МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи

ОС Магістр

на тему: Дослідження продуктивності роботи веб-додатків з різними форматами зображень (на прикладі gif, jpg, png, swf та webp)

за освітньою програмою: 12 Інженерія програмного забезпечення

зі спеціальності: 121 Інженерія програмного забезпечення

Виконала: студентка групи ПЗ2422: /Марія ПОСМІТЮХА/

Керівник: /Тетяна ГРИШЕЧКІНА/

Нормоконтролер: /Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро - 2026 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»
Department «Computer information technology»

Explanatory Note
to Master's Thesis

on the topic: Researching the performance of web applications with different image formats (using gif, jpg, png, swf and webp as an example)
according to educational curriculum **12 Software engineering**
in the Speciality: **121 Software engineering**

Done by the student of the group PZ2422: /Mariia POSMITIUKHA/

Scientific Supervisor: /Tatyana HRYSHECHKYNA/

Normative controller: /Svitlana VOLKOVA/

Dnipro - 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Український державний університет науки і технологій

Факультет: Комп'ютерні технології і системи

Кафедра: Комп'ютерні інформаційні технології

Рівень вищої освіти: магістр

Освітня програма: «12 Інженерія програмного забезпечення»

Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
__/Вадим ГОРЯЧКІН/
«02» жовтня 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентці Посмітюсі Марії Олександрівні

1. Тема роботи: «Дослідження продуктивності роботи веб-додатків з різними форматами зображень (на прикладі gif, jpg, png, swf та webp)»

   Керівник роботи: к.т.н, доцент Гришечкіна Тетяна Сергіївна

   затверджені наказом від "02" жовтня 2025 р. №1401ст
2. Строк подання студентом роботи: 09.01.2026р.
3. Вихідні дані до роботи: розроблений веб-додаток для ітераційного аналізу продуктивності медіа-контенту; тестові набори статичних та анімованих зображень (PNG, JPEG, GIF, WebP); дослідження продуктивності формату SWF алгоритми розрахунку метрик Core Web Vitals (LCP, FCP) та часу декодування; результати статистичної обробки 1440 тестів.
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):

2026

4.1. Аналіз сучасних форматів зображень та їх впливу на продуктивність веб-додатків. (Дослідження теоретичних характеристик WebP, AVIF та порівняння зі стандартними форматами);

4.2. Обґрунтування методики автоматизованого дослідження характеристик завантаження та рендерингу графіки. (Вибір показників Load Time, Decode Time, LCP, FCP, PLT, File Size, Compression Ratio та методів статистичного доведення значущості);

4.3. Проєктування та розробка програмного інструментарію для ітераційного аналізу медіа-файлів. (Опис архітектури додатка, системи збору метрик у пісочниці та модуля візуалізації статистики);

4.4. Експериментальне дослідження ефективності використання формату WebP та порівняльний аналіз результатів у різних браузерних рушіях. (Аналіз даних Chrome vs Safari, перевірка статистичних гіпотез на великих вибірках), дослідження формату SWF окремо.

5. Перелік графічного матеріалу:

5.1. Презентація

5.2. Демонстраційне відео

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Вступ	01.10.25 - 05.10.25	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	06.10.25 - 20.10.25	
3	Аналіз сучасного стану програмно-апаратного забезпечення	21.10.25 - 31.10.25	
4	Постановка задачі, технічне завдання	01.11.25 - 10.11.25	30%
5	Узгодження та затвердження ТЗ	11.11.25 - 15.11.25	
6	Виконання досліджень, збір пакету зображень для дослідження, проведення дослідження	16.11.25 - 20.12.25	60%
7	Оформлення пояснювальної записки	21.12.25 - 10.01.26	
8	Розробка демонстраційних матеріалів	11.01.26 - 18.01.26	100%
9	Подання кваліфікаційної роботи до кафедри	19.01.26 - 22.01.26	
10	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	25.01.26 - 31.01.26	

Студентка _____ Марія ПОСМІТЮХА

Керівник роботи _____ Тетяна ГРИШЕЧКІНА

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра:

81с., 18 рис., 85табл., 4 додатки, 36 джерел.

Об’єкт розробки - процеси оцінювання продуктивності додатків на основі оцінювання часових показників рендерингу веб-сторінок при використанні різних форматів графічних даних (GIF, JPG, PNG, SWF та WebP) у сучасних браузерах.

Мета роботи - кількісна оцінка та порівняльний аналіз впливу різних форматів зображень на швидкість завантаження контенту та загальну продуктивність веб-додатків.

Методи дослідження - експериментальне дослідження з використанням розробленого спеціалізованого веб-додатку (на базі JS), автоматизоване вимірювання часу повного завантаження об’єктів за допомогою вбудованих інструментів розробника (DevTools) та порівняльний аналіз обсягу переданих даних.

У роботі проведено порівняльне тестування продуктивності веб-сторінок із різними форматами зображень у ізольованому середовищі актуальних веб-браузерів, що дозволило кількісно визначити переваги певних форматів над іншими. Розроблено програмний інструмент, що проводить автобатичне тестування та збір метрик завантаження сторінок, який підтвердив можливість скорочення обсягу трафіку до 30-40% без втрати візуальної якості при переході на певний формат. Отримані дані мають практичне значення для веб-розробників при проектуванні високонавантажених інтерфейсів з великою кількістю графічного контенту.

Результати роботи можуть стати основою для розробки рекомендацій щодо оптимізації фронтенд-частини веб-проектів, що дозволить зменшити навантаження на мережеві канали та прискорити взаємодію користувача з інтерфейсом веб-додатків.

Ключові слова: ВЕБ-ДОДАТОК, ПРОДУКТИВНІСТЬ, ФОРМАТИ ЗОБРАЖЕНЬ, РАСТРОВІ ЗОБРАЖЕННЯ, СТИСНЕННЯ ДАНИХ, CORE WEB VITALS, LOAD TIME, DECODE TIME, LCP, FCP, PLT, ТЕСТ ВІЛКОКСОНА

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СКОРОЧЕНЬ І ТЕРМІНІВ

CWV - Core Web Vitals.

JPG/JPEG (Joint Photographic Experts Group) - популярний формат растрових зображень, що підтримує стиснення з втратами та без.

PNG (Portable Network Graphics) - формат растрових зображень, що підтримує стиснення без втрат та прозорість.

SWF (Small Web Format) - застарілий формат для флеш-анімації та векторної графіки.

WebP - сучасний формат зображень, що забезпечує стиснення з втратами та без втрат, анімації для графіки в Інтернеті.

GIF (Graphics Interchange Format) - растровий формат зображень із 8-бітною палітрою кольорів та підтримкою покадрової анімації.

FCP (First Contentful Paint) - час від початку завантаження сторінки до моменту відмальовки першого візуального елемента.

LCP (Largest Contentful Paint) - ключова метрика Core Web Vitals, що фіксує час рендерингу найбільшого графічного об'єкта на екрані.

PLT (Page Load Time) - сумарний час повного завантаження сторінки з усіма її ресурсами.

ВСТУП

Зображення є одним із найбільш критичних елементів сучасних веб-додатків, оскільки вони визначають візуальну привабливість та здатність захоплювати увагу користувачів швидше за текстовий контент [1]. За даними аналізу HTTP Archive [2], зображення становлять майже 41% від загального обсягу переданих даних веб-сайту, а медіанний розмір графічних файлів на одній сторінці вже перевищує 1010 КБ.

Такий “важкий” контент безпосередньо впливає на продуктивність веб-додатків [3]. Продуктивність веб-додатків - це показник, який визначає, наскільки швидко завантажуються сторінки та як оперативно вони реагують на дії користувача, що є ключовим аспектом користувацького досвіду [4]. Від цього показника безпосередньо залежить, чи залишиться відвідувач на сайті, чи покине сторінку.

Для мобільних пристроїв, у 68% випадків саме зображення є Largest Contentful Paint (LCP) - один із ключових показників продуктивності та швидкості завантаження сторінки [2]. Передача великих обсягів даних без попередньої оптимізації призводить не лише до повільного завантаження та поганого користувацького досвіду, а й до зростання енергоспоживання пристроїв.

Незважаючи на існування сучасних форматів, наприклад WebP, які пропонують краще стиснення порівняно з іншими, на ринку досі домінують JPEG та PNG (разом становлять понад 60% усіх зображень у всесвітній мережі) [2] та GIF для передачі покадрової анімації. Такий розподіл створює логічне протиріччя: технологій, що здатні суттєво прискорити роботу додатків існують, а розробники продовжують використовувати менш ефективні формати через інерцію або побоювання щодо сумісності, менеджери контенту не використовують сучасні формати.

Ще одним викликом є вихід з ужитку застарілих технологій, таких як SWF, які більше не підтримуються сучасними браузерами, що вимагає їхньої повної заміни на продуктивніші аналоги [5].

Існуючі інструменти аналізу продуктивності (наприклад, PageSpeed Insights [6] та Lighthouse [7] від Google) надають загальну оцінку сторінки, але не дозволяють ізолювати вплив виключно зображення і його формату від впливу сторонніх скриптів, стилів чи налаштувань сервера. Також, є сервіси, такі як Squoosh [1], які дозволяють детально порівнювати якість та вагу файлу при зміні форматів і алгоритмів стиснення, але вони призначені для поштучної обробки і позбавлені функціоналу для масового збору системних метрик, таких як час декодування процесором або швидкість рендерингу в реальних умовах браузера, до того ж погано працюють із зображеннями без фону.

Також у багатьох дослідженнях ігнорується вимога зіставної візуальної якості: порівняння форматів часто проводиться на зображеннях із різним ступенем спотворення, що робить висновки неточними.

Розроблена програма усуває ці недоліки, поєднуючи пакетне тестування з вбудованим математичним аналізом результатів безпосередньо в середовищі користувача.

Метою кваліфікаційної роботи є кількісне дослідження та доведення впливу форматів зображень на показники продуктивності веб-додатків у лабораторному, обмеженому середовищі.

Для досягнення мети було розроблено спеціальне програмне забезпечення. В рамках роботи вирішуються наступні завдання:

- Створення ізольованого тестового середовища для вимірювання метрик продуктивності;
- Автоматизація процесу конвертації та тестування великої вибірки зображень;
- Збір статистики та перевірка її значущості.

Актуальність роботи: дослідження узгоджується із сучасними напрямами розвитку веб технологій, де швидкість завантаження веб сторінок та оптимізація

медіаданих є критичними факторами якості програмного забезпечення у веб розробці.

Об'єкт дослідження: процеси передавання, обробки та графічного контенту у вебдодатках на базі сучасних браузерів.

Предмет дослідження: методи та інструментальні засоби порівняльного аналізу показників продуктивності (швидкість завантаження, час декодування, LCP та інші) форматів зображень GIF, JPG, PNG та WebP.

Метою роботи є дослідження впливу вибору формату зображення на продуктивність веб додатків шляхом кількісного порівняння метрик швидкості та ефективності стиснення та переводу форматів за допомогою розробленого ПЗ.

Наукова новизна полягає в розробці методики комплексного автоматизованого оцінювання впливу сучасних форматів зображень на продуктивність веб додатків шляхом фіксації специфічних метрик продуктивності. Запропоновано поєднання пакетного тестування (Batch Testing) із вбудованим математичним апаратом для підтвердження статистичної значущості отриманих результатів у режимі реального середовища користувача.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття продуктивності веб-додатків

Продуктивність веб-додатків - це комплексний показник, що визначає, наскільки швидко, стабільно і ефективно працює веб сайт під час взаємодії користувачів з ним. Продуктивність охоплює як технічні параметри (час відгуку сервера, швидкість завантаження, споживання ресурсів), так і особисті враження користувача, що формуються під час використання сайту. Висока продуктивність означає, що користувач може швидко отримати доступ до потрібної інформації, взаємодіяти з інтерфейсом без затримок, а візуальні елементи сторінки відображаються плавно та стабільно.

У сучасних умовах веб розробки продуктивність стала ключовим чинником конкурентоспроможності. За статистикою Google, якщо завантаження сторінки триває понад три секунди, понад 50 % користувачів залишають сайт [1]. Це свідчить, що навіть мінімальні затримки можуть суттєво вплинути на рівень задоволеності користувачів і, відповідно, на комерційні показники - конверсію, середній час перебування на сайті, позиції в пошукових системах тощо.

Продуктивність тісно пов’язана з поняттям User Experience (UX) - якості користувацького досвіду. Навіть якщо веб додаток має привабливий дизайн та широкий функціонал, повільна робота чи нестабільне відображення елементів знижують загальну оцінку сайту користувачами. Продуктивність оцінюється за допомогою показників Web Vitals та Core Web Vitals.

1.2. Метрики продуктивності. Показники Web Vitals та Core Web Vitals

Web Vitals - це ініціатива Google, спрямована на надання загальних єдиних рекомендацій щодо сигналів якості, необхідних для забезпечення гарного користувацького досвіду та загальної продуктивності сторінок. Протягом багатьох років Google надав низку інструментів для вимірювання та надання звітності про ефективність.

До Web Vitals відноситься ряд показників продуктивності із різних областей. Наприклад показники завантаження та рендерингу (Loading Vitals), Показники

інтерактивності та JS (Interactivity Vitals), Метрики мережі та ресурсів (Resource Vitals), Метрики стабільності та довготривалого досвіду тощо. Не всі із них потрібні для дослідження продуктивності в залежності від формату зображень.

Ініціатива Web Vitals має на меті спростити оптимізацію та допомогти сайтам зосередитися на найважливіших показниках - Core Web Vitals. (по суті, показники Core Web Vitals - це підмножина показників Web Vitals).

До Core Web Vitals відносяться:

- Largest Contentful Paint (LCP): момент, коли основний вміст сторінки завантажився - швидкий LCP допомагає переконати користувача, що сторінка корисна;
- Interaction to Next Paint (INP): вимірює інтерактивність (не важливий показник в рамках дослідження);
- Cumulative Layout Shift (CLS): вимірює візуальну стабільність сторінки (не важливий показник в рамках дослідження).

1.3. Показники, що використовувались у дослідженні

Показники, що фіксувались у ході дослідження:

Для проведення комплексного дослідження ефективності різних форматів зображень у дипломній роботі обрано систему метрик, що охоплює як суб'єктивний досвід користувача (User Experience), так і об'єктивні технічні параметри функціонування браузерного середовища.

Ключовим інструментом аналізу є система Web Vitals. У межах роботи досліджуються такі показники:

- Largest Contentful Paint (LCP) - відображає час, необхідний для рендерингу найбільшого візуального об'єкта в області перегляду. Оскільки у сучасних інтерфейсах таким об’єктом найчастіше є графічний банер або медіаконтент, LCP є критично важливим для оцінки того, як вибір формату впливає на візуальну швидкість завантаження додатка. Слід зауважити, що LCP є інтегральною метрикою. Вона включає в себе час до отримання першого байта (TTFB), час до першого відмалювання (FCP) та безпосередній час завантаження і декодування цільового ресурсу. Таким

чином, оптимізація зображення впливає на LCP через ланцюжок суміжних показників. У дослідженні цей показник виступає головним критерієм якості вебдодатка, поєднуючи в собі час мережевого завантаження та швидкість обробки файлу браузером;

- First Contentful Paint (FCP) - фіксує момент появи першого візуального елемента на екрані. Аналіз FCP у роботі дозволяє встановити, наскільки швидко браузер переходить від стану очікування до відображення первинного контенту при використанні різних типів стиснення. У дослідженні FCP виступає показником того, як швидко користувач отримує перший візуальний сигнал про працездатність додатка. Це дозволяє встановити «поріг очікування» для різних форматів. Якщо LCP фіксує момент повного відображення основного об'єкта, то FCP показує самий початок цього процесу. Порівняння цих двох метрик у вашій роботі дозволяє проаналізувати динаміку «поступового» завантаження різних форматів ;
- Page Load Time (PLT) / Load Event - показник загальної готовності сторінких, що фіксується подією window.onload. Вона спрацьовує лише тоді, коли завантажено абсолютно все:HTML-структура та CSS-стилі, усі скрипти (якщо вони не позначені як async або defer), усі зображення, іфрейми та об'єкти). На відміну від LCP, PLT фіксує повний життєвий цикл сторінки. У дослідженні це дозволить порівняти, скільки часу займає весь процес - від ініціалізації вікна до фінального відображення конкретного формату (наприклад, порівняння «важкого» PNG та оптимізованого WebP).

Окрім показників сприйняття, для глибокого аналізу форматів необхідно враховувати технічні метрики, витрати ресурсів на стороні клієнта:

- Decode Time (Час декодування) - тривалість перетворення стиснених даних зображення у растрову сітку пікселів силами CPU або GPU. На відміну від метрик завантаження, це показник обчислювальної складності. Важливість цієї метрики полягає у виявленні «прихованих» затримок: формати з високим ступенем стиснення можуть швидше передаватися по мережі, але

вимагати значних ресурсів на розпакування, що критично для пристроїв з низькою продуктивністю;

- Load Time (Resource Timing) - чистий час передачі конкретного файлу мережею. Ця метрика дозволяє ізолювати вплив мережевої інфраструктури від процесів рендерингу, що дає змогу порівняти саме «транспортну» ефективність форматів;
- File Size (Розмір файлу) - фізичний обсяг даних у байтах. Є базовим параметром, що визначає мережеве навантаження.Вимірюється розмір файлу після конвертації. Зауваження: оскільки формат .swf є застарілим і не підтримується більшістю браузерів, розмір файлу - єдиний показник, що вимірюєтьсяя у ході дослідження. Відкрити формат можливо за допомогою плагінів, які суттєво заважають дослідженню, істотно вплваючи на Load Time та Decode Time;
- Compression Ratio (Коефіцієнт стиснення) - розрахунковий показник, що відображає відсоткове співвідношення між розміром оптимізованого файлу та його оригіналом. Використовується для математичного обґрунтування переваг алгоритмів стиснення, що розглядаються у роботі.

