assert.strictEqual(fs.existsSync(convertedDir), true, `Папка
${convertedDir} відсутня!`);
    console.log("Директорії для файлів існують.");
  } catch (err) {
    console.error("Тест оточення провалено:", err.message);
  }
```

кінець лістингу 3.5

Тут виконується перевірка файлової системи під час роботи сервера, щоб переконатися, що всі необхідні директорії для збереження файлів існують. Перевіряється наявність двох папок: uploadsDir, де зберігаються завантажені користувачами файли, та convertedDir, де зберігаються вже оброблені або конвертовані файли. Тестування роботи бази даних описано у лістингу 3.6.

Лістинг 3.6 – Перевірка роботи бази даних

```
console.log("\nТестування бази даних SQLite");
  try {
    const testName = "test_file.pdf";
    const opId = await logOperation(testName, "test_out.pdf", "test",
"pdf", "pdf", new Date().toISOString(), new Date().toISOString());
    assert.ok(opId, "ID операції має бути створено");
    db.get("SELECT original_name FROM operations_history WHERE id = ?",
[opId], (err, row) => {
      if (err) throw err;
      assert.strictEqual(row.original_name, testName, "Дані в БД не
збігаються з надісланими!");
      console.log("Логування в базу даних працює коректно.");
    });
  } catch (err) {
    console.error("Тест БД провалено:", err.message);
  }
```

кінець лістингу 3.6

В блоці try створюється тестовий запис, де викликається функція logOperation із фіктивними даними про файл test_file.pdf. Ця функція повинна

додати запис у базу даних і повернути id щойно створеної операції. Після цього через assert.ok перевіряється, чи цей ID дійсно був отриманий, що підтверджує успішне створення запису. Далі виконується SQL-запит SELECT, який дістає з бази даних поле original_name для щойно створеного запису. У callback-функції перевіряється, чи значення в базі row.original_name співпадає з тестовим значенням testName. Якщо дані не збігаються, викликається помилка через assert.strictEqual.

Фінальний тест на перевірку коректності роботи логіки обробки назв файлів і їхніх розширень описано у лістингу 3.7.

Лістинг 3.7 – Перевірка коректності роботи обробки назв

```
console.log("\nПеревірка логіки обробки назв та розширень");
  try {
    const filename = "MyReport.DOCX";
    const ext = path.extname(filename).toLowerCase().replace('.',
'').trim();
    assert.strictEqual(ext, "docx", "Логіка визначення розширення не
працює для верхнього регістру!");

    const cyrillicName = "Документ.pdf";
    const bufferFix = Buffer.from(cyrillicName, 'utf8').toString('utf8');
    assert.strictEqual(bufferFix, "Документ.pdf", "Помилка обробки
кирилиці!");
    console.log("Логіка обробки назв працює вірно.");
  } catch (err) {
    console.error("Тест логіки провалено:", err.message);
  }
```

кінець лістингу 3.7

Тут тестується визначення розширення файлу для імені «MyReport.DOCX» за допомогою path.extname, приведення до нижнього регістру та очищення символу крапки перевіряється, чи правильно отримується значення «docx». Після цього перевіряється обробка назв файлів із кирилицею. Створюється рядок «Документ.pdf», який проходить через перетворення Buffer.from(...).toString('utf8'), що імітує виправлення можливих проблем із кодуванням. Потім перевіряється, чи результат залишається незмінним і коректно зберігає кириличні символи.

У результаті тести пройшли успішно, та результати були виведені у консоль, що можна побачити на рисунку 3.2.

```
PS D:\University\Курсач\Project\server> node tests.js
Запуск тестування системи

Модульне тестування функцій.
Unit-тести пройдено успішно!

Перевірка файлової системи
Директорії для файлів існують.

Тестування бази даних SQLite

Перевірка логіки обробки назв та розширень
Логіка обробки назв працює вірно.
Логування в базу даних працює коректно.