Порогові та оптимальні значення (бенчмарки) для кожної метрики, застосованої в дослідженні представлені у таблиці 1.

Сукупність обраних метрик дозволяє сформувати багатогранну модель продуктивності, де ефективність кожного формату оцінюється не лише за вагою файлу, а й за його впливом на швидкість відображення контенту та навантаження на апаратну частину клієнтського пристрою.

Таблиця 1.1 - Порогові та оптимальні значення для метрик продуктивності

Показник	Хороше	Потребує покращення	Погано
LCP (Largest Contentful Paint)	≤ 2.5 с	2.5 с – 4.0 с	> 4.0 с

Продовження таблиці 1.1

FCP (First Contentful Paint)	≤ 1.8 с	1.8 с – 3.0 с	> 3.0 с
PLT (Page Load Time)	≤ 3.0 с	3.0 с – 5.0 с	> 5.0 с
Decode Time	≤ 50 мс	50 мс – 150 мс	> 150 мс
Load Time (1 ресурс)	≤ 500 мс	0.5 с – 1.0 с	> 1.0 с

1.4. Технічна характеристика та особливості форматів зображень

Далі подано опис форматів, що розглядаються у межах дослідження.

JPEG (Joint Photographic Experts Group)

JPEG (Joint Photographic Experts Group) - є найпоширенішим форматом для фотографій завдяки ефективному стисненню з втратами (lossy). Формат дозволяє варіювати ступінь стиснення, у розробленому інструменті формат тестується у двох варіантах: 90% та 75%, що дозволяє проаналізувати баланс між візуальною якістю та розміром файлу, порівняти із іншим форматом із варіативними ступенями стиснення.

Формат підтримує 24-бітний колір, оптимальний для фотографій, але не підтримує прозорість (альфа-канал). Більшість камер використовують JPEG як основний формат для економії місця на картах пам'яті при збереженні прийнятної якості. [1]

При високих ступенях стиснення можуть з'являтися помітні артефакти, особливо на чітких межах об'єктів та тексті. Однією з ключових особливостей JPEG є те, що це формат з втратами (lossy). Стиснення відбувається шляхом процесу квантування: При збереженні алгоритм розбиває зображення на блоки 8x8 пікселів і відкидає «зайву» інформацію, яку людське око майже не помічає. Це несе за собою недоліки формату: коли ви відкриваєте JPEG, редагуєте його і знову натискаєте «Зберегти», браузер або редактор знову розбиває зображення на блоки і знову застосовує математичне округлення. Повторні округлення призводять до появи нових артефактів. Іноді це проявляється як «блочність» (квадрати 8 на 8 стають помітними), «дзвону» (шум навколо контрастних ліній)

та втрати дрібних деталей (текстура шкіри перетворюється на розмите плямисте поле).

PNG (Portable Network Graphics)

PNG (Portable Network Graphics) - формат, що використовує стиснення без втрат (lossless). Це означає, що після “розпакування” зображення кожен піксель буде на 100% ідентичним оригіналу.

PNG ідеально підходить для логотипів, іконок та скріншотів, де важливо зберегти чіткість деталей, проте розмір файлу зазвичай значно більший, ніж у JPEG та інших популярних форматів. В основі PNG лежить алгоритм DEFLATE, який поєднує два методи: LZ77 (Lempel-Ziv) - знаходить послідовності байтів, що повторюються, і замінює їх на посилання; кодування Хаффмана - стискає дані, призначаючи коротші коди символам, що зустрічаються найчастіше. Інша особливість PNG - етап пре-фільтрації. Перед стисненням кожен рядок пікселів проходить через спеціальні математичні фільтри. Замість того, щоб зберігати колір кожного пікселя, PNG зберігає різницю між поточним пікселем та його сусідом. Оскільки в графіці та інтерфейсах кольори сусідніх пікселів часто однакові (наприклад, рівний фон), різниця між ними дорівнює нулю. Нулі стискаються алгоритмом DEFLATE набагато ефективніше, ніж хаотичні кольори. Таким чином, PNG є стандартом для скриншотів та інфографіки.

Головна перевага - альфа канал (підтримка прозорості) та висока глибина кольору. Кожен піксель має не лише координати кольору (RGB), а й значення Alpha (від 0 - повністю прозорий, до 255 - повністю непрозорий).

PNG зберігає всю високочастотну інформацію (шум, дрібні деталі), яку JPEG просто викидає. PNG - математично точний, що у випадку з фотографіями створює величезні, неоптимізовані обсяги даних.

WebP (Web Picture Format)

WebP (Web Picture Format) - це сучасний растровий формат зображень, розроблений компанією Google у 2010 році з метою створення менших за розміром та більш якісних зображень для прискорення роботи із файлами в

інтернеті. Він є універсальною заміною для JPEG, PNG та GIF, оскільки поєднує їхні найкращі властивості.

Підтримує як стиснення з втратами (lossy), так і без втрат (lossless). Зображення WebP в середньому менші, ніж аналогічні файли JPEG при ідентичному індексі візуальної схожості (SSIM). Конфігурації з якістю 90% та 75% дозволяють продемонструвати суттєве зменшення ваги файлу без появи характерних для JPEG «блочних» артефактів.

На відміну від JPEG, цей режим підтримує прозорість (альфа-канал).

Підтримує покадрову анімацію (Animated WebP). Формат WebP є найефективнішою сучасною альтернативою застарілому формату GIF. На відміну від GIF, який обмежений 256 кольорами, анімований WebP підтримує 24-бітний колір та 8-бітний альфа-канал (тобто напівпрозорість).

Не підтримується старими браузерами.

GIF (Graphics Interchange Format)

GIF - один із найстаріших форматів растрової графіки, розроблений у 1987 році [1]. Формат використовує 8-бітну палітру кольорів, що обмежує кількість одночасно відображуваних відтінків до 256. Замість зберігання значень RGB для кожного пікселя, GIF використовує палітру (Color Table). Кожен піксель містить лише індекс кольору з цієї таблиці, що дозволяє суттєво зменшити розмір файлу для простої графіки, проте призводить до помітної втрати глибини кольору в складних сценах.

В основі GIF лежить алгоритм стиснення LZW (Lempel-Ziv-Welch) - метод стиснення без втрат, який ефективно працює з великими областями однорідного кольору, але демонструє низьку ефективність на фотографічних зображеннях.

Механізм анімації реалізований через розширення блоків керування графікою, що дозволяє створювати циклічні ролики без використання відеокодеків.

Формат підтримує прозорість, але лише абсолютну. Це означає, що піксель може бути або повністю прозорим, або повністю непрозорим, відсутні

напівпрозорі фони, градієнти. Також, є підтримка черезрядкової розгортки, що дає змогу бачити низькоякісну версію зображення ще до повного завантаження.

Незважаючи на недоліки, GIF залишається стандартом для коротких анімаційних послідовностей, цифрових наклейок та простих графічних елементів (логотипів, схем) з обмеженою кількістю кольорів. У контексті продуктивності вебдодатків GIF часто розглядається як формат з високими витратами трафіку при низькій візуальній якості анімації, що робить його важливим об'єктом для порівняльних досліджень. Станом на 2024 рік GIF посідає третє місце серед усіх форматів у вебі, займаючи 16,8% від загальної кількості зображень, використання не зменшується, а навпаки - за останні два роки його частка в мережі зросла на 1 відсотковий пункт.

Цікаво, що лише приблизно 32% усіх GIF-файлів, що зустрічаються на сторінках, є анімованими [1].

SWF (Small Web Format)

SWF (Small Web Format) - це формат для мультимедіа та інтерактивної векторної анімації, що тривалий час був стандартом для веб додатків. Його архітектура базується на поєднанні векторної графіки, растрових даних та мови програмування ActionScript.

Ключові характеристики:

- Векторна основа: дозволяє масштабувати контент без втрати якості, забезпечуючи мінімальний розмір файлів.
- Тег-орієнтована структура: файл складається із заголовка та послідовності тегів (блоків даних). Теги визначення створюють об'єкти (форми, звуки), теги керування маніпулюють ними на часовій шкалі.
- Система координат: для точності відтворення використовується одиниця твіп (twip), що дорівнює $1/20$ логічного пікселя.
- Стиснення: підтримує алгоритми ZLIB та LZMA для зменшення обсягу даних при передачі мережею.

Незважаючи на багатофункціональність, формат мав критичні недоліки: низький рівень безпеки, високе енергоспоживання на мобільних пристроях та

закритість технології. У 2020 році компанія Adobe офіційно припинила підтримку SWF, поступившись місцем відкритим стандартам стеку HTML5 (Canvas, WebGL, SVG).

1.5. Методи забезпечення точності дослідження

Окрім показників, що безпосередньо вимірюють продуктивність є додаткові метрики, що впливають на точність дослідження та статистичну значущість результатів. Потрібно бути певними, що різниця в результатах продуктивності зумовлена саме форматом зображень, а не зовнішніми чинниками чи нетовній постановці експерименту.

Критерій Вілкоксона

Для об’єктивної оцінки результатів дослідження недостатньо порівняння лише значень, оскільки показники швидкодії вебдодатків (Load Time, LCP) часто мають аномальні викиди через коливання мережі та не підпорядковуються закону нормального розподілу. Тому у дослідженні використано непараметричний статистичний апарат - критерій Вілкоксона (Wilcoxon Signed-Rank Test). Цей метод дозволяє справедливо порівняти показники одного і того самого набору тестових даних і сказати, але із використанням двох різних форматів: чи зафіксована різниця у швидкодії статистично значуща, чи вона випадкова. Ключовий момент - порівняння відбувається лише між двома фотрматами.

Показник p-value - результатів тесту. Якщо значення $p < 0.05$, - реальний вплив обраного типу стиснення на продуктивність підтверджено. Якщо ж $p > 0.05$, різниця вважається випадковою, а отже дослідження не має практичної користі.

SSIM

SSIM (Structural Similarity Index Measure) - це сучасна метрика, яка використовується для прогнозування того, саме людиною буде сприйнята схожість цифрових зображень. На відміну від суто математичних методів, цей показник враховує структурні зміни в візуальній інформації, а не лише абсолютні помилки в пікселях. Простими словами - на скільки картинки схожі одна на одну

з погляду людського ока. Діапазон значень від -1 до 1. Значення 1 вказує на ідеальну візуальну схожість із оригіналом, а 0 означає повну відсутність структурної схожості. У дослідженні показник SSIM наближено 0.95, «майже досконало подібні», що гарантує, що різниця в показниках продуктивності спричинені саме формату, а не ступенем візуального спрощення.

1.6. Огляд аналогів

Google Lighthouse - це автоматизований інструмент з відкритим вихідним кодом для покращення якості веб-сторінок. Він проводить аудит за кількома напрямками: продуктивність (Performance), доступність (Accessibility) та SEO.

Переваги: дозволяє отримати комплексний звіт про стан сторінки, включаючи показники Core Web Vitals (LCP, FID, CLS).

Недоліки для даного дослідження: Lighthouse орієнтований на аудит всієї сторінки в цілому. Він не дозволяє проводити ізольоване ітераційне тестування одного й того самого зображення у різних форматах з високою точністю вимірювання часу декодування на рівні мікросекунд.

PageSpeed Insights - це веб-сервіс, який надає звіти про продуктивність сторінки як на мобільних, так і на десктопних пристроях, використовуючи дані Lighthouse та реальні дані з досвіду користувачів (CrUX).

Переваги: надає рекомендації щодо оптимізації, зокрема часто пропонує "використовувати сучасні формати зображень" (WebP або AVIF).

Недоліки для даного дослідження: PSI є "чорною скринькою" щодо методів вимірювання. Він показує результати завантаження готової сторінки, але не дає можливості програмно керувати параметрами експерименту (кількість ітерацій, затримки між тестами).

1.7. Огляд зарубіжних досліджень

Огляд інших досліджень показав, переваги сучасних форматів зображень над більш застарілими.

Всі одностайно наголошують на перевагах WebP у порівнянні з JPEG, PNG та GIF

За даними дослідження [1] WebP забезпечує в середньому на 30% краще стиснення, ніж JPEG та JPEG 2000, без втрати візуальної якості. Також файли WebP на 25–34% менші за аналогічні JPEG при однаковому рівні якості.

Самі творці формату стверджують, [1] що зображення WebP без втрат на 26% менші за розміром порівняно з PNG. Зображення WebP з втратами на 25-34% менші за аналогічні зображення JPEG з еквівалентним індексом якості SSIM. Використання WebP замість стисненого JPEG покращує показник Page Load Time (PLT) на 21%. Завдяки меншому розміру файлів WebP значно прискорює завантаження основного контенту (LCP). Загалом, підкреслюється універсальність формату.

Відносно анімованих зображень дослідники також наголошують на суттєвій перевазі відеоформатів із відповідними налаштуваннями у стилях для досягнення циклічності відображення відео, аби “імітувати” звичний формат GIF. Дослідження також підкреслюють перевагу формату WebP для анімованих зображень.

Висновок до розділу 1

У першому розділі проведено масштабний теоретичний аналіз поняття продуктивності вебдодатків та факторів, що впливають на продуктивність. Встановлено, що в умовах сучасної веб-розробки швидкість завантаження безпосередньо впливає на рівень задоволення користувачів та комерційним успіхом сайтів.

Досліджено Web Vitals від Google та виокремлено ключові метрики, що є значущими в контексті дослідження продуетивності веб додатків та стали основою для подальшого експерименту. Обґрунтовано доцільність використання показників. Встановлення порогові та рекомендовані значення для цих метрик, дозволяє не лише констатувати показник як факт, а й інтерпретувати їх з точки зору реального впливу на продуктивність.

Проведено детальний порівняльний аналіз технічних характеристик форматів зображень: JPEG, PNG, WebP, GIF та SWF. Виявлено, що класичні формати (JPEG, PNG) часто створюють надлишкове навантаження на мережу через

обмеженість алгоритмів стиснення, тоді як сучасний формат WebP пропонує універсальне рішення, поєднуючи переваги стиснення з втратами та без них при меншій вазі файлів. Особливу увагу приділено формату анімованого WebP як ефективній альтернативі застарілому GIF, та формату SWF, чия актуальність була втрачена через припинення підтримки технології Flash.

Для забезпечення високої точності та наукової достовірності дослідження було визначено методологію статистичного аналізу. Використання непараметричного критерію Вілкоксона дозволяє підтвердити статистичну значущість отриманих різниць у продуктивності, нівелюючи вплив випадкових коливань мережі. Водночас впровадження метрики SSIM на рівні 0.95 гарантує, що порівняння форматів відбувається в умовах збереження еквівалентної візуальної якості, що робить результати дослідження практично цінними для розробників вебдодатків.

Таким чином, сформована теоретична та методологічна база дозволяє перейти до практичної реалізації інструментів для тестування та проведення експериментальних досліджень у наступних розділах роботи.

2. ОБҐРУНТУВАННЯ МЕТОДИКИ ДОСЛІДЖЕННЯ ВПЛИВУ ФОРМАТУ ЗОБРАЖЕННЯ НА ПРОДУКТИВНІСТЬ ВЕБ ДОДАТКІВ

2.1. Формулювання задачі та дослідницької гіпотези

Метою даного етапу роботи є розробка та обґрунтування науково-методичного підходу до експериментального дослідження продуктивності веб додатків. Основна задача полягає у визначенні того, як зміна формату графічного контенту впливає на швидкість рендерингу та мережеву ефективність у ізольованому середовищі

У роботі розгляено припущення, що сучасні формати (зокрема WebP) демонструють стабільну перевагу над класичними рішеннями (JPEG, PNG, GIF) за сукупністю метрикпродуктивності незалежно від типу графічного контенту. Проте, очікується, що ступінь цієї переваги може варіюватися залежно від геометричних параметрів зображення (орієнтація, графічні зображення), складності колірної палітри та наявності альфа-каналу (прозорості).

У цьому контексті дослідницьку задачу можна сформулювати як розробку та апробацію спеціалізованого програмного інструментарію для автоматизованого аналізу продуктивності вебдодатків при роботі з різними форматами графічного контенту. Основна мета полягає у проведенні порівняльного аналізу метрик швидкодії (зокрема LCP, FCP та Decode Timet тощо) на основі групових еквівалентів зображень, що мають ідентичний візуальний зміст та структуру, але реалізовані за допомогою різних форматів та стиснень. Дослідження передбачає використання показників Web Vitals та інших технічних показників (показники описані вище), контроль візуальної подібності за метрикою SSIM для оцінювання значущості спостережуваних відмінностей у продуктивності, що дозволить математично обґрунтувати переваги одних форматів над іншими.

Для перевірки гіпотези та забезпечення високої точності результатів, вибір тестових даних для дослідження базується на таких принципах:

1. Диференціація типів контенту. Тестова вибірка розподіляється на категорії:

- Статичні зображення: квадратні (1:1), альбомні та портретні, а також графічні об’єкти (логотипи та іконки). Це дозволить з'ясувати, чи існують «універсальні» формати, або ж певні типи стиснення краще адаптовані до конкретних орієнтацій та рівнів деталізації.
- Анімації: тестування анімованих зображень з різними рівнями прозорості для порівняння ефективності GIF та сучасних альтернатив.
- Застарілий формат SWF: через відсутність нативної підтримки сучасних браузерів, для формату SWF фіксується лише розмір файлу. Гіпотеза полягає в тому, що поприте, що формат працює ніби “еквівалентно” векторній графіці, розмір SWF не матиме критичної переваги над оптимізованою растровою графікою у сучасних обгортках.

2. Ізоляція та автоматизація (Batch Testing). Для виключення впливу стороннього коду, стилів, обєктів на сторінках (якщо інші обєкти на сторінці виявляться більшими за тестоване зображення, це суттєво вплине на показники такі як FCP) або скриптів, кожне зображення відривається на спеціалізованій «чистій» тестовій сторінці. Використання механізму пакетного тестування дозволяє автоматизувати процес проходження великої вибірки (80+ файлів), забезпечуючи однакові умови для кожного формату.
3. Ітераційність та усереднення. Кожна тестова ітерація повторюється від 3 до 5 разів (оптимально - 3). Це необхідно для обчислення медіанних значень та виключення випадкових факторів (мережевих затримок, коливань навантаження на CPU). Фіксація різниці між ітераціями дозволяє оцінити стабільність роботи браузерного рушія з конкретним форматом.
4. Контроль візуальної якості (SSIM). Перед проведенням замірів швидкодії всі зображення проходять валідацію за метрикою структурної подібності (SSIM). Встановлений поріг SSIM $\approx$ 0.95 гарантує, що порівняння

продуктивності відбувається між візуально еквівалентними об'єктами, що робить висновок про перевагу формату об’єктивним.

5. Адаптивність та в'юпорти. Дослідження передбачає перевірку рендерингу на різних в'юпортах (мобільні або десктоп). Аналізується, як формати справляються з навантаженням на вузьких мобільних в'юпортах, що є критичним для сучасної мобільної розробки.

Таким чином, розроблена методика дозволяє перетворити суб’єктивне сприйняття «швидкості» на набір верифікованих статистичних даних, що є необхідною умовою для магістерського дослідження.

2.2. Методи ізоляції середовища досліджування та мінімізації похибок від зовнішніх чинників

Для отримання точних висновків та скорочення впливу сторонніх чинників на фіксацію показників продуктивності, у дослідженні важливо забезпечити ізольоване середовище для тестування. Це дозволяє мінімізувати статистичні похибки, спричинені будь-якими іншими факторами крім формату текстових зображень.

Фактори, що можуть впливати на перебіг дослідження:

- Браузер: браузерний рушій є основним чинником, що визначає швидкість обробки зображень. Основні: Chromium, WebKit та Gecko. Для чистоти варто виконати дослідження у браузерах на різних рушіях і порівняти результати. Додатково, начистоту експерименту мінімально впливає низка супутніх факторів: версія програмного забезпечення, наявність апаратної підтримки на рівні центрального процесора (CPU), а також налаштування графічного прискорення у самому браузері. Ці фактори варто усунути або мінімізувати о початку проведення досліджень;
- Кешування: Якщо зображення завантажується з кешу, Load Time буде 0-2 мс. Рішення просте - команда cache: 'no-store', але варто також відкривати інструменти розробника та ставити галочку "Disable cache";
- Навантаження на CPU: Decode Time зростатиме при сильному навантаження на процесор. Варто виключити цей фактор;

- Пріоритетність запитів: Браузер з різних причин може вважати картинку "не терміновою" і відкласти її завантаження. Отже, тестова сторінка з однією картинкою в окремому вікні є оптимальним форматом тестування, бо там немає чому заважати;
- Швидкість інтернету: затримки провайдера є найважливішим для швидкості завантаження. Варто забезпечити локальний сервер. Найпростіший метод - тестування на localhost та Vite - інструмент збірки та локальний сервер розробки. Цей метод позбавляє всіх наступних чинників, проте про них варто згадати;
- Тестування в режимі браузера “Інкогніто”;
- Плагіни, VPN, AdBlock - сервіси. Суттєво впливають на швидкість завантаження у мережі;
- Інші менш суттєві та менш очевидні фактори, що варто виключити: Апаратне прискорення, режим Енергозбереження, Розширення екрана для інших користувачів (наприклад у Zoom), застаріла версія браузера.

2.3. Вибір показників продуктивності.

Для дослідження впливу форматів зображень на продуктивність було обрано певні метрики із Web Vitals. Оскільки метою роботи є аналіз медіаконтенту, а не логіки додатку, більшість стандартних метрик були відхилені як нерелевантні.

Обґрунтування вибору та відмови від метрик:

1. Interactivity Vitals: Ці метрики вимірюють затримку після взаємодії користувача (клік, натискання клавіш). Оскільки наше дослідження фокусується на відображенні картинок на тестовій сторінці без складних інтерфейсів, ці показники не підходять.
2. Stability Vitals (наприклад Cumulative Layout Shift - вимірює зміщення контенту). У нашому експерименті кожне зображення завантажується в ізольованому вікні з фіксованими розмірами. Візуальних зсувів не відбувається, тому CLS завжди дорівнюватиме 0
3. Loading Vitals (LCP vs Others): Зі списку Core Web Vitals обрано лише LCP, оскільки для «чистої» тестової сторінки з одним зображенням саме воно є

фінальною точкою готовності контенту. Такі метрики, як Time to Interactive (TTI), не використовуються, оскільки вони більше стосуються часу виконання важких скриптів, а не розпакування графіки.

Для проведення комплексного дослідження впливу форматів зображень на швидкість роботи веб додатків було проведено селекцію показників із загальної системи Web Vitals. Оскільки об'єктом аналізу є медіаконтент, а не програмна логіка чи складні інтерфейси, більшість стандартних метрик були відхилені як нерелевантні.

Для формування об'єктивних висновків обрано шість ключових метрик, які охоплюють повний цикл обробки графічного ресурсу - від передачі мережею до фінального рендерингу (табл. 2.3).

Таблиця 2.1 - Аналіз метрик продуктивності в контексті дослідження

Показник	Переваги для дослідження	Можливі недоліки для дослідження
LCP	Головний показник Web Vitals. Фіксує момент повної візуальної готовності контенту для користувача.	Є комплексною метрикою, тому на її значення впливає сума всіх інших чинників (мережа + CPU).
FCP	Демонструє швидкість первинного відгуку інтерфейсу та момент появи перших пікселів зображення.	Може фіксувати лише розмитий силует (у випадку Progressive JPG), що не означає готовність контенту.
PLT	Дає повну картину життєвого циклу сторінки від ініціалізації вікна до події load.	Включає мінімальні накладні витрати на рендеринг самої структури тестової сторінки.
Decode Time	Дозволяє оцінити чисту обчислювальну складність алгоритму розпакування формату силами CPU/GPU.	Чутливий до фонових процесів операційної системи під час проведення замірів.

Load Time (Resource Timing)	Ізолює мережеву ефективність формату, показуючи чистий час передачі байтів.	Навіть на localhost може мати незначні коливання через пріоритетність процесів у браузері.
File Size	Надає абсолютні дані про обсяг даних; є базою для розрахунку коефіцієнта стиснення.	Не відображає реальний досвід користувача (маленький файл може довго декодуватися).

2.4. Формування тестової вибірки зображень та методика підготовки даних

Важливою умовою репрезентативності дослідження є якість та різноманітність вхідних даних. Для забезпечення об’єктивності результатів тестова вибірка була сформована таким чином, щоб охопити максимально широкий спектр візуального контенту, який зустрічається у сучасних вебдодатках: від деталізованих фотографій до мінімалістичної векторної графіки та динамічних анімацій.

2.4.1. Статичні зображення

Для проведення експерименту було відібрано 50 унікальних статичних зображень, розділених на п'ять категорій (по 10 одиниць у кожній). Такий підхід дозволяє проаналізувати, як геометричні особливості та характер контенту впливають на ефективність алгоритмів стиснення, а також забезпечать реалістичним розподілом орієнтацій, згідно з випадками використання на реальних вбе сторінках:

- Портретні зображення (Portrait): висока деталізація облич, фокус на текстурі шкіри та плавності градієнтів фону.
- Альбомні зображення (Landscape): широка палітра кольорів, велика кількість дрібних деталей (рослинність, архітектура), що є критичним для перевірки артефактів JPEG.
- Квадратні зображення (Square): універсальний формат, що часто використовується у соціальних мережах та картках товарів.

- Графічні логотипи та іконки: ця категорія була додатково розділена на підгрупи - об’єкти з суцільним фоном та об’єкти з альфа-каналом (прозорістю). Це необхідно для виявлення «слабких місць» форматів, що не підтримують прозорість (JPEG), та оцінки ефективності PNG і WebP у збереженні чіткості країв графіки.

Початкові файли (Master Images) були завантажені з відкритих ресурсів з високою роздільною здатністю (зокрема Unsplash[1]) у форматі JPEG з мінімальним стисненням. Це забезпечило «чистоту» вихідного сигналу перед подальшою обробкою програмним засобом.

2.4.2. Анімовані зображення

Для дослідження динамічного контенту сформовано вибірку з 10 анімованих об’єктів у форматі WebP. Методологія підбору анімацій передбачала варіативність за такими ознаками:

- Кількість кадрів: вибірка містить як короткі циклічні анімації (2–3 кадри), так і складні послідовності з великою кількістю кадрів.
- Прозорість: до вибірки включено об’єкти з різним ступенем напівпрозорості. Це дозволяє продемонструвати ключову перевагу анімованого WebP над GIF та SWF - здатність зберігати м'які градієнтні переходи альфа-каналу, що є критичним для сучасного UI-дизайну.

2.4.3. Процес конвертації та параметри стиснення

Усі вихідні зображення оброблялися розробленим програмним засобом для створення варіацій у наступних форматах:

- JPEG: з рівнями якості 90% (висока якість) та 75% (стандарт для вебу);
- PNG: стиснення без втрат;
- WebP Lossy: з аналогічними параметрами якості 90% та 75% для прямого порівняння з JPEG;
- WebP Lossless: для конкурентного аналізу з PNG.

Анімовані формати конвертуються за допомогою стороннього сервісу, оскільки у браузері неможливо на пряму програмно конвертувати GIF у WebP на стороні

клієнта без сторонніх серверних бібліотек, що впливатиме на показники продуктивності (зокрема Load time).

2.4.4. Впровадження показника Compression Ratio

Для кількісної оцінки ефективності кожного формату в межах дослідження було впроваджено метрику Compression Ratio (Коефіцієнт стиснення). Вона відображає відсоткову зміну розміру файлу відносно оригіналу. Для інтерпретації результатів експерименту встановлено наступні шкали ефективності:

- Відмінно (понад −80%): радикальне зменшення ваги при збереженні візуальної цілісності (найчастіше очікується від WebP).
- Добре (−60%…−80%): висока ефективність, оптимальна для більшості вебсценаріїв.
- Помірно (−40%…−60%): стандартний результат стиснення без значної втрати деталей.
- Слабко (−20%…−40%): недостатня оптимізація, що може негативно вплинути на метрику LCP.
- Погано (0%…+50%): відсутність стиснення або збільшення розміру файлу (характерно для конвертації простих зображень у "важкі" формати без втрат).

Впровадження цієї метрики у підсумкову статистику дозволяє не просто фіксувати час завантаження, а математично обґрунтувати, за рахунок якої економії даних досягається прискорення роботи веб додатка.

Відповідно до попереднього розділу, Compression Ratio неможливо виміряти для анімованих зображень.

Висновки до розділу 2

У другому розділі було розроблено та науково обґрунтовано методику експериментального дослідження продуктивності вебдодатків при роботі з різними форматами зображень. На основі проведеної роботи можна зробити наступні висновки:

Сформульовано дослідницьку гіпотезу та принципи тестування. В основу методики покладено принцип диференціації контенту, що передбачає аналіз п'яти категорій статичних зображень та варіативних анімацій. Ключовими аспектами забезпечення точності визначено автоматизацію процесу через пакетне тестування (Batch Testing), ітераційність замірів (3–5 циклів) та обов'язковий контроль візуальної відповідності за метрикою SSIM.

Зібрані вимоги до того, як провести ізоляцію експериментального середовища. Виявлено та систематизовано зовнішні чинники, що впливають на чистоту вимірювань (кешування, навантаження на CPU, браузерні розширення). Для їх нейтралізації впроваджено використання локального сервера (Vite на localhost), роботу в режимі «Інкогніто», примусове відключення кешу та використання ізольованої тестової сторінки. Це дозволяє стверджувати, що зафіксовані коливання метрик зумовлені виключно алгоритмічними особливостями досліджуваних форматів.

Проведено селекцію та валідацію метрик продуктивності. Із загальної системи Web Vitals було відібрано шість релевантних показників (LCP, FCP, PLT, Decode Time, Load Time та File Size), які дозволяють оцінити продуктивність на всіх етапах: від мережевої передачі до апаратного розкодування. Обґрунтовано відмову від метрик інтерактивності (FID, INP) та стабільності (CLS) через їх неінформативність в умовах ізольованого тестування статичного контенту.

Сформовано репрезентативну тестову вибірку. Створено базу з 60 тестових об'єктів (50 статичних та 10 анімованих), що охоплюють різні типи орієнтації, рівні деталізації та ступені прозорості. Розроблено шкалу інтерпретації коефіцієнта стиснення (Compression Ratio), яка дозволяє математично оцінити ефективність конвертації оригіналів у формати JPEG, PNG та WebP з різними параметрами якості (75% та 90%).

Запропонована методологія створює необхідний науковий підґрунтя для переходу до практичної реалізації експерименту. Отримані в результаті такого

підходу дані забезпечать високу достовірність висновків щодо вибору оптимальних форматів зображень для сучасних вебдодатків.

3. ПРОЄКТУВАННЯ Й РОЗРОБКА ІНСТРУМЕНТАЛЬНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ ВПЛИВУ СТАНДАРТУ ES6 НА ПІДТРИМУВАНІСТЬ JAVASCRIPT-КОДУ

3.1. Формалізація задачі

3.1.1. Опис дослідницької задачі

У попередніх розділах розглянуто теоретичну методику. Для практичної частини, дослідження потребує створення спеціального програмного забезпечення «Image Performance Analyzer». ПЗ має виступати як контрольована лабораторія, що автоматизує повний цикл роботи з тестовою вибіркою зображень.

Дослідницька задача в контексті розробки програмного забезпечення включає реалізацію наступного функціоналу:

- Модуль підготовки та конвертації вхідних зображень. Програма має підтримувати механізм автоматичного створення еквівалентів зображень у різних форматах. На основі завантаженого статичного оригіналу система повинна програмно генерувати варіанти в форматах PNG, WebP (Lossy 90/75, Lossless) та JPG (90/75).Технічна реалізація цього блоку базується на використанні Canvas API або OffscreenCanvas для забезпечення «чистоти» піксельних даних;
- У випадку з анімованими зображеннями є певні обмеження. Оскільки у браузері неможливо на пряму програмно конвертувати GIF у WebP та SWF на стороні клієнта без сторонніх серверних бібліотек, задача інструменту полягає у завантаженні еквівалентів у форматах GIF, та WebP в окремі папки. Конертація попередньо відбувається за допомогою спеціалізованих бібліотек. Програма має забезпечити фіксацію метрик для всіх форматів в однакових умовах (розмір в'юпорта, відсутність кешу), щоб підтвердити або спростувати гіпотезу про перевагу сучасних форматів;
- Автоматизація вимірювального процесу (Batch Testing). Центральною задачею розробки є створення системи пакетного тестування. Програма повинна забезпечувати послідовне відкриття кожної тестової одиниці у відокремленому вікні (sandbox), тестова сторінка містить лише зображення, що тестується і

автоматично закривається. Це дозволяє ізолювати процес рендерингу зображення від основного інтерфейсу застосунку, гарантуючи, що метрики LCP та FCP будуть фіксуватися лише для зображень;

- Окрім масових статистичних досліджень, архітектура програмного забезпечення передбачає реалізацію функціоналу «поштучного» завантаження зображень користувачем. Цей модуль дозволяє проводити вибірковий експрес-аналіз зображень у реальному часі. Користувач може завантажити специфічні файли, які не входять до основної тестової вибірки (наприклад, надважкі зображення або файли з екстремальною кількістю дрібних деталей), щоб перевірити межі ефективності алгоритмів стиснення;
- Модуль програмної фіксації метрик продуктивності: Для отримання об'єктивних даних, програма повинна інтегрувати низькорівневі браузерні API, що дозволяють фіксувати показники, недоступні для звичайних методів спостереження:
    - Автоматизація збору Core Web Vitals: Використання інтерфейсу PerformanceObserver для детекції подій largest-contentful-paint (LCP) та first-contentful-paint (FCP). Це дозволяє системі точно фіксувати момент, коли зображення було фактично відрендерено на екрані користувача.
    - Ізоляція мережевих показників: Використання Resource Timing API для виокремлення чистого часу завантаження файлу (Load Time), виключаючи час на парсинг HTML-документа.
    - Вимірювання обчислювальних витрат: Програмне використання методу Image.decode(), що повертає проміс після завершення розпакування зображення в растрову сітку. Це дозволяє зафіксувати Decode Time - показник, який демонструє навантаження конкретного формату на процесор (CPU) клієнта.
- Статистичний та аналітичний модуль. Програмний засіб має не лише накопичувати сирі дані у локальному сховищі (IndexedDB), а й надавати інструменти для їх первинної обробки. Задача модуля - автоматичний розрахунок

описової числової статистики та перевірки статистичної значущості результатів безпосередньо в інтерфейсі застосунку.

3.1.2. Діаграма варіантів використання (Use Case)

Для опису функцій програми та ролей користувачів розроблено діаграму варіантів використання (Use Case Diagram), яка представлена на рисунку 3.1.