Тестування успішне
```

Рисунок 3.2 – Результати автоматизованих тестів

Джерело: розроблено автором

Оскільки вебсервіс не є дуже вибагливим до системних параметрів користувача, то мінімальні вимоги включатимуть в себе:

– процесор 1.6 ГГц (2 ядра);

– оперативна пам’ять 1 ГБ;

– вільне місце на диску 500 МБ, що залежить від потреб користувача;

– браузер Microsoft Edge.

А також, рекомендовані вимоги, що впливатимуть на комфорт користувача:

– процесор 2.4 ГГц (4 ядра);

– оперативна пам’ять 4 ГБ;

– вільне місце на диску 1 ГБ;

– остання версія браузеру Chrome.

3.3 Інструкція користувача з інсталяції та використання сервісу

Для користування сайтом користувачу необхідні визначені мінімальні вимоги до пристрою, мережу Інтернет, а також наявність браузеру. Інструкція

користувача потрібна для того, щоб пояснити як правильно використовувати вебсервіс. Така документація демонструє завершеність розробки та підтверджує, що програмний продукт може використовуватися іншими людьми без участі автора.

Щоб перейти до вебсервісу, необхідно перейти за посиланням, яке згенерує тунель localtunnel. Після того, як користувач потрапить до сайту, він побачить головну сторінку, що зображена на рисунку 3.3, через яку можна перейти до будь-яких інструментів сайту.

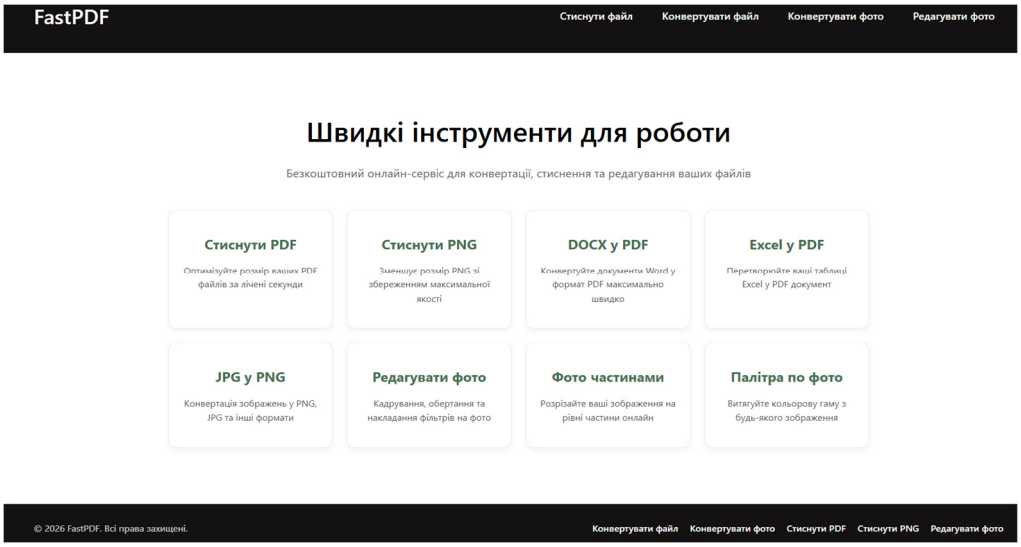

Рисунок 3.3 – Головна сторінка вебсервісу

Джерело: розроблено автором

Наступним чином користувач натискає або на меню сайту зверху, або на картки-посилання в центрі, або на меню в підвалі сайту. Оскільки інструменти стиснення та конвертування різних типів файлів працюють за однаковим принципом, буде розглянуто лише один випадок цих функцій. Дизайн головної сторінки виконаний у сучасному розумінні стилів із використанням зрозумілої навігації, контрастних елементів керування та гармонійної кольорової гами. Для покращення взаємодії з користувачем застосовано інтуїтивно зрозумілі кнопки, інформаційні блоки та візуальні підказки.

Користувач натискає на картку «DOCX у PDF» та бачить інтерфейс, що зображено на рисунку 3.4.

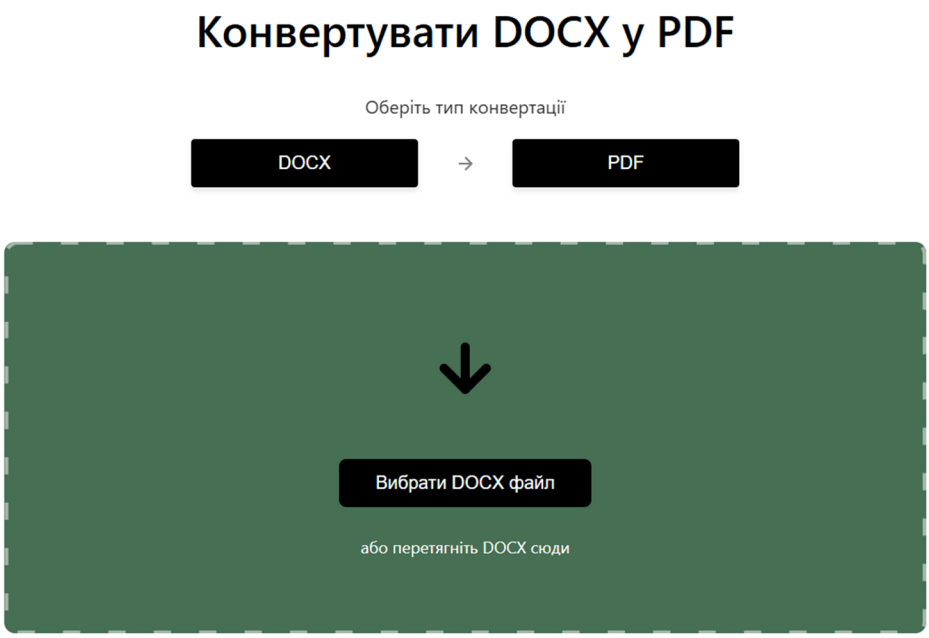

Рисунок 3.4 – Інтерфейс конвертування файлу

Джерело: розроблено автором

Він спочатку натискає на першу кнопку, яка на даний момент знаходиться під назвою «DOCX», обираючи тип файлу JPG із випадного списку, а потім за таким самим принципом кнопку «PDF», обравши тип файлу PDF. Інтерфейс пояснює, що перша кнопка відповідає за тип файлу, який необхідно перетворити, а друга навпроти – тип в який вона перетвориться. Дані кнопки супроводжуються пояснювальним текстом над ними.

Користувач або перетягує свій файл у зелену велику зону, або натискає кнопку «Вибрати DOCX файл». Після вибору сайт автоматично перенаправляє людину на наступну сторінку очікування, яка пояснює, що процес конвертації відбувається.

Така сторінка містить графічний блок із «спінером» завантаження, а також текст «Йде конвертація у PDF. Зачекайте будь ласка..», що повідомляє користувача про успішне завантаження та прохання зачекати.

Після успішної конвертації користувач бачить сторінку для перегляду та завантаження конвертованого файлу, що зображено на рисунку 3.5, разом із кнопками для завантаження, початку нової обробки файлу, та відображення конвертованого результату.

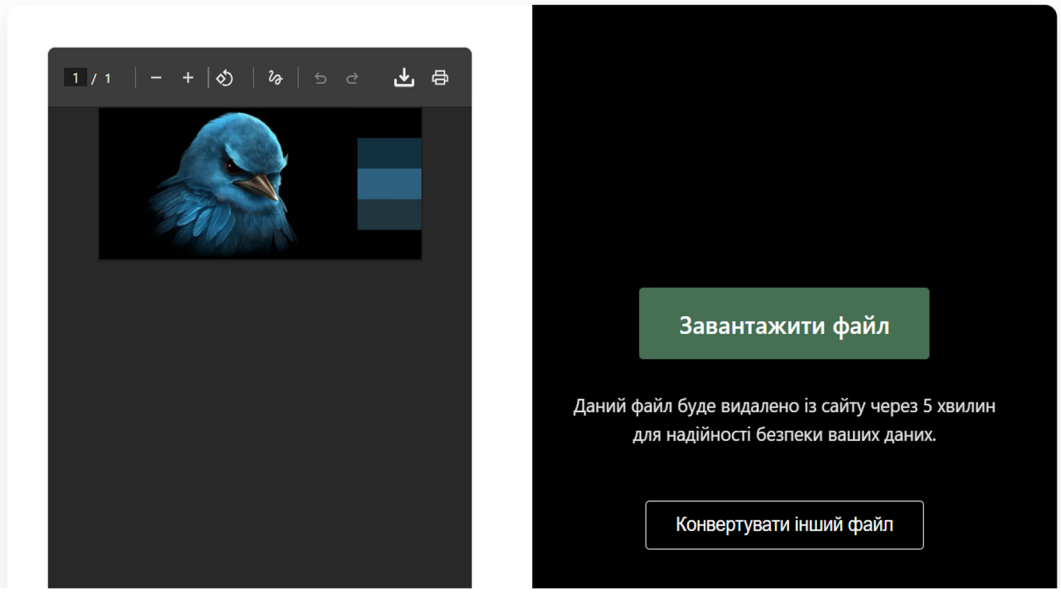

Рисунок 3.5 – Сторінка перегляду та завантаження файлу

Джерело: розроблено автором

Після оцінки конвертованого файлу за допомогою інтерфейсу з лівої сторони, користувач натискає кнопку «Завантажити файл», після чого файл автоматично завантажується на пристрій користувача. Після натискання кнопки «Конвертувати інший файл» сторінка очищується, доступ до файлу припинено, а процес конвертування можна реалізувати повторно.

Для початку процесу редагування фото обирається розділ «Редагувати фото» у меню сайту, після чого у випадному списку користувач натискає на кнопку «Редагувати фото», або він користується відповідною карткою на головній сторінці сайту. Після чого повторюється вищеописаний процес завантаження фото.

Наступним чином користувачу відображається інтерфейс редагування. Він бачить інтерфейс, який містить розділи «Змінити розмір», «Поворот та Віддзеркалення», «Обрізання», «Колір», «Якість», в кожному з яких існують відповідні до цих розділів параметри для обробки.

Розділи мають активний статус, тому вони розгортаються при натисканні на них, або на знак «+» у правому кінці об’єкта, даний інтерфейс зображено на рисунку 3.6.

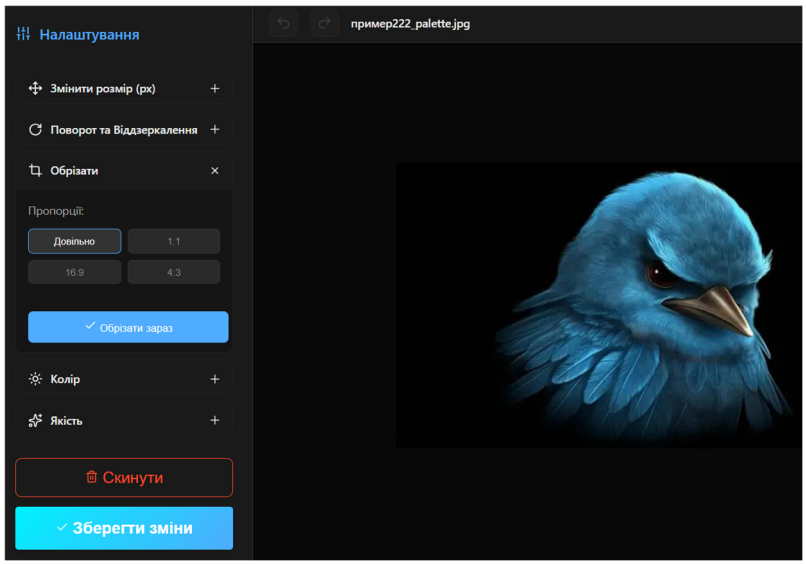

Рисунок 3.6 – Інтерфейс редагування фото

Джерело: розроблено автором

Розділ «Змінити розмір» дозволяє вписати необхідну ширину та висоту зображення у пікселях, за допомогою клавіатури.

Розділ «Поворот та Віддзеркалення» містить 4 кнопки «-90», «+90», «Горизонтально», «Вертикально», що означає відповідно поворот проти 90 градусів, поворот за 90 градусами, віддзеркалення горизонтально та вертикально.

Розділ «Обрізати» містить 4 кнопки, кнопка «Довільно» дозволяє обрізати фото за сіткою по бажанню користувача, кнопка «1:1» робить сітку прив'язаною до фігури квадрату, та подальша обрізка не виходить за норми цієї форми, такі ж самі обмеження відносяться і до кнопок «16:9» та «4:3», а кнопка «Обрізати зараз» підтверджує область обрізки та застосовує зміни.

Розділ «Колір» містить повзунки «Яскравіть», «Насиченість» та радіо-кнопку «Чорно-білий фільтр». Повзунки за стандартом знаходяться на рівні 100 %, що дозволяє знизити параметр до 0 %, або підвищити до 200 %.

Розділ «Якість» містить повзунки «Чіткість» та «Контраст», перший параметр починається від значення 0 %, а другий – від 100 %. Червона кнопка «Скинути» скидує всі активні зміни до тих, що були на початку роботи. Дві стрілочки біля назви файлу дозволяють відмінити зміни, або навпаки –

повернути їх. Кнопка «Зберегти зміни» зберігає зміни та переносить користувача на сторінку для завантаження фото, яка аналогічно працює із сторінкою для завантаження, що розглядалася на рисунку 3.5.