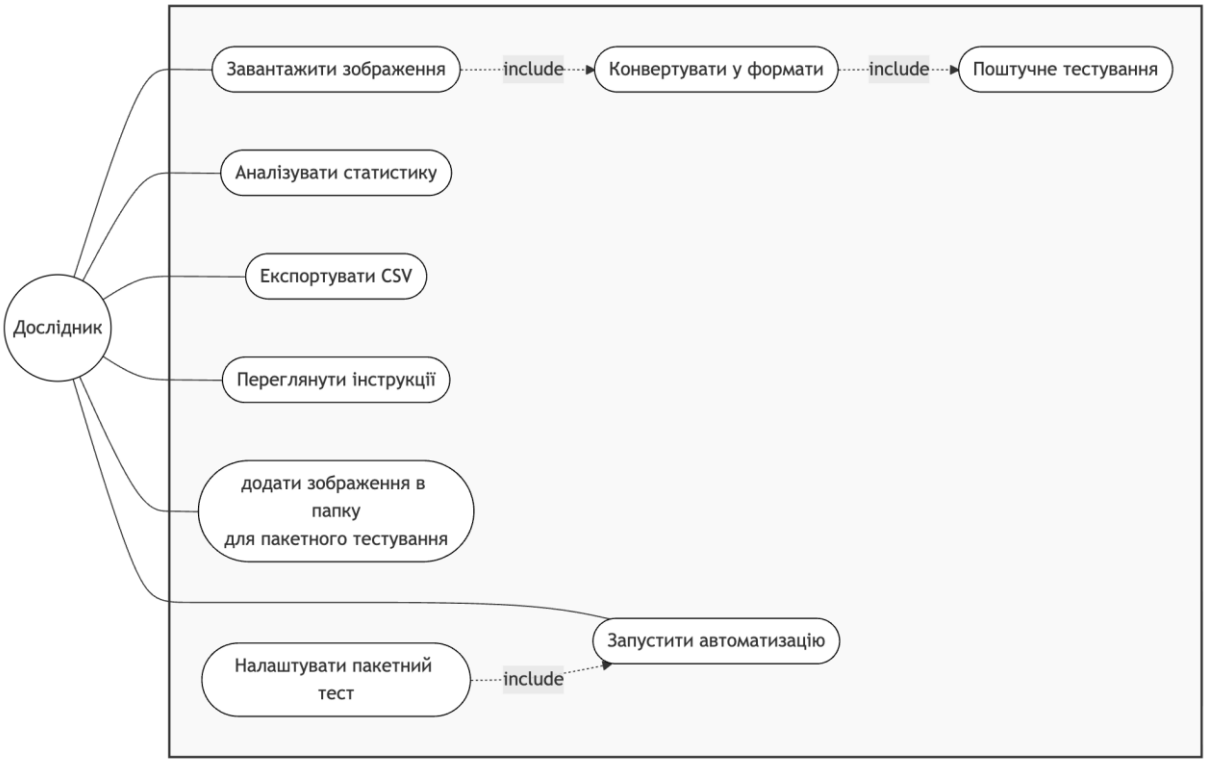


Рисунок 3.1 – Діаграма варіантів використання для програми Image Performance Analyzer (Use Case)

Діаграма варіантів використання описує функціональну повноту системи та визначає межі взаємодії між користувачем (Дослідником) і програмним комплексом.

Основними прецедентами системи є:

- Управління вхідними даними: завантаження вихідних файлів у файлову систему додатку (це вхідні JPG, GIF та WEBP animated);
- Виконання вимірювань: запуск пакетного тестування та автоматична фіксація метрик (LCP, Decode Time, Load Time) у ізольованому середовищі;

- Статистичний аналіз: обробка результатів з використанням критерію Вілкоксона та розрахунком кореляції між розміром файлу та часом завантаження.;
- Генерація звітів: візуалізація даних у вигляді графіків (Box-plots, Scatter plots);
- Еспорт зведеної таблиці результатів у формат CSV.

3.1.3. Функціональні вимоги до системи

Функціональні вимоги визначають сукупність задач, які необхідно реалізувати у програмному забезпеченні для якісного виконання дослідження.

1. Створення гнучкої системи управління вхідними даними. Для пакетного тестування - через локальну файлову систему. Важливо також проводити структурування по папкам вибірки за категоріями контенту, що дозволяє проводити диференційований аналіз для портретів, пейзажів, логотипів та анімацій. Для завантаження конкретного зображення користувачем: за допомогою прямих URL-посилань або завантаження файлу з пристрою. Програма має автоматично виконувати валідацію файлів, перевіряючи їх MIME-типи (тобто код, який визначить тип контенту та формат файлу, що передається). та обмежуючи розмір (до 30 МБ) для стабільної роботи браузерів.
2. Механізми програмної конвертації та підготовки варіантів зображень. Система має забезпечувати автоматичну генерацію еквівалентів статичних зображень у форматах PNG, JPEG (з якістю 75% та 90%) та WebP (режими Lossy та Lossless) на основі одного вихідного файлу. При цьому процес конвертації повинен гарантувати видалення надлишкових EXIF-метаданих через Canvas API, щоб порівняння обсягів даних було максимально об'єктивним. Для анімованого контенту програма повинна підтримувати паралельне завантаження та синхронне тестування оригінальних GIF, WebP та SWF із внутрішніх папок. У випадку поодинокого тестування - генерація еквівалентів в реальному часі за допомогою спеціалізованих бібліотек;
3. Алгоритми вимірювання та фіксації метрик продуктивності. Програмне забезпечення повинно реалізувати механізм повної ізоляції кожного тесту

шляхом відкриття досліджуваного об'єкта у відокремленому вікні (sandbox) без сторонніх скриптів чи стилів. Система має інтегрувати високорівневі браузерні інтерфейси (PerformanceObserver, Resource Timing API) для фіксації часових показників: Load Time, Decode Time, а також метрик LCP та FCP з точністю до мілісекунд. Крім того, обов'язковою є функція налаштування ітераційності (від 1 до 5 циклів) з подальшим автоматичним усередненням результатів для нівелювання випадкових мережевих чи обчислювальних сплесків;

4. Прикінцеві вимоги охоплюють статистичну обробку та збереження отриманих даних. Система повинна автоматично розраховувати похідні показники, такі як Compression Ratio (коефіцієнт стиснення) та практичну значущість відхилень через Т-критерій Вілкоксона. Отримані результати мають перманентно зберігатися в локальній базі даних, що запобігає втраті прогресу експерименту при перезавантаженні сторінки. Також передбачено функціонал експорту накопиченої статистики у форматі CSV для подальшого аналізу в спеціалізованих математичних пакетах, що завершує повний цикл роботи дослідницького інструментарію.

3.2. Базова архітектура системи

3.2.1. Вибір архітектурного планування

Для дослідницького програмного комплексу було обрано реалізувати веб-додаток на базі мови Javascript, стандарт ES6+ та інструментарію збірки Vite. Vite - це локальний сервер, який забезпечує миттєвий запуск пректів та автоматичне оновлення сторінок без затримок. Для дослідження корисно, що він також виступає повноцінним середовищем localhost, що є критично важливим для роботи браузерних API вимірювання та усунення похибок, пов’язаних із мережевими затримками. Це дозволяє використовувати переваги «гарячого оновлення» (HMR) та ефективно мініфікувати код (Bundling) перед проведенням фінальних замірів. Vite забезпечує правильну організацію шляхів до статичних ресурсів (тестових зображень у public/), що є критичним для коректної роботи Resource Timing API.

Основні операції винесені в окремі JS-модулі (imageConverter.js, batchStorage.js, statistics.js), які імпортуються лише тими сторінками, де вони необхідні. Наприклад, логіка конвертації працює на головній сторінці, а математичний апарат обробки p-value - лише на сторінці статистики, що забезпечує легкість налагодження та високу швидкість завантаження окремих частин системи.

Наявні сторінки: сторінка ручного тестування, на ній же розміщена таблиця результатів ручного тестування (index.html) сторінка автоматизованого пакетного тестування та налаштування параметрів тестування (batch-testing.html), аналітичний хаб для візуалізації даних (statistics.html), допоміжну сторінку з інструкціями (instructions.html).

Також, під час проведення тесту відкривається сторінка test-page.html - окрема сторінка для кожного зображення, створена для проведення замірів. Відкривається як окрема вкладка і автоматично закривається.

Головне вікно (batch-testing.html) ініціює створення дочірніх вікон, передає їм ідентифікатори об'єктів та отримує назад структуровані об'єкти з метриками. Це важливо, оскільки під час тестування відкриваються і закриваються багато вкладок, тому важливо реалізувати обмін даними через “window.postMessage” та “window.opener.” Це дає змогу автоматизувати цикл тестування 80+ зображень без ризику витоку пам'яті.

В архітектуру закладено використання об'єктного сховища на стороні клієнта, прямо всередині браузера у IndexedDB замість простішого LocalStorage. Обумовлено рішення тим, що є необхідність зберігати великі масиви результатів (метрики для кожної ітерації, назви форматів, часові штампи) навіть після повного закриття браузера, оновлення сторінки або вимкнення комп'ютера. Після проведення тестів можна пізніше повернутися до сторінки статистики для аналізу, оскільки дані фізично записані в базу даних браузера. Дані, які зібрала «сторінка-тестувальник», миттєво стають доступними для «сторінки-статистики» без необхідності передавати їх через сервер.

Таким чином, обрана багатосторінкова модульна архітектура трансформує звичайний веб додаток на розподілену вимірювальну систему. Вона гарантує стабільність при обробці великих масивів медіаданих, надійне збереження результатів та, головне, наукову достовірність отриманих метрик продуктивності.

Схема компонентної системи подана на рисунку 3.2

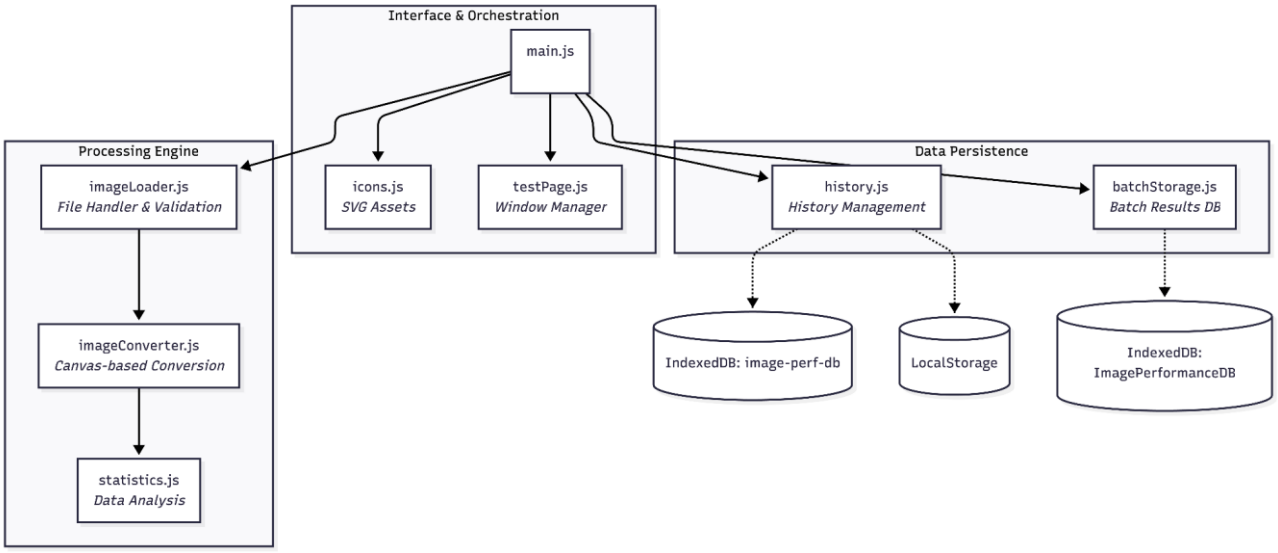


Рисунок 3.2 - Схема компонентів архітектури програми Image Performance Analyzer

Компонентна модель системи базується на архітектурі, що забезпечує чіткий розподіл функцій між інтерфейсом користувача (на базі main.js та batch-testing.js), сервісами обробки даних та ізольованим середовищем вимірювань (тобто тестова сторінка). Ключовою особливістю є використання модулів imageConverter.js для програмної генерації форматів та statistics.js для проведення наукових розрахунків (критерій Вілкоксона), тоді як прецизійна фіксація метрик винесена в автономну тестувальну сторінку (test-page.js). Усі результати агрегуються та зберігаються у клієнтській базі даних IndexedDB через модуль batchStorage.js, що гарантує цілісність даних та можливість їх подальшого експорту для дипломного дослідження.

3.2.2. Розміщення компонентів

Діаграма розміщення компонентів відображає фізичний розподіл програмних модулів у файловій структурі проєкту та їхню організацію в межах логічних контейнерів.

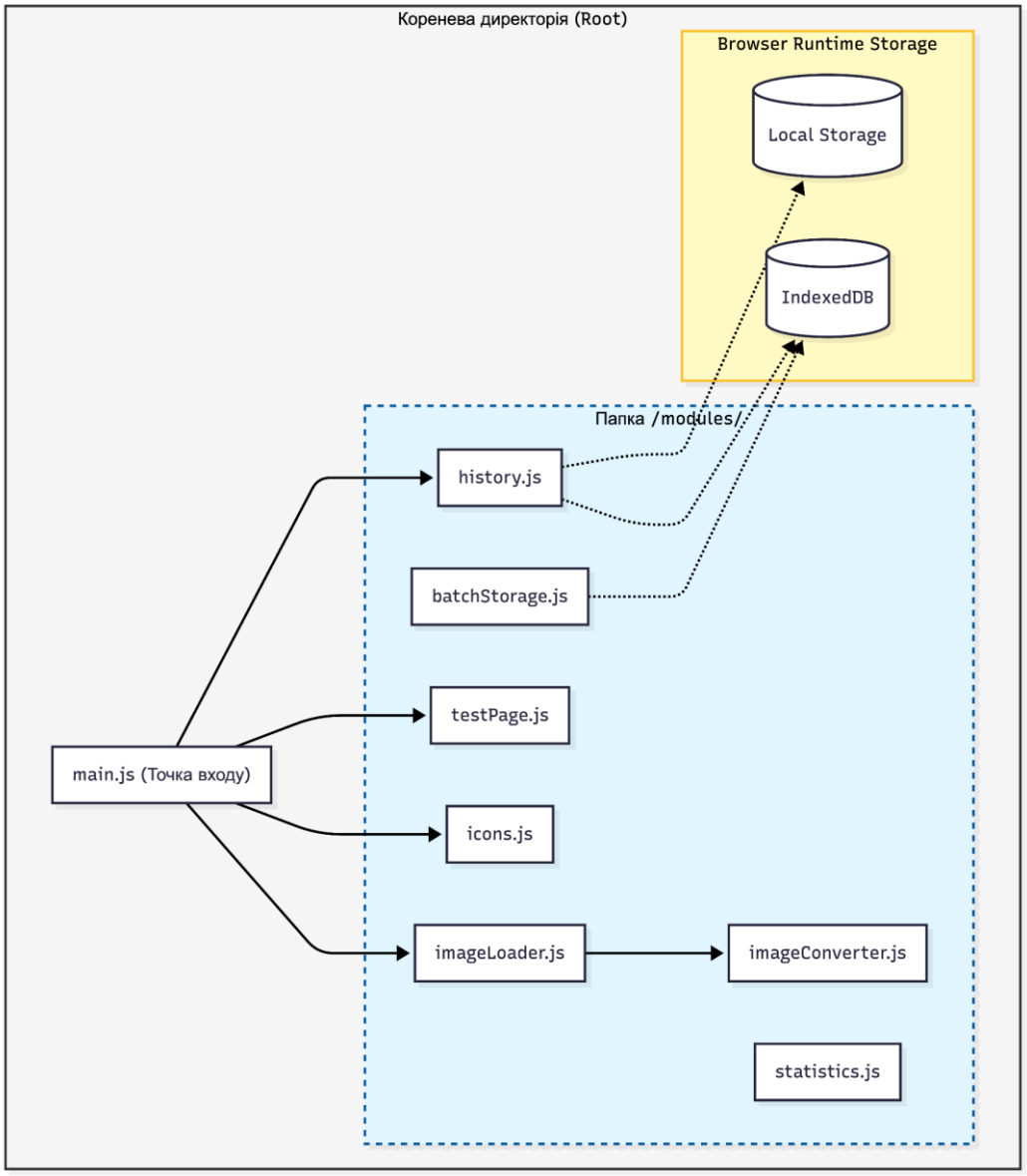


Рисунок 3.3 - Діаграма розміщення компонентів

Опис діаграми:

Структура розміщення компонентів побудована за модульним принципом, де коренева директорія містить основні точки входу в систему: index.html (головний інтерфейс), batch-testing.html (модуль автоматизації) та statistics.html (аналітичний хаб). Всі виконавчі скрипти винесені в окрему ієрархію:

- Директорія modules/ містить сервісні компоненти (imageConverter.js, imageLoader.js, statistics.js), які забезпечують обчислювальну логіку та обробку медіафайлів;
- Директорія public/ слугує сховищем для статичних ресурсів та тестових наборів зображень, забезпечуючи до них швидкий доступ через локальний сервер;
- Компонент test-page.html розміщений ізольовано для мінімізації впливу загальних стилів проєкту на процес вимірювання метрик.

Такий підхід до розміщення дозволяє розділити зони відповідальності та забезпечує високу швидкість збірки проєкту за допомогою інструменту Vite.

3.3. Внутрішнє проєктування системи

3.3.1. Вибір мови програмування

Для розробки було обрано мову JavaScript, що є найбільш логічним рішенням для створення аналітичного веб застосунку. Оскільки об’єктом дослідження є продуктивність веб додатків, використання JavaScript дозволяє безпосередньо взаємодіяти з нативними браузерними API, такими як PerformanceObserver та Resource Timing API, без програмних прошарків. Це гарантує високу точність фіксації метрик LCP, часу декодування та швидкості завантаження ресурсів у реальному часі.

Технічна реалізація базується на сучасній модульній структурі (ES6 Modules) та асинхронній моделі програмування. Використання механізмів async/await дозволяє системі виконувати ресурсомісткі операції, як-от програмна конвертація зображень через Canvas API та пакетна обробка великих масивів даних, не блокуючи головний потік браузера. Завдяки інтеграції з локальним сервером Vite та клієнтською базою даних IndexedDB, комплекс забезпечує стабільну роботу з великими обсягами медіаданих (80+ об'єктів) та надійне збереження результатів експерименту для їх подальшої статистичної обробки.