Наступним чином користувач переходить звичним способом до сторінки «Фото частинами», що зображено на рисунку 3.7.

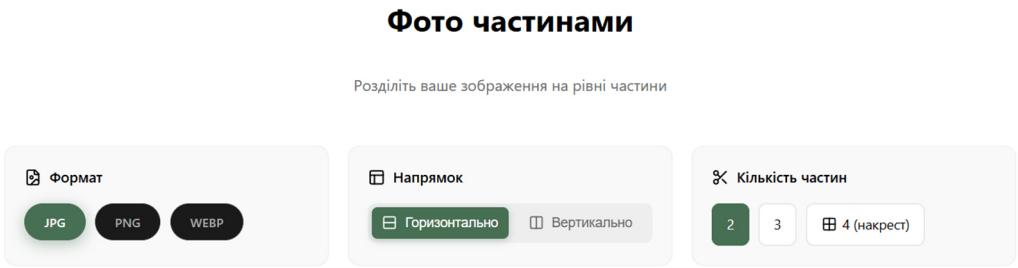

Рисунок 3.7 – Інтерфейс сторінки «Фото частинами»

Джерело: розроблено автором

Тут користувач бачить, окрім звичної зони для завантаження файлу, нові кнопки, де розділ «Формат» відповідає за вибір формату завантаженого фото; «Напрямок» за напрямок, в якому будуть обрізатися частини; «Кількість частин» для вибору кількості частин, на які буде розбите завантажене фото. Після успішного завантаження фото та вибору параметрів користувач проходить звичний процес та завантажує оброблені частини окремо.

Сторінка «Палітра по фото» зображена на рисунку 3.8.

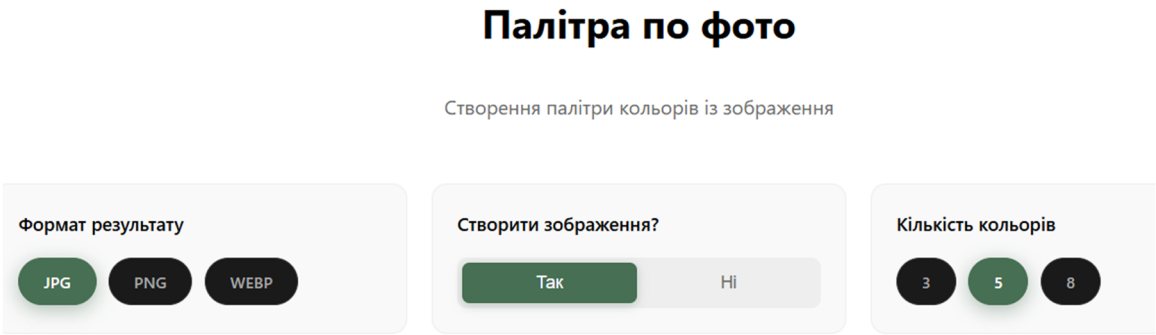

Рисунок 3.8 – Інтерфейс сторінки «Палітра по фото»

Джерело: розроблено автором

Вона має схожий інтерфейс зі сторінкою «Фото частинами», але містить два нових блоки. Блок «Створити зображення?» дозволяє визначити, чи потрібно домальовувати до завантаженого фото палітру кольорів, яка була витягнута у процесі. Блок «Кількість кольорів» визначає кількість кольорів, які будуть витягнуті із фото. При звичному завантаженні та обробці користувачу відобразиться нова сторінка для завантаження, що показана на рисунку 3.9.

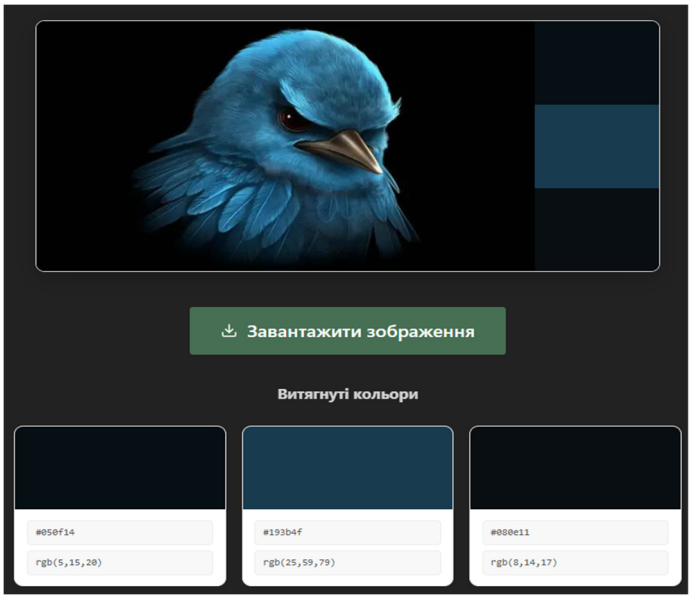

Рисунок 3.9 – Інтерфейс завантаження палітри по фото

Джерело: розроблено автором

Тут користувач має змогу завантажити зображення за кнопкою «Завантажити зображення», а також має можливість побачити картки із зображеним кольором у такій кількості, якій було обрано на початку завантаження.

Картки містять колір, HEX та RGB коди, які можна скопіювати у буфер обміну, шляхом натиснення на необхідний варіант. Також сторінка супроводжується текстом, який повідомляє, що згенероване фото буде видалено через 5 хвилин після обробки.

РОЗДІЛ 4
СПЕЦІАЛЬНІ РОЗРАХУНКИ

Одним із критичних аспектів безпеки розробленої вебсистеми є захист від атак типу DoS та DDoS та зловмисного автоматизованого використання ресурсів сервера. Для вирішення цієї задачі в системі реалізовано механізм обмеження інтенсивності запитів Rate Limiting за допомогою бібліотеки express-rate-limit. В основі алгоритму фіксованого вікна лежить поділ часової шкали на фіксовані інтервали або «вікна» тривалістю Δt. Для кожної унікальної IP-адреси користувача система підтримує лічильник запитів у межах поточного вікна. При надходженні запиту система ідентифікує клієнта за його IP-адресою. Перевіряється наявність лічильника для даного вікна. Якщо вікно нове, то лічильник ініціалізується нулем. Якщо значення лічильника менше встановленого порогу N, то запит передається на подальшу обробку, а лічильник інкрементується. Якщо поріг досягнуто, сервер миттєво повертає статус «Too Many Requests», не витрачаючи ресурси на обробку конвертування або обробки файлів. Умова допуску запиту описана у формулі 4.1.

Відомо, що

$$\begin{aligned} \mathrm{A}(r) = 1, \text{якщо } \sum(r_i) < N_{max}, \\ \mathrm{A}(r) = 0, \text{якщо } \sum(r_i) \geq N_{max}, \end{aligned} \tag{4.1}$$

де, A(r) – функція рішення (1 – дозволити, а 0 – блокувати);

$\sum(r_i)$ – сума запитів від однієї IP-адреси за поточний період;

N_{max} – максимальний ліміт запитів у кількості 100;

часове вікно (Δt) = 900 секунд.

При перевищенні ліміту користувач отримує JSON-повідомлення про помилку «Занадто багато запитів з цієї IP-адреси, спробуйте пізніше». Така конфігурація забезпечує баланс між зручністю для звичайних користувачів та надійним захистом сервера від спроб перевантаження під час конвертації файлів.

Наступним аспектом безпеки розроблюваного вебсервісу є контроль обсягу вхідних даних. Без певних обмежень зловмисник може здійснити атаку з вичерпання ресурсів, завантажуючи файли надвеликого розміру, що призведе до переповнення дискового простору сервера або перевантаження оперативної пам’яті. Для контролю вхідного трафіку в системі використано конфігурацію об’єкта limits у проміжній бібліотеці Multer. Даний механізм перевіряє розмір файлу в процесі його передачі, не чекаючи повного завантаження в пам’ять, що дозволяє миттєво розірвати з’єднання у разі порушення лімітів. Формула 4.2 описує захист від переповнення пам’яті та дискового простору.

Відомо, що

$$S_{file} \leq S_{max}, \tag{4.2}$$

де, S_{file} – це розмір завантаженого файлу;

S_{max} – максимально прийнятний розмір файлу у кількості 35 Мегабайт.

При ініціалізації завантаження сервер аналізує заголовок Content-Length, якщо значення заголовка або фактичний потік байтів перевищує задане значення, то генератор обробки переривається. Сервер повертає виключення LIMIT_FILE_SIZE, яке обробляється глобальним перехоплювачем помилок для виведення коректного повідомлення користувачеві. Даний підхід дозволяє гарантувати стабільність роботи сервера навіть за умов інтенсивного використання, запобігаючи аварійному завершенню процесів через помилку «Out of Memory» та захищаючи сервер від переповнення накопичувача.

Для забезпечення безпеки конфіденційних даних, таких як історія операцій користувачів та системні логи, у проекті реалізовано розмежування рівнів доступу. Основний механізм захисту адміністративної панелі базується на процесі авторизації. Доступ до ресурсу /admin обмежено програмним фільтром, який аналізує надіслані облікові дані перед наданням доступу до об’єктів бази даних SQLite. Процес перевірки прав доступу до адміністративної панелі описується у логічній формулі 4.3.

Відомо, що

$$\text{Result} = (\text{Input\_Login} \text{ === "admin"}) \text{ AND } (\text{Input\_Password} \text{ === "pdfMaster2026!"}), \quad (4.3)$$

де, Input_Login – це поле для логіну;

Input_Password – поле для паролю;

(===) – це знак «суворого порівняння»;

AND – це логічне «І», тому обидві умови мають бути виконані одночасно;

Result – це підсумок.

Вся перевірка відбувається виключно на стороні сервера, що виключає можливість підміни статусу авторизації через маніпуляції з кодом у браузері клієнта. Завдяки поєднанню з механізмом Rate Limiting, принцип якого описано у формулі 4.1, система захищена від автоматизованого підбору паролів, оскільки кількість спроб входу обмежена часовим вікном. Клієнтська частина не отримує жодної інформації про структуру бази даних або логів, поки не буде пройдено етап авторизації з отриманням успішного статусу.

ВИСНОВКИ

Внаслідок виконання кваліфікаційної роботи бакалавра було проведено аналіз сучасних проблем в області обробки цифрових файлів та веборієнтованих сервісів. Було досліджено основні недоліки існуючих рішень, серед яких складність у використанні, перевантаженість функціоналу, відсутність гармонійного користувацького досвіду та обмеження доступності для звичайних користувачів. На основі проведеного аналізу було визначено основні напрями вдосконалення сервісів цифрової обробки файлів, що відповідають сучасним світовим тенденціям розвитку вебресурсів. Внаслідок отримання результатів було сформовано основу для створення доступного та ефективного вебсервісу для конвертації файлів та цифрової обробки зображень.

У процесі виконання роботи були визначені та опрацьовані функціональні й нефункціональні вимоги до вебсервісу. Особливу увагу приділено забезпеченню простоти використання, швидкодії, доступності та комфортної взаємодії користувача із системою. Сформовані вимоги дозволили реалізувати сервіс, орієнтований на сучасних користувачів, які потребують позитивного користувацького досвіду без використання складних програмних продуктів.

Розробка архітектури вебсервісу забезпечила гармонійну взаємодію клієнтської та серверної частини системи. Для реалізації серверної частини використано платформу Node.js та фреймворк Express, а клієнтська частина реалізована із застосуванням React. Обрана архітектура дозволила забезпечити стабільність роботи сервісу, обмін даними та можливість подальшого масштабування системи. Результати розробки узгоджуються зі світовими підходами до побудови вебзастосунків та мають змогу бути використаними у подальших розробках.

Для реалізації функцій конвертації та обробки файлів спочатку було обґрунтовано методи їх опрацювання, з подальшим дослідженням сучасних програмних бібліотек та інструментів, зокрема засобів для роботи із цифровими зображеннями та документами. Реалізовані методи забезпечують коректне

виконання операцій конвертації різних типів файлів, їх зменшення об’єму, а також редагування зображень, що включає застосування фільтрів, створення палітри, поворот, віддзеркалення, обрізку та налаштування інших функцій до фото.

Створення інтуїтивного та легкого користувацького інтерфейсу вебсервісу з урахуванням принципів сучасного UI/UX-дизайну забезпечило просту навігацію та комфортне використання функціоналу сервісу навіть для користувачів без спеціальної технічної підготовки. Соціальна значущість роботи полягає у спрощенні виконання повсякденних завдань користувачів, пов’язаних із цифровими файлами та зображеннями, а також у підвищенні доступності сучасних інструментів цифрової обробки через вебсередовище.

На завершальному етапі роботи було проведено тестування та оцінено результати його функціонування. Перевірка працездатності підтвердила коректність реалізації основних функцій системи, стабільність роботи та відповідність визначеним вимогам.

Розроблений вебсервіс може бути використаний у навчальній та повсякденній діяльності користувачів. Отримані результати можуть бути застосовані у сфері веброзробки та цифрової обробки даних. Подальший розвиток роботи може передбачати розширення підтримуваних форматів файлів, впровадження додаткових інструментів редагування, оптимізацію продуктивності сервісу та інтеграцію нових технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Delivering Document Conversion as a Cloud Service with High Throughput and Responsiveness. 2022. URL: https://arxiv.org/abs/2206.00785 (дата звернення: 10.02.2026).

2. Data Transformation Techniques in ETL. International Journal of Multidisciplinary on Science and Management. 2024. URL: https://www.ijmsm.org/ijmsm-v1i2p101.html (дата звернення: 11.02.2026).

3. Overview of Research Progress of Digital Image Processing Technology. 2022. URL: https://doi.org/10.1088/1742-6596/2386/1/012034 (дата звернення: 11.02.2026).

4. Accessibility Issues in Ad-Driven Web Applications. 2024. URL: https://arxiv.org/abs/2409.18590 (дата звернення: 11.02.2026).

5. Secure Data Storage and Sharing Techniques for Data Protection in Cloud Environments: A Systematic Review. 2022. URL: https://clouds.cis.unimelb.edu.au/papers/DataProtectionClouds2022.pdf (дата звернення: 12.02.2026).

6. Security and Privacy Concerns in Cloud-based Scientific and Business Workflows: A Systematic Review. 2022. URL: https://arxiv.org/abs/2210.02161 (дата звернення: 12.02.2026).

7. Rethinking Image Compression on the Web with Generative AI. 2024. URL: https://arxiv.org/abs/2407.04542 (дата звернення: 12.02.2026).

8. Zamzar. URL: https://www.zamzar.com (дата звернення: 13.02.2026).

9. CloudConvert. URL: https://cloudconvert.com (дата звернення: 13.02.2026).

10. iLoveIMG. URL: https://www.iloveimg.com/uk (дата звернення: 13.02.2026).

11. React Documentation. URL: https://react.dev/reference/react (дата звернення: 15.02.2026).

12. ECMAScript 2025 Language Specification. 2025. URL: https://262.ecma-international.org (дата звернення: 15.02.2026).

13. Node.js v26.1.0 documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 15.02.2026).

14. Express API Reference. URL: https://expressjs.com/en/api.html (дата звернення: 16.02.2026).

15. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 16.02.2026).

ДОДАТКИ

Додаток А

Сертифікат про участь у міжнародній науково-практичній конференції

Додаток Б

Файл налаштування серверної частини