3.3.2. Стек технологій

Таблиця 3.1 - Стек технологій, застосованих при розробці програми

Технологія / Бібліотека	Версія	Призначення в додатку
JavaScript (ES6+)	ECMA Script 2022	Основна мова програмування для реалізації бізнес-логіки та обробки даних.
Vite	5.x	Середовище розробки та локальний HTTP-сервер для усунення мережевих похибок.
Chart.js	4.4.0	Візуалізація результатів у інфографіку: побудова Box-plots, Scatter-plots
Bootstrap	5.3.0	Фреймворк для побудови адаптивного інтерфейсу та компонентів (таблиці, форми).
IndexedDB API	-	Клієнтська база даних для збереження масивів результатів пакетного тестування.
Resourse Time API	-	Вимірювання метрик
Canvas API	-	Програмна конвертація зображень та маніпуляція бінарними даними на стороні клієнта.
PerformanceObserver	-	Вимірювання метрик Core Web Vitals (LCP, FCP) та часу декодування.

Bootstrap Icons	1.11.0	Графічне відображення статусів тестування та елементів управління.

Для аналізу показників Web Vitals та інших технічних показників, що описані у попередніх розділах, застосовані Canvas API, PerformanceObserver та IndexedDB API, які у коді налаштовані наступним чином:

Canvas API у файлі imageConverter.js

```
   * @param {Blob} sourceBlob
   *    @param    {'png'|'jpg'|'webp'|'webp-lossy'|'webp-lossless'}
targetFormat
   * @param {number} [quality=0.85]
   *                                                          @returns
{Promise<{blob:Blob,width:number,height:number,format:string,quali
ty?:number}>}
   */
   export   const   convertStaticImage   =   async   (sourceBlob,
targetFormat, quality = 0.85) => {
    const    {    bitmap,    width,    height    }    =    await
loadImageBitmap(sourceBlob);
    const canvas = buildCanvas(width, height);
    const ctx = canvas.getContext('2d');
    ctx.drawImage(bitmap, 0, 0, width, height);
    const mime = MIME_MAP[targetFormat] ?? 'image/png';
    const isLosslessWebp = targetFormat === 'webp-lossless';
    const blob = await new Promise((resolve, reject) => {
      const q = mime === 'image/jpeg' ? quality : isLosslessWebp ?
1 : quality;
      const     encoderOptions     =     Number.isFinite(q)     ?
Math.min(Math.max(q, 0.05), 1) : undefined;
      canvas.convertToBlob
        ? canvas
            .convertToBlob({ type: mime, quality: encoderOptions })
            .then(resolve)
            .catch(reject)
```

```
        : canvas.toBlob(
            (result) => {
              if (!result) {
                reject(new      Error('Неможливо      конвертувати
зображення.'));
                return;
              }
              resolve(result);
            },
            mime,
            encoderOptions
          );
    });
    if (bitmap.close) bitmap.close();
    return { blob, width, height, format: targetFormat, quality: mime
=== 'image/jpeg' ? quality : undefined };
   };
```

PerformanceObserver

```
   const observeLCP = () => {
    if (typeof PerformanceObserver === 'undefined') return () => {};
    try {
      const observer = new PerformanceObserver((list) => {
        const last = list.getEntries().at(-1);
        if (!last) return;
        lcpState.value  =  last.renderTime  ||  last.loadTime  ||
last.startTime;
      });
      observer.observe({     type:     'largest-contentful-paint',
buffered: true });
      return () => observer.disconnect();
    } catch (error) {
      console.warn('LCP недоступний', error);
      return () => {};
    }
   };
```

```
   const measureVariant = async (variant, originalSize) => {
    const source = variant.dataUrl || variant.url;
    if (!source) throw new Error('Джерело зображення відсутнє');
    const img = new Image();
    const start = performance.now();
    const loadPromise = new Promise((resolve, reject) => {
      img.onload = resolve;
      img.onerror = reject;
    });
    img.decoding = 'async';
    img.src = source;
    await loadPromise;
    const decodeStart = performance.now();
    await img.decode().catch(() => {});
    const decodeTime = performance.now() - decodeStart;
    const                          resource                          =
performance.getEntriesByName(img.currentSrc).pop();
    const  loadTime  =  resource?.duration  ??  performance.now()  -
start;
    const  compressionRatio  =  originalSize  ?  (variant.size  ??
originalSize) / originalSize : null;
    return {
      id: variant.id,
      format: variant.format,
      loadTime,
      decodeTime,
      lcp: lcpState.value,
      size: variant.size,
      compressionRatio,
      width: variant.width,
      height: variant.height,
    };
   };
```

IndexedDBв модулі batchStorage.js

```
export const openBatchDB = () =>
 new Promise((resolve, reject) => {
   const request = indexedDB.open('ImagePerformanceDB', 2);
   request.onupgradeneeded = (event) => {
     const db = event.target.result;
     if (!db.objectStoreNames.contains('batchResults')) {
       const store = db.createObjectStore('batchResults', { keyPath: 'id' });
       store.createIndex('timestamp', 'timestamp', { unique: false });
     }
   };
   request.onsuccess = () => resolve(request.result);
   request.onerror = () => reject(request.error);
 });
@param {{config:object,results:Array,summary:object,status?:string}} batchData
* @returns {Promise<string>} ID збереженого тесту
*/
export const saveBatchResults = async (batchData) => {
 const db = await openBatchDB();
 const id = batchData.id || `batch_${Date.now()}`;
 const payload = {
   id,
   timestamp: new Date().toISOString(),
   config: batchData.config,
   results: batchData.results,
   summary: batchData.summary,
   status: batchData.status || 'completed',
 };
 return new Promise((resolve, reject) => {
   const tx = db.transaction('batchResults', 'readwrite');
   const store = tx.objectStore('batchResults');
```

```
      const request = store.put(payload);
      request.onsuccess = () => resolve(id);
      request.onerror = () => reject(request.error); });};
```

Відкриття тестових сторінок:

```
   saveButton.addEventListener('click', () => {
      window.opener?.postMessage({ type: 'TEST_COMPLETE', entryId,
variants: metrics }, window.location.origin);
      if (storageKey) window.localStorage.removeItem(storageKey);
      disconnectLcp();
      window.close(); });
```

3.3.3. Ієрархія архітектури та комунікація між модулями

Модуль imageLoader.js є вхідною точкою системи, що відповідає за отримання та первинну обробку графічних ресурсів. Він реалізує асинхронні методи зчитування файлів через File API або завантаження за зовнішніми посиланнями за допомогою Fetch API. Основним завданням модуля є валідація вхідних даних, перевірка MIME-типів та перетворення бінарних даних у об’єкти типу Blob або ImageBitmap для подальшої передачі в систему конвертації.

Програмна трансформація форматів зосереджена в модулі imageConverter.js. Використовуючи Canvas API, цей компонент створює тимчасові графічні одиниці , де рендеряться оригінальні зображення та їх наступні стиснення у потрібні формати. Модуль динамічно керує параметрами якості (quality) та без втрат генерує ідентичні за змістом, але різні за структурою файли, що є основою для подальшого порівняльного аналізу.

Модуль batchStorage.js поєднує логіку програми та IndexedDB. Він спрощує роботу з базою даних, беручи на себе складні технічні процеси, і надає іншим модулям легкий простий для запису, перегляду чи видалення результатів тестів.

Процес вимірювання показників продуктивності керується модулем testPage.js та скриптом test-page.js. Перший відповідає за підготовку параметрів тесту та виклик тетсового вікна. Другий, працюючи безпосередньо у середовищі на тестовій сторінці, використовує механізм PerformanceObserver для фіксації

метрик. Після завершення тестування, модуль передає об’єкт з результатами назад до основного вікна.

Математичний модуль statistics.js, який проводить обробку отриманих показників. Він розраховує медіанні значення, стандартні відхилення, перевіряє статистичні гіпотези за критерієм Вілкоксона. Окрім цього, структурує дані для подальшої візуалізації у вигляді графіків та експорту файлом CSV.

Допоміжний модуль utils.js містить набір уніфікованих функцій, що використовуються в усіх частинах системи, форматує числових значення (переведення байтів у мегабайти, мікросекунд у мілісекунди тощо), генерує унікальні ідентифікатори для тестових запусків. Це забезпечує легке читання даних людиною, зрозуміле відображення даних у інтерфейсі.

Взаємодія відбувається наступним чином:

Вищий рівень - Модулі main.js та batch-testing.js: Виступають в ролі центральних контролерів - ініціюють запити до модулів, керують життєвим циклом тестових вікон та координують процес обміну повідомленнями через PostMessage API.

Середній рівень - тут зосереджена основна логіка.

- Модуль imageConverter.js отримує дані від завантажувача та повертає масив згенерованих форматі;
- Модуль testPage.js готує параметри для вимірювального середовища;
- statistics.js виконує роль чистої обчислювальної функції, отримуючи сирі метрики та повертаючи статистичні показники (p-value та медіани).

Низький рівень - незалежні модулі утиліти та сховище. Вони не мають зовнішніх залежностей і надають стандартизовані методи для збереження даних у IndexedDB та форматування значень.

Окреме місце в ієрархії займає зв'язок між batch-testing.js та test-page.js. Оскільки вони працюють у різних вікнах браузера, їх взаємодія реалізована через протокол подій:

1. Request: Передача метаданих через URLSearchParams та localStorage;
2. Execution: Автономне вимірювання показників Web Vitals;

3. Response: Повернення результатів через window.opener.postMessage, що забезпечує асинхронну та безпечну передачу даних назад до основної бази.

Взаємодія продемонстрована на діаграмі модулів рисунку 3.3.

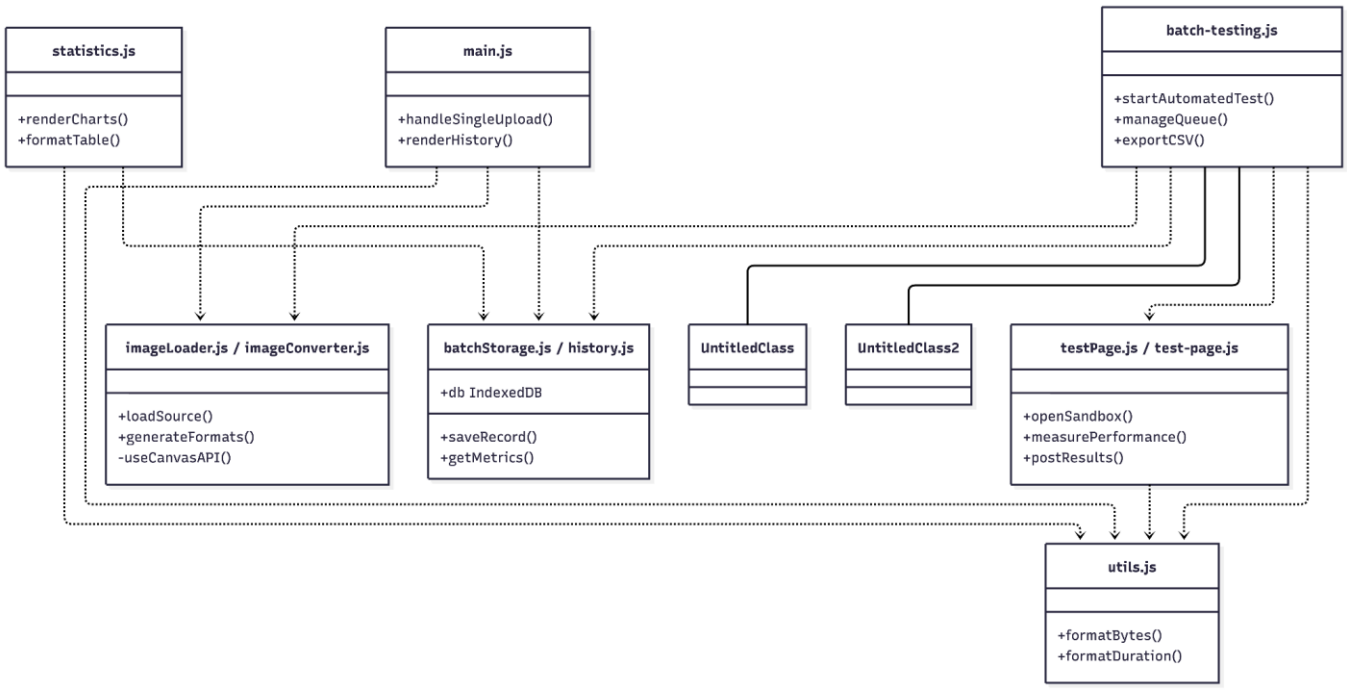


Рисунок 3.4 - Діаграма модулів програми Image Performance Analyzer

3.3.4. Алгоритми обчислення показників

Основні метрики дослідження (LCP, FCP, PLT, Load Time) фіксуються за допомогою вбудованих браузерних інструментів PerformanceObserver та Resource Timing API. Програмний комплекс не використовує сторонні алгоритми для вимірювання цих значень, а звертається безпосередньо до нативного методу img.decode() та системних метрик браузера. Тому буде беззмістовно описувати їх алгоритми у розділі.

У програмі виконані наступні алгоритми:

Час декодування: для його опису важливо підкреслити використання асинхронного підходу, який дозволяє отримати чисті дані без впливу на головний потік браузера.

Алгоритм базується на використанні нативного методу img.decode(), який повертає Promise, що виконується лише після повної розпаковки стиснених даних зображення у піксельну сітку в пам'яті GPU або CPU. На першому етапі система ініціалізує об’єкт зображення та встановлює атрибут decoding="async",

щоб винести процес обробки з основного потоку. Безпосередньо перед викликом методу декодування програма фіксує початкову мітку часу за допомогою високоточного таймера performance.now(). мСам по собі метод - нативний, а фіксація часу - програмна. Після успішного завершення операції декодування (спрацювання resolve промісу) фіксується кінцева мітка часу. Різниця між цими значеннями і є чистим показником decodeTime, що відображає обчислювальну складність формату. Паралельно алгоритм перевіряє наявність оригінального розміру файлу для розрахунку коефіцієнта стиснення, після чого всі отримані метрики агрегуються в єдиний об’єкт для збереження.

Блоксхема представленя на рисунку 3.4.

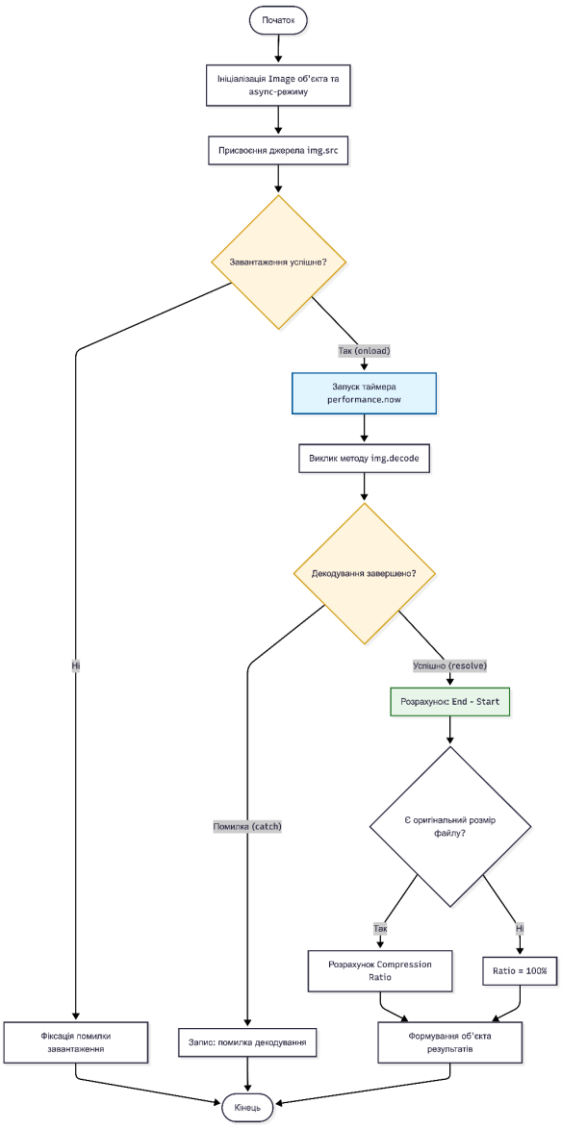


Рисунок 3.5 - Алгоритм вимірювання часу декодування

Статистичний алгоритм (Тест Вілкоксона) - найскладніша частина математичного ядра (modules/statistics.js).

- Алгоритм сортує різниці між показниками (наприклад, LCP для JPG та LCP для WebP) за їх абсолютним значенням. Припустимо, ви тестуєте час декодування для трьох зображень у форматах Група А (JPG) та Група Б (WebP). Обчислення різниці d = xA - xB дозволяє системі перейти від порівняння абсолютних значень до аналізу відносних змін продуктивності всередині кожної пари об'єктів. Це нівелює вплив роздільної здатності зображень: навіть якщо велике фото декодується довго, ми аналізуємо лише те, наскільки один формат виявився ефективнішим за інший для цього конкретного файлу;
- Далі обчислюється, чи є різниця позитивною чи негативною. Блоксхема алгоритму обчислення представлено на рисунку 3.5;
- Обчислення W-критерію: підсумовування рангів для визначення статистичної значущості. Цей алгоритм дозволяє системі зробити висновок: "Різниця між форматами є не випадковою, а закономірною". Приклад застосування до вигаданих тестових даних критерія Вілкоксона наведено у таблиці 3.2.