```
const path = require("path");
const express = require("express");
const helmet = require("helmet");
const cors = require("cors");
const { cleanupOldFiles, convertedDir } = require("./utils");
const apiRoutes = require("./routes/api");
const adminRoutes = require("./routes/admin");
const app = express();
app.set('trust proxy', 1);

app.use(helmet({
  contentSecurityPolicy: false,
  crossOriginEmbedderPolicy: false,
  crossOriginResourcePolicy: false,
  frameguard: false
}));
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use("/api/admin", adminRoutes);
app.use("/api", apiRoutes);
app.use("/converted", (req, res, next) => {
  const fs = require("fs");
  const file = req.path.split('/').pop();
  const filePath = path.join(convertedDir, file);
  if (fs.existsSync(filePath)) {
    const suggestedName = req.query.name || file;
    const encodedName = encodeURIComponent(suggestedName);
    res.setHeader('Content-Disposition', `inline;
filename="${encodedName}"; filename*=UTF-8''${encodedName}`);
    return res.sendFile(filePath);
  }
  next();
});
app.use("/converted", express.static(convertedDir));
app.use(express.static(path.join(__dirname, '../client/website/dist')));
app.get(/.*/, (req, res, next) => {
  if (req.originalUrl.startsWith('/api') ||
req.originalUrl.startsWith('/converted')) {
    return next();
  }
  res.sendFile(path.join(__dirname, '../client/website/dist',
'index.html'));
});
cleanupOldFiles();
setInterval(cleanupOldFiles, 1 * 60 * 1000);
```

Додаток В

Обробка завантаження файлів