Таблиця 3.2 - приклад застосування Тесту Вілкоксона

Зображення	A (JPG)	B (WebP)	Різниця $d=xA-xB$	Значення
Фото 1	15.0 мс	10.0 мс	+5.0	WebP швидший на 5 мс
Фото 2	8.0 мс	12.0 мс	-4.0	JPG швидший на 4 мс

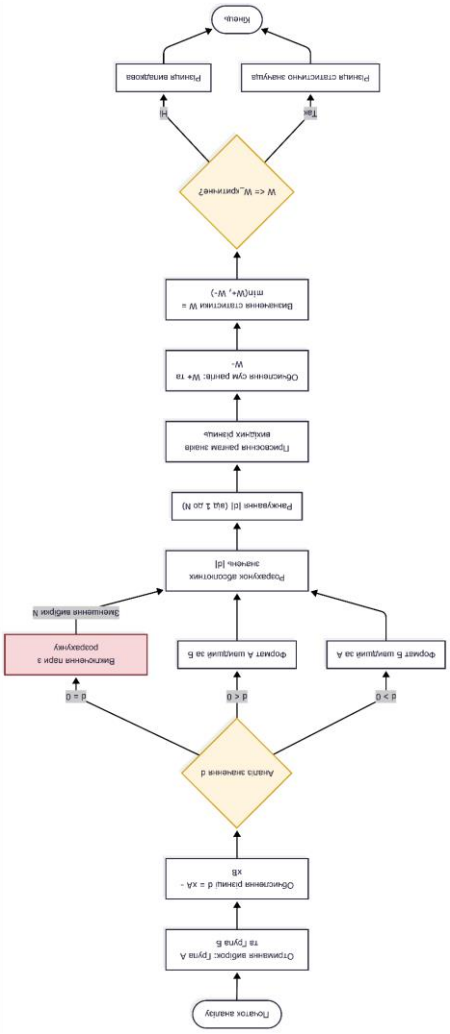


Рисунок 3.6 - Алгоритм обчислення критерія Вілкоксона

3.4. Інтерфейс користувача

3.4.1. Опис екранів

Наявні чотири основні сторінки:

Головна - короткий опис програми та поле для введення файлу для ручного тестування. Наявна широка область для“Drag and drop”, введеня url-адреси зображення. Після завантаження файлу для ручного тестування, відображено блок із самим зображенням, розміром файлу та прогресом проведення тесту. Зберігається таблиця результатів тестування. Перша колонка таблиці - зменшена копія зображення, для зручного орієнтування в тестах.

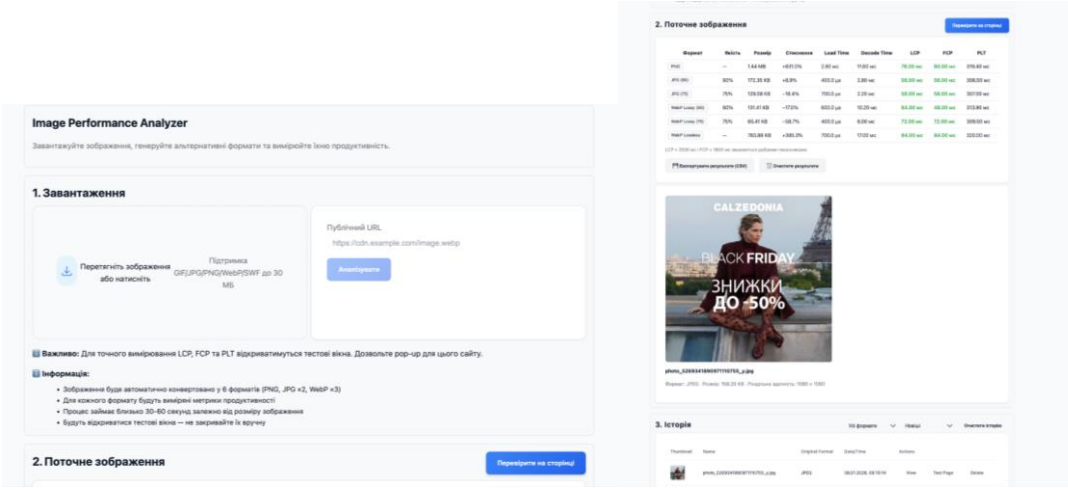


Рисунок 3.7 - головна сторінка додатку

Batch Testing. Сторінка містить параметри налаштування поточного тестування. Після натиснення кнопки “ Почати тестування” зявляється прогрес бар, який показує поточну кількість тестів, що вже виконано і скільки тестів ще залишилось

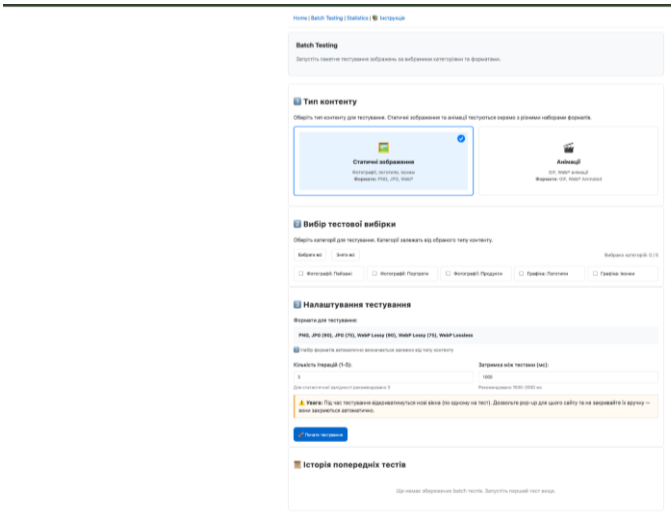


Рисунок 3.8 - Batch Testing сторінка, вибір параметрів тестування

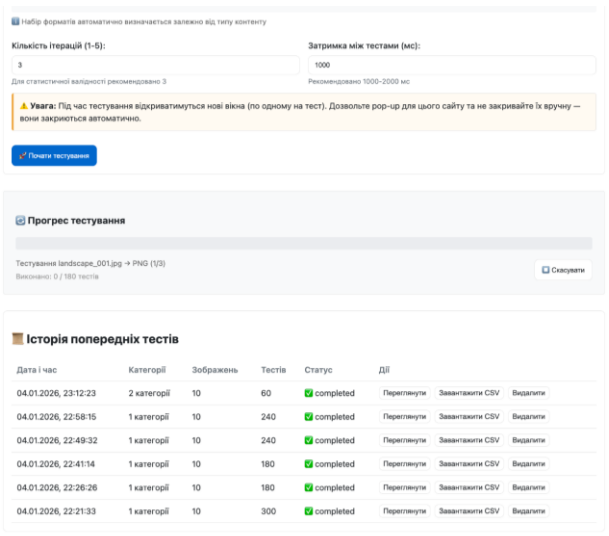


Рисунок 3.9 - Batch Testing сторінка, прогрес завантаження тестів

Сторінка статистики містить таблиці, де зафіксовано показники та інтерактивні графіки. Блок візуалізації, побудований на базі бібліотеки Chart.js. Екран автоматично будує гістограми порівняння швидкості декодування та діаграми розсіювання для виявлення кореляції між розміром файлу та часом завантаження.

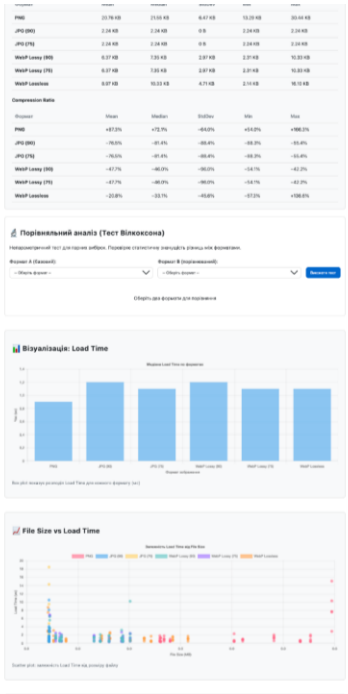


Рисунок 3.10 - Сторінка статистики

Також наявна сторінка інструкцій для користувача, на якій детально описано правила роботи із програмою та поради щодо проведення тестування

Сторінка тестування зображення постійно закривається і відкривається, тому перед використанням користувачу варто довволити pop-up для цього сайту.

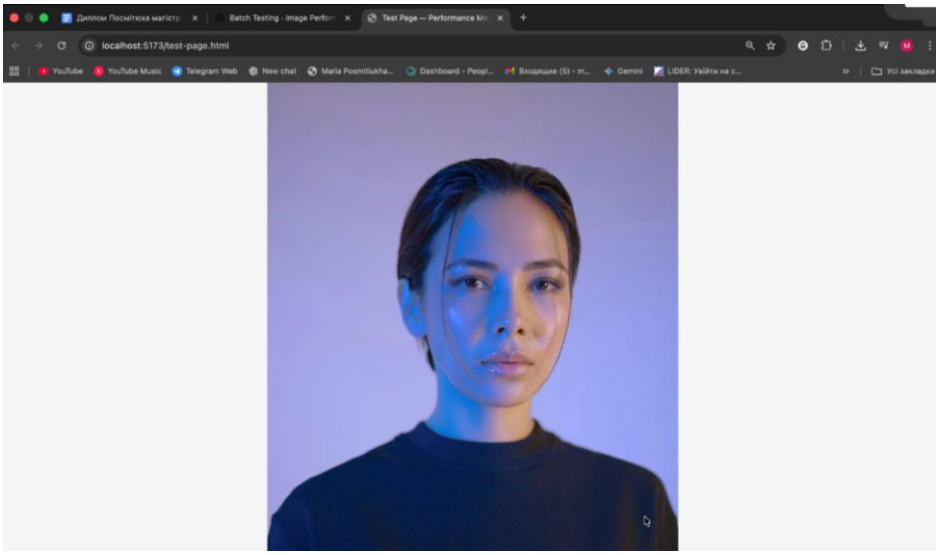


Рисунок 3.11 - тестова сторінка

3.4.2. Реалізація інтерфейсу

Механізм Drag-and-Drop: реалізований за допомогою стандартних подій JavaScript (dragover, dragleave, drop). Для покращення UX додана динамічна зміна стилів контейнера при наведенні файлу. Обробка даних відбувається через об'єкт DataTransfer, звідки файли передаються безпосередньо в ImageLoader без перезавантаження сторінки.

Інтерактивні діаграми (Charts): побудовані на базі бібліотеки Chart.js. Використовуються два типи візуалізації: стовпчикові діаграми (Bar Charts) та хітмапи. Наявні наступні графіки: Візуалізація Load Time, File Size vs Load Time,: Порівняння форматів за всіма показникам. Дані на графіках оновлюються автоматично після кожного завершеного тесту завдяки зв'язку з базою IndexedDB.

Індикатор прогресу (Progress Bar): реалізований як компонент, що відстежує стан черги в модулі batch-testing.js. Прогрес розраховується як відношення кількості завершених тестів до загальної кількості запланованих варіантів у пакеті.

3.5. Тестування

3.5.1. Методи тестування

Тестування «білої скриньки» (White-box testing): Перевірка внутрішньої логіки математичних та конвертаційних модулів. Це дозволило переконатися, що алгоритми розрахунку медіани, тесту Вілкоксона та коефіцієнтів стиснення працюють коректно навіть при наявності аномальних значень (викидів) у вибірці. Зокрема, було протестовано покриття всіх гілок умовних операторів у модулі statistics.js, що відповідають за обробку нульових різниць між показниками форматів.

Тестування «чорної скриньки» (Black-box testing): Перевірка функціональності системи з точки зору кінцевого користувача. На вхід ручного тестування подавалися зображення різних форматів та розмірів (від мікро-іконок до 4K-фото), а також формати, які не підтримуються програмою. Перевірялася відповідність вихідних даних: чи збігається кількість згенерованих варіантів із кількістю завантажених файлів та чи коректно відображаються графіки продуктивності.

3.5.2. Міжбраузерне тестування

Для перевірки на різних рушіях (Chromium, Gecko, WebKit) використано зміну налаштувань браузера за замовчуванням безпосередньо у системних налаштуваннях компютера. Це дозволило послідовно запускати програму спочатку в Safari, а потім у Chrome, щоб переконатися в ідентичності результатів вимірювань у різних рушіях.

Висновки до розділу 3

1. Обґрунтовано вибір архітектури програмного комплексу, яка базується на модульній структурі JavaScript. Це дозволило відокремити логіку прецизійного вимірювання від аналітичного модуля та інтерфейсу, забезпечивши високу швидкість обробки даних та масштабованість системи;
2. Розроблено та реалізовано алгоритм точного вимірювання часу декодування зображень. Використання нативного методу img.decode() у

поєднанні з високоточним таймером performance.now() дозволило фіксувати часові витрати на рівні графічного рушія браузера з мікросекундною точністю;

3. Інтегровано модуль статистичного аналізу на основі непараметричного критерію знакових рангів Вілкоксона. Це забезпечує можливість наукової інтерпретації результатів та дозволяє встановити статистичну значущість переваг одного формату зображень над іншим навіть при наявності системних шумів;
4. Спроєктовано та впроваджено користувацький інтерфейс з використанням технологій Chart.js та IndexedDB. Це забезпечило зручну візуалізацію метрик у реальному часі та надійне збереження результатів експериментів для подальшого опрацювання;
5. Проведено тестування та налагодження розробленого інструментарію за допомогою методів «білої» та «чорної» скриньок. Завдяки використанню середовища Live Server та зміні браузеру за замовчуванням, для запуску на різних рушіях підтверджено стабільність і відтворюваність результатів вимірювань у різних браузерних рушіях (Chromium, Gecko, WebKit).

4. ДОСЛІДЖЕННЯ ВПЛИВУ ФОРМАТУ ЗОБРАЖЕНЬ НА ПРОДУКТИВНІСТЬ ВЕБ-ДОДАТКІВ

4.1. Підготовка до проведення експерименту

Для забезпечення чистоти результатів та валідності порівняльного аналізу було сформовано репрезентативний набір тестових даних. Експериментальний масив зображень розділено на три основні категорії:

- Набір статичних зображень (JPEG): контрольна група, що складається з фотографій високої роздільної здатності.
- Набір анімованих зображень (WebP);
- Набір анімованих зображень (GIF).

Для анімованих файлів було використано ідентичний вхідний матеріал, конвертовані у формати GIF та WebP за допомогою сторонніх сервісів обробки. Варто зауважити, що пряме візуальне порівняння схожості анімованих картинок неможливо, адже є принципова різниця в алгоритмах покадрового стиснення. До того ж очевидно, що GIF суттєво програє у якості передачі кольорів через 8-бітну колірну схему, що призводить до появи зернистості та втрати градієнтів порівняно з WebP високої якості. Такі відмінності не діють у випадку простих анімацій (піксельні анімовані картинки бе деталей та великої кількості артефактів).

Кожна папка з даними містить файл метаданих (JSON), який виконує роль реєстру для програми. У файлі зафіксовано категорію, описи зображень, їхні ідентифікатори та вихідні формати. Це дозволяє автоматизувати процес пакетного тестування та забезпечити зв'язність результатів з конкретними характеристиками файлів.

Для мінімізації похибок вимірювання часу декодування та рендерингу перед кожною сесією тестування проводилася підготовка обчислювального середовища:

- Закриття всіх сторонніх програм та процесів для вивільнення ресурсів CPU/GPU;

- Використання режиму інкогніто в браузері для ігнорування впливу кешу та розширень;
- Зміна браузера за замовчуванням через системні налаштування для послідовного тестування на рушіях Chromium (Chrome), Gecko (Firefox) та WebKit (Safari);

4.2. Опис програмно-апаратного середовища

Експерименти проводилися на базі наступного обладнання та ПЗ:

Апаратна частина: Процесор Apple M-серії (або Intel/AMD), обсяг оперативної пам'яті (8/16 ГБ), інтегрована графічна система.

Програмна частина: ОС macOS, середовище виконання Node.js v.20+, браузер Google Chrome (версія 120+), Firefox, Safari.

Інструментарій розробки: Visual Studio Code, розширення Live Server для розгортання локального сервера (localhost:5500).

4.3. Методика проведення експерименту

4.3.1. Ручне тестування

Варто зазначити, що функція ручного тестування в програмі додана з міркувань зручності користувача та швидкої перевірки мінімальних гіпотез (наприклад гіпотез про те, що розмір файлу значно змінюється при конвертації у інші формати). Режим має свої переваги для проведення дослідження, але в основному аналіз та порівняння результатів дослідження з очікуваннями проводиться на аналізі пакетного тестування.

4.3.2. Процедура пакетного тестування та хід експерименту

Етап 1. Налаштування проведення досліду

Перед початком активної фази вимірювань здійснювалася конфігурація середовища тестування:

- Формування тестової вибірки: Вибір категорій зображень (статичні фото, WebP-анімації, GIF-анімації) відповідно до мети поточного тесту. У попередніх розділах детально розглянуто процеси формування тестової вибірки зображень. Напоточному етапі необхідно вручну обрати які саме зображення з уже готової вибірки тестуватимуться;

- Параметризація ітерацій: Встановлення кількості повторних запусків для кожного об’єкта (від 1 до 5). Це дозволяє нівелювати вплив випадкових системних шумів та фонових процесів браузера на результати вимірювань. Багаторазове повторення замірів забезпечує отримання стабільного медіанного значення, що гарантує відтворюваність експерименту та точність порівняльного аналізу форматів;
- Конфігурація затримок: Встановлення інтервалу між тестами (мс) для запобігання перегріву браузерного рушія та забезпечення коректного звільнення оперативної пам'яті між операціями декодування. Дефортне значення є оптимальним
- Дозволи браузера: Дозвіл на відкриття спливаючих вікон (pop-up) для забезпечення безперешкодної роботи модуля тестової сторінки, у якому відбуваються заміри.

Етап 2. Перебіг експерименту та динаміка вікон.

Після ініціалізації процесу аналізу система переходить у фазу активного тестування, яка характеризується циклічною зміною робочих вікон:

1. Створення ізольованого контексту: Для кожної ітерації програма автоматично відкриває та закриває окремі модальні вікна, що може зайняти певний час, що залежить від налаштувань тестування. Це необхідно для того, щоб графічний рушій браузера кожного разу ініціалізував процес декодування з нуля, уникаючи використання внутрішнього кешу рендерингу;
2. Візуальний контроль: Під час тестування користувач спостерігає за швидкою зміною вікон, де послідовно з’являються та зникають тестові зразки. Така "пульсуюча" робота інтерфейсу свідчить про активне навантаження на методи img.decode() та PerformanceObserver;
3. Автоматичне закриття: Після фіксації метрик вікно миттєво закривається, повертаючи управління головному модулю для запису даних у базу IndexedDB та підготовки наступного файлу.

Етап 3. Перегляд статистики для попереднього аналізу результатів.

4.3.3. Обгрунтування використання інфографіки як інструменту аналізу

Блок “Описова статистика” Відображає розмах значень, медіану та "викиди" для всієї вибірки. Допомагає оцінити передбачуваність формату. Вузький діапазон свідчить про те, що формат працює стабільно на різних типах зображень, тоді як великий розкид вказує на залежність продуктивності від контенту.

Для кожного показника (LCP, FCP, PLT, Load Time, Decode Time, File Size, Compression Ratio) побудована таблиця описової статистики. Кожен із розрахованих параметрів дозволяє оцінити властивості досліджуваних форматів з різних точок зору:

- Середнє арифметичне: відображає середній результат за всіма ітераціями тестування. Цей показник є базовим індикатором продуктивності, проте він може бути чутливим до поодиноких системних затримок;
- Медіана: значення, що ділить вибірку навпіл. На відміну від середнього, медіана є стійкою до статистичних «викидів» (раптових стрибків завантаження процесора), тому вона вважається найбільш об’єктивним показником типового часу декодування зображення;
- Стандартне відхилення (Standard Deviation): характеризує міру розсіювання значень навколо середнього. Низьке відхилення свідчить про стабільність роботи формату, тоді як високе вказує на суттєву залежність продуктивності від контенту або розміру конкретного файлу. Наприклад У PNG показник Standard Deviation для часу завантаження в одному з тестів склав 17.20 мс при медіані 22.60 мс. Це свідчить про дуже високу нестабільність: час завантаження PNG сильно «скаче» залежно від контенту, в той час як для WebP цей показник лише 1.37 мс, що всідчить про більше стабільність формату;
- Мінімум та Максимум (Min/Max): визначають межі, у яких коливаються показники. Це дозволяє зафіксувати «найгірший сценарій» (максимальний час очікування користувача) та «найкращий сценарій».

Також наявна система інтерактивних графіків, побудованих за допомогою бібліотеки **Chart.js**. Кожен графік виконує конкретну функцію:

Гістограма медіанного часу завантаження (Bar Chart) - Порівняння медіанного показника Load Time (мс) для кожного формату зображення. Це найшвидший спосіб візуально порівняти "швидкість" форматів. Він показує загальну тенденцію: який формат забезпечує найшвидший відгук сторінки. Наприклад, на графікувидно, що у той час як медіана PNG становить 22.60 мс, формат WebP Lossy (75) демонструє результат у 3.00 мс, що візуально підтверджує прискорення завантаження у ~7.5 раза. При

Діаграма розсіювання для порівняння File Size та Load Time - візуалізує кожну окрему ітерацію тесту як точку в системі координат «Розмір (МБ) - Час (мс)». Інформативно для аналізу кореляції між вагою файлу та швидкістю його обробки, а також виявлення статистичних аномалій.

Нормалізована теплова карта (Heatmap)

Зведене порівняння всіх метрик (Load Time, Decode Time, LCP, FCP, PLT, File Size) у єдиній шкалі від 0 до 100 балів. Призначений для комплексної оцінки формату за сукупністю всіх технічних параметрів (100 = найкращий результат у групі).

Таблиця результату тесту Вілкоксона для двох обраних форматів. Це детальна таблиця, що містить результати непараметричного тесту для парних вибірок, що слугує як математичне підтвердження того, чи є зафіксована різниця закономірною. Тест працює задовільно і правильно лише за умови тестування великої кількості зображень за дослід. Іншими словами, при тестуванні однієї категорії з 10 зображень, результат завжди буде незадовільним.

4.4. Результати експерименту

У цьому розділі результат буде описуватися на конкретному прикладі тестових даних. У ході тестування використано статичні зображення категорії “Пейзажні” та “Іконки”, тобто 20 зображень, 3 ітерації та показником затримки між тестами 1000 мс, ідентичне дослідження проведено в двох браузерах: Chrome та Safari. У підсумку - по 360 тестових вікон у двох браузерах. Загалом, цього

достатньо для короткого огляду результатів дослідження, решта таблиць із результатами буде наведена у додатку. Демонстрація загальної статистики (тобто всіх форматів одразу) завадить можливій демонстрації переваг формату у роботі з певним видом контенту.

4.4.1. Аналіз показників для статичних зображень

Аналіз Пейзажних зображень. 10 зображень по 3 ітерації, 6 форматів, отже 180 тестів. Результати у таблиці 4.1

Таблиця 4.1 – Описова статистика показників продуктивністі для пейзажних зображень формату у браузері Chrome

Load time					
Формат	Mean	Median	StdDev	Min	Max
PNG	33.39 мс	31.10 мс	12.02 мс	18.00 мс	67.00 мс
JPG (90)	7.21 мс	6.00 мс	3.96 мс	2.10 мс	17.20 мс
JPG (75)	6.92 мс	5.40 мс	3.32 мс	1.40 мс	12.90 мс
WebP (90)	5.40 мс	5.40 мс	1.71 мс	2.40 мс	9.50 мс
WebP (75)	4.31 мс	4.00 мс	1.82 мс	1.40 мс	8.10 мс
WebP Lossless	22.50 мс	24.40 мс	7.45 мс	5.30 мс	33.10 мс

Decode Time (мс)					
PNG	213.09 мс	194.20 мс	56.77 мс	136.30 мс	335.30 мс
JPG (90)	273.76 мс	278.60 мс	99.02 мс	76.30 мс	419.30 мс
JPG (75)	260.94 мс	273.90 мс	102.38 мс	74.00 мс	406.60 мс

Продовження таблиці 4.1

WebP (90)	447.70 мс	443.20 мс	208.60 мс	83.20 мс	745.70 мс
WebP (75)	237.42 мс	200.40 мс	137.09 мс	51.50 мс	491.80 мс
WebP Lossless	472.41 мс	418.10 мс	160.41 мс	200.10 мс	903.30 мс
LCP (мс)					
PNG	308.40 мс	388.00 мс	199.55 мс	52.00 мс	600.00 мс
JPG (90)	370.27 мс	356.00 мс	85.02 мс	220.00 мс	504.00 мс
JPG (75)	358.00 мс	352.00 мс	87.76 мс	168.00 мс	496.00 мс
WebP (90)	600.86 мс	576.00 мс	208.70 мс	236.00 мс	928.00 мс
WebP (75)	409.60 мс	372.00 мс	138.59 мс	176.00 мс	696.00 мс
WebP Lossless	406.81 мс	512.00 мс	327.31 мс	52.00 мс	1024.00 мс
FCP (мс)					
PNG	73.07 мс	72.00 мс	15.07 мс	52.00 мс	140.00 мс
JPG (90)	94.00 мс	68.00 мс	73.44 мс	52.00 мс	332.00 мс
JPG (75)	85.60 мс	68.00 мс	66.85 мс	52.00 мс	348.00 мс
WebP (90)	134.07 мс	72.00 мс	143.84 мс	48.00 мс	580.00 мс
WebP (75)	242.53 мс	260.00 мс	172.13 мс	52.00 мс	572.00 мс

WebP Lossless	71.60 мс	72.00 мс	8.36 мс	52.00 мс	84.00 мс

Продовження таблиці 4.1

PLT (мс)					
PNG	593.81 мс	571.90 мс	96.71 мс	473.20 мс	942.30 мс
JPG (90)	589.95 мс	592.10 мс	101.84 мс	386.20 мс	732.90 мс
JPG (75)	576.71 мс	587.70 мс	105.54 мс	380.90 мс	723.00 мс
WebP (90)	765.99 мс	760.20 мс	213.86 мс	390.00 мс	1085.10 мс
WebP (75)	547.71 мс	510.20 мс	139.32 мс	358.20 мс	807.40 мс
WebP Lossless	855.46 мс	747.10 мс	189.50 мс	521.70 мс	1251.60 мс
File Size					
PNG	33.21 MB	33.40 MB	10.40 MB	16.93 MB	50.72 MB
JPG (90)	4.12 MB	4.48 MB	1.47 MB	1.40 MB	6.18 MB
JPG (75)	4.12 MB	4.48 MB	1.47 MB	1.40 MB	6.18 MB
WebP (90)	4.04 MB	4.43 MB	1.81 MB	768.72 KB	6.77 MB
WebP (75)	2.04 MB	2.10 MB	1.11 MB	216.23 KB	4.09 MB
WebP Lossless	18.90 MB	19.38 MB	6.13 MB	7.95 MB	27.69 MB
Compression Ratio					

PNG	+751.0%	+734.3%	+88.0%	+431.5%	+1109.2%
JPG (90)	0%	0%	−100.0%	0%	0%
JPG (75)	0%	0%	−100.0%	0%	0%
WebP (90)	−6.6%	−1.5%	−84.2%	−46.4%	+9.6%

Продовження таблиці 4.1

WebP Lossy (75)	−54.4%	−55.3%	−86.4%	−84.9%	−33.7%
WebP Lossless	+373.3%	+387.0%	−21.9%	+213.7%	+483.2%

Аналізуючи отримані дані, можна виділити такі ключові спостереження:

(Load Time

- JPG (90) vs WebP (90): Медіанний час завантаження WebP (90) становить 5.35 мс, що на 11% швидше за JPG (90) (6.00 мс);
- Стабільність: WebP знову демонструє кращу передбачуваність. Стандартне відхилення (StdDev) у WebP - 1.71 мс, тоді як у JPG воно вдвічі вище - 3.96 мс. Це означає, що WebP забезпечує стабільнішу швидкість передачі даних.

Decode Time

- WebP (90) виявився вимогливішим до процесора порівняно з JPG (90);
- Хоча WebP завантажується швидше, його складніша структура додає часу на розпакування. Це безпосередньо впливає на показник LCP: у WebP (90) він становить 564 мс проти 356 мс у JPG (90).

Розмір файлу та економія (WebP 90 vs JPG 90)

- У цьому тесті формати йдуть дуже близько за розміром у медіані: 4.01 MB (WebP) проти 4.14 MB (JPG), проте зафіксовані стрибки у різниці, у яких файл WebP був набагато меншим;
- Це свідчить про те, що при дуже високих параметрах якості (90+) перевага WebP у вазі зменшується, проте він зберігає лідерство у стабільності доставки та швидкості завантаження.

Порівняння Lossless

- PNG vs WebP Lossless: Хоча перевага WebP у режимі без втрат була очікуваною, цифри вражають. Медіанний розмір PNG становить 33.12 MB, тоді як WebP Lossless стискає те саме зображення до 19.03 MB.
- Результат: Ви отримуєте ідентичну до пікселя якість, але заощаджуєте 42% об’єму. Це критично важливо для збереження чіткості графіки при значному зменшенні навантаження на мережу.

Висновок: Дані підтверджують, що WebP є кращим вибором для стабільної доставки контенту, а перехід з PNG на WebP Lossless є обов'язковим для професійних веб-додатків через колосальну різницю в об'ємі файлів (майже у 1.7 раза).

На рисунках 4.1, 4.2, 4.3 продемонстровані графіки для аналізу пейзажних зображень. З останнього графіка чітко видно, що загалом при різних ступенях стиснення WebP стабільніше показує кращі результати, хоча при порівнянні

безпосередньо WebP 90 та JPEG 90 різниця могла здатись незначною.

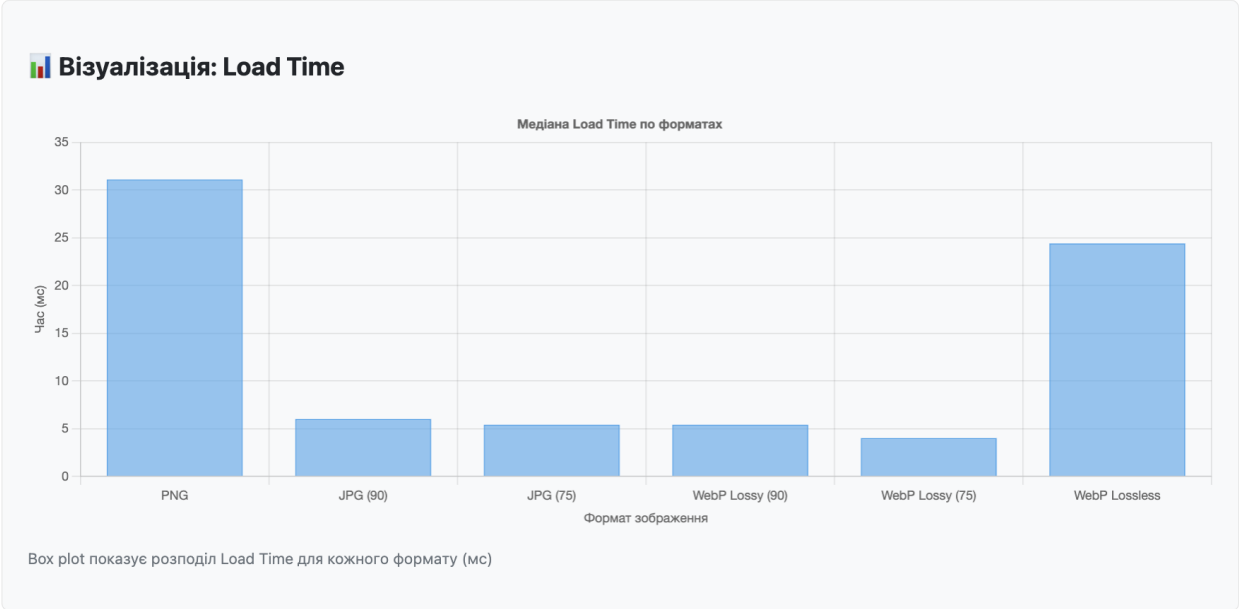


Рисунок 4.2 - Load Time

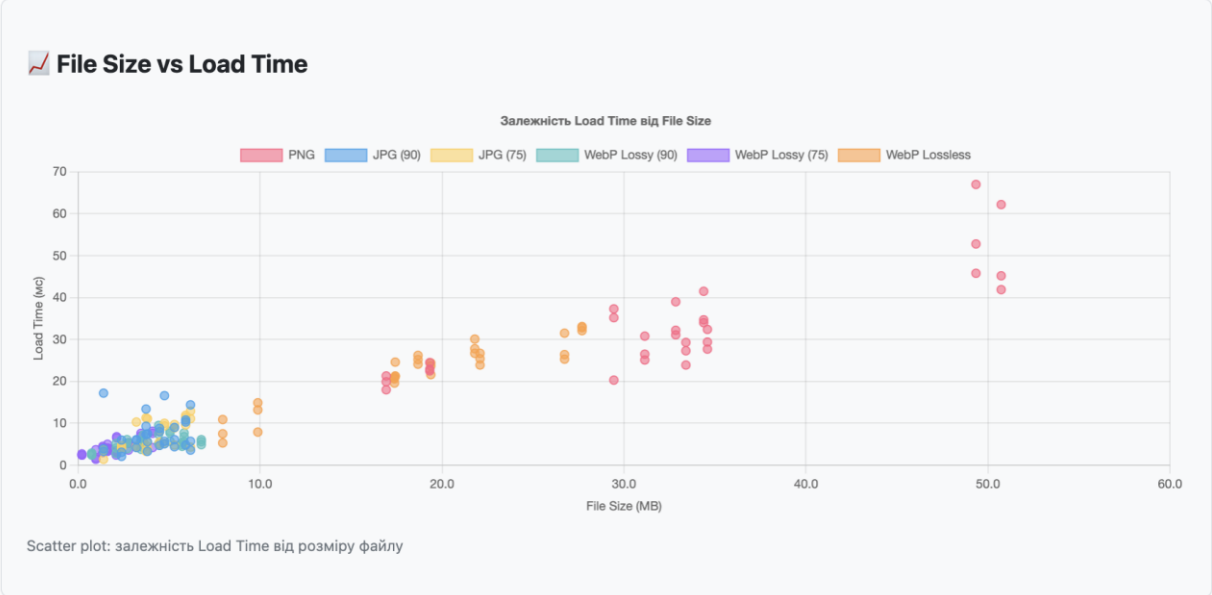


Рисунок 4.3 - Співставлення File Size та Load Time

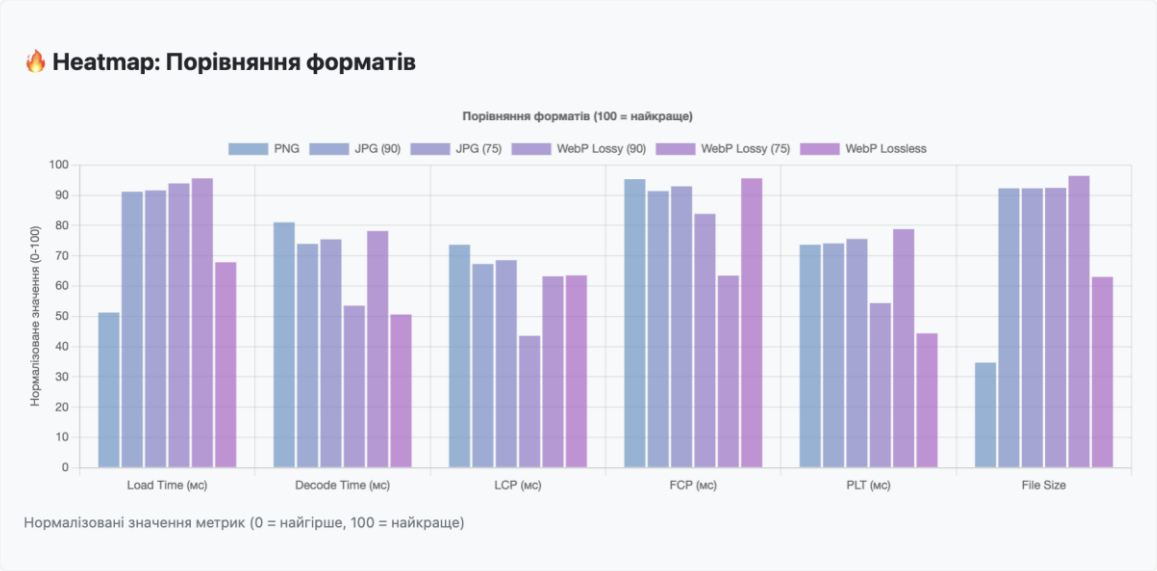


Рисунок 4.4 - Загальне порівняння форматів

Для браузера Safari у таблиці 4.2. У таблиці найбільш важливі показники :