```
const handleFileUpload = (req, res, next) => {
  upload.single("file")(req, res, (err) => {
    if (err instanceof multer.MulterError) {
      if (err.code === "LIMIT_FILE_SIZE") {
        return res.status(413).json({ error: "Файл перевищує максимальний
розмір 35 МБ." });}
      return res.status(400).json({ error: "Помилка завантаження файлу: "
+ err.message });
} else if (err) {
      return res.status(500).json({ error: "Невідома помилка
завантаження." });}
    if (!req.file) {
      return res.status(400).json({ error: "Файл не завантажено" });}
    try {
      req.file.originalname = Buffer.from(req.file.originalname,
'latin1').toString('utf8');
    } catch (e) { }
    req.uploadTime = new Date().toISOString();
    next();
  });
};
```

Додаток Г

Логіка конвертації файлів

```
router.post("/convert", apiLimiter, handleFileUpload, async (req, res) =>
{
  const { originalname, filename, path: inputPath } = req.file;
  const targetFormat = (req.body.format ? req.body.format.toLowerCase() :
"pdf").trim();
  const outputFileName = `${filename}.${targetFormat}`;
  const outputPath = path.join(convertedDir, outputFileName);

  try {
    const sourceExt =
path.extname(originalname).toLowerCase().replace('.', '').trim();
    const isSourceImage = IMAGE_FORMATS.includes(sourceExt);
    const isTargetImage = IMAGE_FORMATS.includes(targetFormat);

    if (sourceExt === targetFormat) {
      fs.copyFileSync(inputPath, outputPath);
    }
    else if (isSourceImage && isTargetImage) {
      await sharp(inputPath).toFile(outputPath);
}
    }
    else {
      const fileBuf = fs.readFileSync(inputPath);
      let filter = undefined;
      if (sourceExt === "pdf" && targetFormat === "docx") {
        filter = "Office Open XML Text";
      }
      const outputBuf = await libre.convertAsync(fileBuf, targetFormat,
filter);
      fs.writeFileSync(outputPath, outputBuf);
    }
```

Додаток Д

Робота із стисненням PDF

```
router.post("/compress-pdf", apiLimiter, handleFileUpload, async (req,
res) => {
  const { originalname, filename, path: inputPath } = req.file;
  const inputPathWithExt = inputPath + ".pdf";
  fs.renameSync(inputPath, inputPathWithExt);

  const outputFileName = `${filename}_compressed.pdf`;

  const outputPath = path.join(convertedDir, outputFileName);

  try {
    const fileBuf = fs.readFileSync(inputPathWithExt);
    const pdfDoc = await PDFDocument.load(fileBuf);
    const compressedDoc = await PDFDocument.create();
    const pages = await compressedDoc.copyPages(pdfDoc,
pdfDoc.getPageIndices());
    pages.forEach(page => compressedDoc.addPage(page));

    const compressedPdfBytes = await compressedDoc.save({
useObjectStreams: true, addDefaultPage: false });
    fs.writeFileSync(outputPath, compressedPdfBytes);

    const processEndTime = new Date().toISOString();

    const operationId = await logOperation(originalname, outputFileName,
"compress", "pdf", "pdf", req.uploadTime, processEndTime);

    setTimeout(() => {
      [inputPathWithExt, outputPath].forEach(p => { if (fs.existsSync(p))
fs.unlinkSync(p); });
      updateDeleteTime(operationId, new Date().toISOString());
    }, 5 * 60 * 1000);
```

Додаток Е

Реалізація модулів для роботи із зображеннями

```
// Стиснення PNG
router.post("/compress-png", apiLimiter, handleFileUpload, async (req,
res) => {
  const { originalname, filename, path: inputPath } = req.file;
  const outputFileName = `${filename}_compressed.png`;
  const outputPath = path.join(convertedDir, outputFileName);

  try {
    await sharp(inputPath).png({ quality: 80, compressionLevel: 9
}).toFile(outputPath);
    const processEndTime = new Date().toISOString();
    const operationId = await logOperation(originalname, outputFileName,
"compress", "png", "png", req.uploadTime, processEndTime);

    setTimeout(() => {
      [inputPath, outputPath].forEach(p => { if (fs.existsSync(p))
fs.unlinkSync(p); });
      updateDeleteTime(operationId, new Date().toISOString());
    }, 5 * 60 * 1000);

    const downloadName = originalname.replace(/\.[^/.]+$/, "") +
"_compressed.png";
    return res.json({
      success: true,
      downloadUrl:
`${req.protocol}://${req.get('host')}/api/download/${outputFileName}?name
=${encodeURIComponent(downloadName)}`,
      previewUrl:
`${req.protocol}://${req.get('host')}/converted/${outputFileName}`
    });
  } catch (err) {
    console.error("Compression error:", err);
    if (fs.existsSync(inputPath)) fs.unlinkSync(inputPath);
    return res.status(500).json({ error: "Помилка під час стиснення: " +
err.message });
  }
});

// Редагування фото (Фільтри, Розмір, Поворот)
router.post("/edit-photo", apiLimiter, handleFileUpload, async (req, res)
=> {
  const { originalname, filename, path: inputPath } = req.file;
  const edits = JSON.parse(req.body.edits || "{}");
  const extension = edits.format ? `.${edits.format.replace('.', '')}` :
(path.extname(originalname).toLowerCase() || '.jpg');
  const outputFileName = `edited_${filename}${extension}`;
```

```
  const outputPath = path.join(convertedDir, outputFileName);
  try {
    let transform = sharp(inputPath);
    if (edits.width && edits.height) {
      transform = transform.resize({ width: Math.round(edits.width),
height: Math.round(edits.height), fit: 'fill' });
    }
    if (edits.rotation) transform = transform.rotate(edits.rotation);
    if (edits.flip) transform = transform.flip();
    if (edits.flop) transform = transform.flop();
    transform = transform.modulate({ brightness: (edits.brightness ||
100) / 100, saturation: (edits.saturation || 100) / 100 });
    if (edits.grayscale) transform = transform.grayscale();
    if (edits.contrast && edits.contrast !== 100) {
      const slope = edits.contrast / 100;
      transform = transform.linear(slope, -(128 * slope) + 128);
    }
    if (edits.sharpen && edits.sharpen > 0) transform =
transform.sharpen({ sigma: edits.sharpen / 20 });
    if (edits.crop && edits.crop.width > 0 && edits.crop.height > 0) {
      transform = transform.extract({ left: Math.round(edits.crop.x),
top: Math.round(edits.crop.y), width: Math.round(edits.crop.width),
height: Math.round(edits.crop.height) });
    }

    await transform.toFile(outputPath);
    const processEndTime = new Date().toISOString();
    const operationId = await logOperation(originalname, outputFileName,
"edit", path.extname(originalname).replace('.', '').slice(1) || 'jpg',
extension.slice(1), req.uploadTime, processEndTime);

    setTimeout(() => {
      [inputPath, outputPath].forEach(p => { if (fs.existsSync(p))
fs.unlinkSync(p); });
      updateDeleteTime(operationId, new Date().toISOString());
    }, 5 * 60 * 1000);

    const downloadName = originalname.replace(/\.[^/.]+$/, "") +
"_edited" + extension;
    return res.json({
      success: true,
      downloadUrl:
`${req.protocol}://${req.get('host')}/api/download/${outputFileName}?name
=${encodeURIComponent(downloadName)}`,
      previewUrl:
`${req.protocol}://${req.get('host')}/converted/${outputFileName}`
    });
  } catch (err) {
    console.error("Editing error:", err);
    if (fs.existsSync(inputPath)) fs.unlinkSync(inputPath);
```

```
    return res.status(500).json({ error: "Помилка під час редагування: "
+ err.message });
  }
});

// Розділення фото на частини
router.post("/split-photo", apiLimiter, handleFileUpload, async (req,
res) => {
  const { originalname, filename, path: inputPath } = req.file;
  const splitType = req.body.type || "horizontal";
  const partsCount = parseInt(req.body.parts) || 2;
  const extension = path.extname(originalname).toLowerCase() || '.jpg';

  try {
    const metadata = await sharp(inputPath).metadata();
    const { width, height } = metadata;
    const partFiles = [];
    const crops = [];

    if (partsCount === 4) {
      const halfW = Math.floor(width / 2), halfH = Math.floor(height /
2);
      crops.push({ left: 0, top: 0, width: halfW, height: halfH });
      crops.push({ left: halfW, top: 0, width: width - halfW, height:
halfH });
      crops.push({ left: 0, top: halfH, width: halfW, height: height -
halfH });
      crops.push({ left: halfW, top: halfH, width: width - halfW, height:
height - halfH });
    } else {
      for (let i = 0; i < partsCount; i++) {
        if (splitType === "vertical") {
          const partW = Math.floor(width / partsCount);
          crops.push({ left: i * partW, top: 0, width: (i === partsCount
- 1) ? (width - (i * partW)) : partW, height: height });
        } else {
          const partH = Math.floor(height / partsCount);
          crops.push({ left: 0, top: i * partH, width: width, height: (i
=== partsCount - 1) ? (height - (i * partH)) : partH });
        }
      }
    }

    for (let i = 0; i < crops.length; i++) {
      const outName = `${filename}_part${i + 1}${extension}`, outPath =
path.join(convertedDir, outName);
      await sharp(inputPath).extract(crops[i]).toFile(outPath);
      partFiles.push({
        id: i + 1,
        url:
`${req.protocol}://${req.get('host')}/api/download/${outName}?name=${enco
```

```
deURIComponent(originalname.replace(/\.[^/.]+$/, "") + "_part" + (i + 1)
+ extension)}`,
        preview:
`${req.protocol}://${req.get('host')}/converted/${outName}`
      });
    }

    const operationId = await logOperation(originalname, `${partsCount}
parts`, "split", extension.replace('.', '').slice(1) || 'jpg',
extension.replace('.', '').slice(1) || 'jpg', req.uploadTime, new
Date().toISOString());
    fs.unlink(inputPath, () => { });
    setTimeout(() => {
      partFiles.forEach(pf => {
        const p = path.join(convertedDir,
pf.url.split('?')[0].split('/').pop());
        if (fs.existsSync(p)) fs.unlink(p, () => { });
      });
      updateDeleteTime(operationId, new Date().toISOString());
    }, 5 * 60 * 1000);
    res.json({ success: true, parts: partFiles });
  } catch (err) {
    console.error("Split error:", err);
    fs.unlink(inputPath, () => { });
    res.status(500).json({ error: "Помилка при розділенні фото: " +
err.message });
  }
});
```

Додаток Ж

Створення палітри по фото

```
router.post("/extract-palette", apiLimiter, handleFileUpload, async (req,
res) => {
  const { originalname, filename, path: inputPath } = req.file;
  const { generateImage, format, numColors } =
JSON.parse(req.body.options || "{}");
  const count = parseInt(numColors) || 5;
  try {
    const { data } = await sharp(inputPath).resize(count, 1, { fit:
'cover' }).raw().toBuffer({ resolveWithObject: true });
    const colors = [];
    for (let i = 0; i < data.length; i += 3) colors.push([data[i], data[i
+ 1], data[i + 2]]);
    const palette = colors.map(rgb => ({ rgb:
`rgb(${rgb[0]},${rgb[1]},${rgb[2]})`, hex: rgbToHex(rgb[0], rgb[1],
rgb[2]) }));

    let resultDownloadUrl = null;
    if (generateImage === "yes") {
      const targetFormat = format || "jpg", outputFilename =
`palette_${filename.split('.')[0]}.${targetFormat}`, outputPath =
path.join(convertedDir, outputFilename);
      const metadata = await sharp(inputPath).metadata();
      const paletteWidth = Math.round(metadata.width * 0.25), barHeight =
Math.round(metadata.height / count);

      const paletteCanvas = await sharp({ create: { width: paletteWidth,
height: metadata.height, channels: 4, background: { r: 255, g: 255, b:
255, alpha: 1 } } })
        .composite(colors.map((rgb, i) => ({
          input: { create: { width: paletteWidth, height: i === count - 1
? metadata.height - (barHeight * (count - 1)) : barHeight, channels: 3,
background: { r: rgb[0], g: rgb[1], b: rgb[2] } } },
          top: i * barHeight, left: 0
        }))).png().toBuffer();

      await sharp(inputPath).extend({ right: paletteWidth, background: {
r: 255, g: 255, b: 255, alpha: 1 } })
        .composite([{ input: paletteCanvas, left: metadata.width, top: 0
}]).toFormat(targetFormat).toFile(outputPath);
```

Додаток И

Утилітний модуль серверної частини

```
const fs = require("fs");
const path = require("path");
const libre = require("libreoffice-convert");
const sharp = require("sharp");
const { db } = require("./db");

sharp.cache(false);

process.env.LIBRE_OFFICE_EXE =
"D:\\Programs\\LibreOffice\\program\\soffice.exe";

libre.convertAsync = (document, format, filter, options = {}) => {
  return new Promise((resolve, reject) => {
    libre.convertWithOptions(document, format, filter, options, (err,
result) => {
      if (err) reject(err);
      else resolve(result);
    });
  });
};

const uploadsDir = path.join(__dirname, "uploads");
const convertedDir = path.join(__dirname, "converted");

if (!fs.existsSync(uploadsDir)) fs.mkdirSync(uploadsDir);
if (!fs.existsSync(convertedDir)) fs.mkdirSync(convertedDir);

const CLEANUP_AGE_MS = 5 * 60 * 1000;

const rgbToHex = (r, g, b) => {
  return "#" + ((1 << 24) + (r << 16) + (g << 8) +
b).toString(16).slice(1);
};

const cleanupOldFiles = () => {
  const now = Date.now();
  const dirsToClean = [uploadsDir, convertedDir];
  let deletedCount = 0;

  dirsToClean.forEach(dir => {
    if (!fs.existsSync(dir)) return;
    const files = fs.readdirSync(dir);
    files.forEach(file => {
      const filePath = path.join(dir, file);
      try {
        const stats = fs.statSync(filePath);
```

```
        const ageMs = now - Math.min(stats.birthtimeMs || stats.mtimeMs,
stats.mtimeMs);
        if (ageMs > CLEANUP_AGE_MS) {
          fs.unlinkSync(filePath);
          deletedCount++;
        }
      } catch (err) {
      }
    });
  });

  if (deletedCount > 0) {
    console.log(`Deleted ${deletedCount} old files.`);
    const fiveMinsAgo = new Date(now - CLEANUP_AGE_MS).toISOString();
    const currTime = new Date().toISOString();

    db.run(
      `UPDATE operations_history SET delete_time = ? WHERE delete_time IS
NULL AND upload_time < ?`,
      [currTime, fiveMinsAgo],
      function (err) {
        if (!err && this.changes > 0) {
          console.log(`Updated ${this.changes} database records.`);
        }
      }
    );
  }
};

module.exports = {
  libre,
  sharp,
  uploadsDir,
  convertedDir,
  rgbToHex,
  cleanupOldFiles,
  CLEANUP_AGE_MS
};
```

Додаток К

Логіка входу на адмін-панель

```
const ADMIN_PASS = "pdfMaster2026!";
const isAuthenticated = (req, res, next) => {
  const cookies = req.headers.cookie || "";
  if (cookies.includes("admin_auth=loggedIn")) {
    next();
  } else {
    res.status(401).json({ error: "Unauthorized" });
  }
};

router.get("/check-auth", (req, res) => {
  const cookies = req.headers.cookie || "";
  if (cookies.includes("admin_auth=loggedIn")) {
    res.json({ authenticated: true });
  } else {
    res.json({ authenticated: false });
  }
});
router.post("/login", (req, res) => {
  const { username, password } = req.body;
  if (username === ADMIN_USER && password === ADMIN_PASS) {
    res.setHeader("Set-Cookie", "admin_auth=loggedIn; HttpOnly; Path=/;
Max-Age=3600; SameSite=Lax");
    res.json({ success: true, message: "Вхід успішний" });
  } else {
    res.status(401).json({ success: false, error: "Неправильний логін або
пароль" });
  }
});

router.post("/logout", (req, res) => {
  res.setHeader("Set-Cookie", "admin_auth=; HttpOnly; Path=/; Max-Age=0;
SameSite=Lax");
  res.json({ success: true });
});
```

Додаток Л

Реалізація сторінки адмін-панелі

```
  try {
    const uploadedFiles = fs.existsSync(uploadsDir) ?
fs.readdirSync(uploadsDir) : [];
    const convertedFiles = fs.existsSync(convertedDir) ?
fs.readdirSync(convertedDir) : [];
    const getLogs = () => new Promise((resolve, reject) => {
      db.all("SELECT * FROM operations_history ORDER BY id DESC LIMIT
200", [], (err, rows) => {
        if (err) reject(err);
        else resolve(rows);
      });
    });
    const logs = await getLogs();

    res.json({
      success: true,
      stats: {
        uploadedCount: uploadedFiles.length,
        convertedCount: convertedFiles.length,
        uploadedFiles: uploadedFiles,
        convertedFiles: convertedFiles
      },
      logs: logs
    });
  } catch (error) {
    res.status(500).json({ error: "Помилка при отриманні даних: " +
error.message });
  }
});
```

Додаток М

Реалізація сторінки для завантаження файлу

```
{state === "converting" && (
        <div className="converting-state">
           <div className="spinner"></div>
           <h2>Йде стиснення PNG... Зачекайте будь ласка.</h2>
        </div>
      )}

      {state === "result" && (
        <div className="result-state">
           <div className="preview-container">
             <img src={previewUrl} alt="Preview" className="preview-img"
style={{maxWidth: "100%", maxHeight: "100%", objectFit: "contain"}} />
           </div>
           <div className="download-container">
             <a
               href={downloadUrl}
               className="method-btn download-btn"
             >
               Завантажити файл
             </a>
             <p className="warning-text">Даний файл буде видалено із
сайту через 5 хвилин для надійності безпеки ваших даних.</p>
             <button className="method-btn secondary-btn"
style={{marginTop: "20px", background: "transparent", border: "2px solid
#fff"}} onClick={resetPage}>Стиснути інший файл</button>
           </div>
        </div>
      )}
```

Додаток Н

Основні стилі сайту

```
:root {
  --gray-bg: #131111;
  --theme-red: #476f53;
  --text-white: #ffffff;
  --text-black: #000000;
  --bg-light: #f5f5f5;
}
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #fff;
  color: var(--text-black);
  line-height: 1.6;
}
.app-container {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
}
.main-content {
  flex: 1;
  display: flex;
  flex-direction: column;
  padding: 40px 0;
}
button {
  cursor: pointer;
}
```