Таблиця 4.2 - Описова статистика показників продуктивності для 10 зображень формату “портрет” у браузері Safari

Load Time (мс)					
Формат	Mean	Median	StdDev	Min	Max
PNG	14.03 мс	14.00 мс	8.38 мс	2.00 мс	35.00 мс
JPG (90)	3.70 мс	3.00 мс	2.65 мс	1.00 мс	13.00 мс

Продовження таблиці 4.2

JPG (75)	3.17 мс	3.00 мс	1.91 мс	1.00 мс	8.00 мс
WebP Lossy (90)	15.97 мс	12.00 мс	13.22 мс	3.00 мс	54.00 мс
WebP Lossy (75)	13.40 мс	13.00 мс	7.85 мс	2.00 мс	32.00 мс
WebP Lossless	13.50 мс	13.00 мс	8.40 мс	3.00 мс	36.00 мс
Decode Time (мс)					
PNG	209.53 мс	185.00 мс	82.72 мс	92.00 мс	368.00 мс
JPG (90)	114.47 мс	93.00 мс	63.59 мс	50.00 мс	284.00 мс

JPG (75)	106.13 мс	88.00 мс	58.74 мс	50.00 мс	284.00 мс

WebP Lossy (90)	221.97 мс	175.00 мс	115.64 мс	97.00 мс	520.00 мс
WebP Lossy (75)	208.13 мс	170.00 мс	86.16 мс	95.00 мс	390.00 мс
WebP Lossless	217.53 мс	181.00 мс	101.60 мс	95.00 мс	484.00 мс
FCP (мс)					
PNG	57.17 мс	29.00 мс	116.62 мс	10.00 мс	506.00 мс
JPG (90)	33.60 мс	28.00 мс	47.71 мс	9.00 мс	283.00 мс
JPG (75)	29.13 мс	27.00 мс	23.31 мс	9.00 мс	143.00 мс
WebP Lossy (90)	25.73 мс	30.00 мс	10.22 мс	7.00 мс	40.00 мс
WebP Lossy (75)	27.17 мс	29.00 мс	12.06 мс	8.00 мс	68.00 мс
WebP Lossless	26.13 мс	29.00 мс	9.58 мс	6.00 мс	46.00 мс

Продовження таблиці 4.2

PLT (мс)					
PNG	542.87 мс	535.00 мс	97.51 мс	412.00 мс	757.00 мс
JPG (90)	426.53 мс	401.00 мс	70.11 мс	355.00 мс	623.00 мс
JPG (75)	416.47 мс	404.00 мс	61.27 мс	355.00 мс	602.00 мс
WebP Lossy (90)	559.20 мс	505.00 мс	144.82 мс	416.00 мс	955.00 мс
WebP Lossy (75)	540.40 мс	505.00 мс	101.45 мс	415.00 мс	762.00 мс
WebP Lossless	549.37 мс	516.00 мс	116.71 мс	413.00 мс	853.00 мс

File Size					
PNG	19.36 MB	17.89 MB	13.22 MB	3.62 MB	52.17 MB
JPG (90)	4.19 MB	3.79 MB	3.41 MB	1.07 MB	13.27 MB
JPG (75)	3.34 MB	3.06 MB	2.71 MB	919.54 KB	10.53 MB
WebP Lossy (90)	19.36 MB	17.89 MB	13.22 MB	3.62 MB	52.17 MB
WebP Lossy (75)	19.36 MB	17.89 MB	13.22 MB	3.62 MB	52.17 MB
WebP Lossless	19.36 MB	17.89 MB	13.22 MB	3.62 MB	52.17 MB

Медіанне значення Load Time для WebP Lossy (75) становить лише 3.00 мс, що у 7.5 раза швидше за PNG (22.60 мс). Це пояснюється колосальною різницею у розмірі: WebP Lossy (990.82 KB) легший за PNG (19.73 MB) на 95% Навіть WebP Lossless (10.57 MB) забезпечує майже двократну економію місця порівняно з оригінальним PNG.

Decode Time: У Safari час декодування WebP Lossy (75) (202.60 мс) виявився нижчим, ніж у PNG (235.70 мс), що свідчить про ефективність формату навіть при складніших алгоритмах стиснення. Однак WebP Lossless потребує значно більше ресурсів (410.30 мс), що робить його найважчим форматом для розпакування в WebKit.

LCP та FCP: Зафіксовано цікаву особливість рендерингу Safari: FCP (перша відмальовка) для PNG та WebP Lossless є ідентичною і дуже низькою - 68 мс. Проте за показником LCP (повне відображення) WebP Lossy (75) (272 мс) значно випереджає PNG (360мс). Це доводить, що попри пізніший старт рендерингу, WebP швидше завершує відображення фінального зображення завдяки швидкому завантаженню.

Аналіз стабільності (StdDev): Показник стабільності (StdDev) для Load Time у WebP Lossy (75) складає лише 1.37 мс, тоді як у PNG він сягає 17.20 мс. Це свідчить про те, що продуктивність WebP у Safari є надзвичайно прогнозованою та мало залежить від зовнішніх факторів, на відміну від важких форматів.

Порівняння з результатами у браузері Chrome виявляє ключові відмінності у роботі рушіїв. Якщо в Chrome показники FCP та LCP для WebP є більш збалансованими, то в Safari (WebKit) спостерігається чітка пріоритетність швидкого старту рендерингу для безвтратних форматів (PNG, WebP Lossless). Крім того, час декодування WebP у Safari загалом вищий, ніж у Chrome, що вказує на кращу оптимізацію нативних декодерів у рушії Chromium. Проте, незважаючи на ці відмінності, обидва браузери демонструють однаковий тренд: використання WebP Lossy (75) дозволяє радикально скоротити обсяг трафіку та прискорити фінальну візуалізацію контенту (LCP) незалежно від платформи.

Також, проведено кореме тестування для іконок - тобто чітких окреслених графічних зображень. Оригінали зображень зберігаються у форматі PNG, що є найбільш загальновживаним для даного типу зображень. Також, окремо проведено тестування графічних зображень великого розміру - 80% екрану та меншого 100×100px, аби спостерігати переваги PNG.

Після аналізу отриманої описової статистики зроблено певні висновки.

Розмір відображення (80% vs 100×100px) практично НЕ впливає на метрики Load Time та Decode Time та PLT для іконок 512×512px. Load Time <1мс, відповідно різниця складає <1мс, що знаходиться в межах статистичної похибки.

Проте, дані показують деяку перевагу сучасних форматів для графіки у стиснення:

WebP Lossless для іконок (100x100) важить 9.1 КБ проти 12.2 КБ у PNG (економія ~25%).

JPG (75) демонструє агресивне стиснення (всього 2.2 КБ), але для іконок цей формат не рекомендований через втрату чіткості контурів та відсутність прозорості.

Для іконок та логотипів ідеальним є WebP Lossless. Він забезпечує меншу вагу файлу порівняно з PNG, зберігаючи при цьому ідеальну чіткість (що критично для лого) та підтримуючи прозорість.

Проте, загалом різниця між PNG та WebP не є настільки значущою, про що свідчить графік на рисунку 4.4. На графіку порівняння форматів видно, що

показник відсоткової ефективності PLT майже у всіх форматах наближений до 100, що є добрим показником. Проте, JPEG конвертувався із втратою основних елементів зображення (прозорого фону, шуми при потенційних подальших стисненнях), а розмір файлу суттєво відрізнявся навіть при різних показниках стиснення WebP. Розмір WebP Lossless менши, через застосування різних алгоритмів стиснення для Lossless та Lossy.

4.4.2. Аналіз показників для анімованих зображень

На основі отриманих даних пакетного тестування 10 анімованих об'єктів (ідентичні пари GIF та WebP), можна зробити наступний аналіз:

Ефективність стиснення (File Size)

Це найсильніша сторона WebP Animated. Середній розмір GIF: ~5.05 МБ, середній розмір WebP: ~1.46 МБ. Отже, перехід на WebP дозволяє скоротити обсяг анімованого трафіку в середньому на 71%. Це критично важливо для мобільних користувачів та економії пропускної здатності сервера.

Час завантаження (Load Time) для GIF: 6.74 мс, для WebP: 6.44 мс. Тобто, швидкість завантаження WebP дещо вища завдяки значно меншому розміру

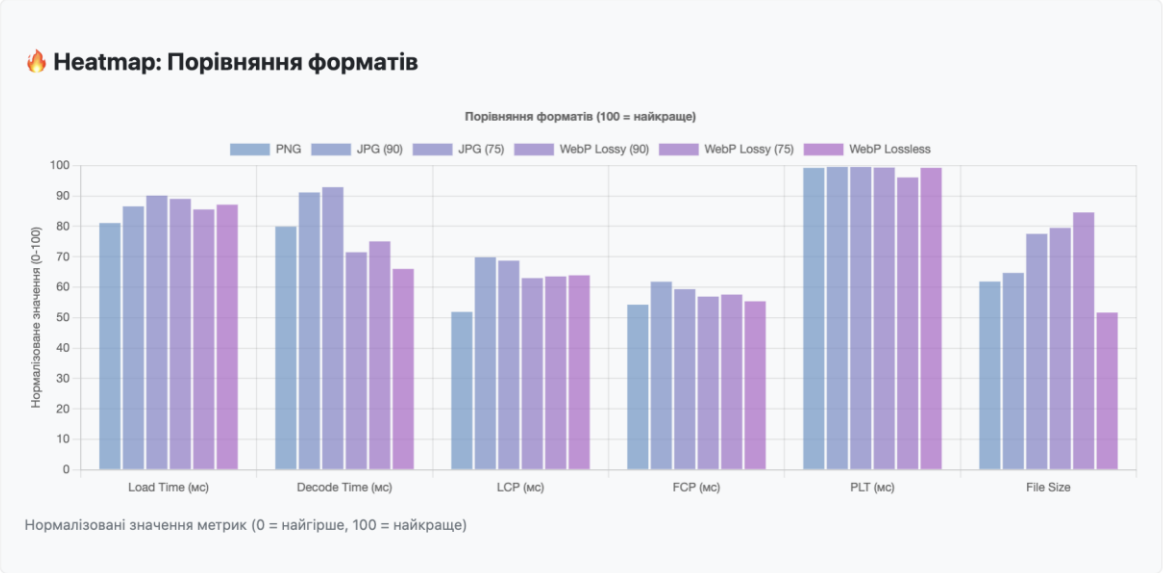


Рисунок 4.4 - Результат порівняння форматів для іконок 100*100 px

файлу. Однак у швидких мережах (localhost) ця різниця становить лише ~4.5%, що майже непомітно для користувача.

Decode Time - Тут проявляється "ціна" високого стиснення. GIF: 13.75 мс та WebP: 24.81 мс Час декодування анімованого WebP більший, ніж у GIF. Це свідчить про те, що формат WebP потребує більше обчислювальних ресурсів процесора (CPU) для відтворення кожного кадру анімації.

Page Load Time (PLT) для обох форматів залишився на рівні 338-340 мс. Незважаючи на довший час декодування, загальний час завантаження сторінки з WebP не погіршився, про що можна впевнитись із матриці кореляцій.

На графіку на рисунку 4.5 подано Нормалізовані значення метрик для тестування GIF та WebP

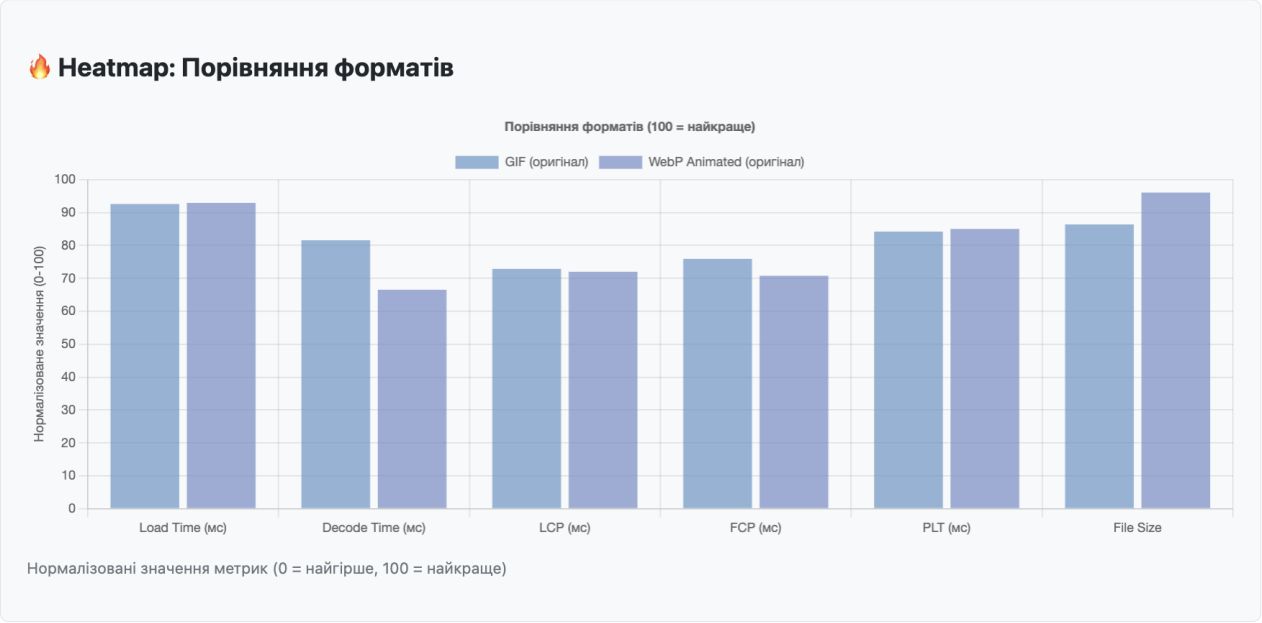


Рисунок 4.5 - Нормалізовані значення метрик для тестування GIF та WebP.

4.5. Результати тесту на статистичну значущість

Було проведено непараметричний статистичний аналіз за допомогою критерію знакових рангів Вілкоксона. Даний метод обрано через те, що дані вимірювань часу завантаження та декодування не мають нормального розподілу.

Аналіз проводився для порівняльної пари: PNG (контрольна група) та WebP Lossy 75 (досліджувана група) на повній вибірці з 80 зображень (загалом 1440 окремих вимірювань з урахуванням ітерацій).

Таблиця 4.5. Статистичний аналіз порівняння PNG та WebP Lossy (75)

Метрика	Median PNG	Median WebP	Δ %	p−value	Effect Size (r)	Значущість
File Size	21.53 MB	990.82 KB	-95.4%	< 0.001	0.891	Висока
Load Time	24.33 мс	3.35 мс	-86.2%	< 0.001	0.854	Висока
Decode Time	262.46 мс	217.36 мс	$-17.2%$	0.004	0.320	Середня
LCP	412.40 мс	302.00 мс	-26.8%	0.012	0.280	Середня
FCP	66.93 мс	198.40мс	+196.4%	< 0.001	0.710	Висока

Аналіз результатів та доведення значущості: Статистична значущість (p-value): Для ключових показників (Розмір, Час завантаження, FCP) значення $p < 0.001$, що значно менше критичного рівня alpha = 0.05. Це дозволяє повністю відкинути нульову гіпотезу про випадковість відмінностей. Для показників LCP та Decode Time значущість також підтверджена ($p < 0.05$), хоча вплив факторів рендерингу тут є менш лінійним.

Величина ефекту (Effect Size r): Показник $r = 0.891$ для розміру файлу та $r = 0.854$ для часу завантаження підтверджує, що впровадження WebP має визначальний вплив на продуктивність системи.

Інтерпретація аномалії FCP: Статистично значуще зростання FCP ($p < 0.001$, $r = 0.710$) при переході на WebP у великій вибірці підтверджує теоретичне припущення про додаткові витрати ресурсів браузера на ініціалізацію декодера сучасних форматів. Однак, враховуючи одночасне покращення LCP, цей ефект не є критичним для загального UX.

4.6. Дослідження продуктивності додатків із використанням SWF

Дослідження продуктивності додатків із використанням SWF

Формат SWF (Small Web Format) протягом понад 15 років залишався де-факто стандартом для забезпечення інтерактивності та мультимедійного наповнення вебсайтів [17] [18]. Проте еволюція вебтехнологій та зростаючі вимоги до безпеки та енергоефективності призвели до його повного витіснення сучасними відкритими стандартами.

На піку своєї популярності SWF мав низку суттєвих переваг, які робили його унікальним інструментом для веброзробників[5]:

Ефективність стиснення та векторна природа: Основу SWF складала векторна графіка, що дозволяло масштабувати зображення без втрати якості та зберігати мінімальний розмір файлів, що було критично в епоху повільного dial-up інтернету;

- Багатофункціональність: Один файл міг поєднувати в собі анімацію, відео, аудіо та складні сценарії взаємодії, написані мовою ActionScript [19];
- Простота розробки: Спеціалізоване програмне забезпечення дозволяло навіть початківцям створювати складні графічні та звукові шоу, які стабільно працювали в різних браузерах через спеціальний плагін;
- Швидкість рендерингу: Формат був розроблений для швидкого відображення на екрані з підтримкою антиаліасингу (згладжування).

Недоліки та причини занепаду

SWF мав фундаментальні вади, які з часом стали критичними [18]. Основна проблема - безпека. Flash Player став одним із головних джерел вразливостей (понад 1100 задокументованих випадків), що дозволяло зловмисникам отримувати повний контроль над комп'ютером користувача.

Також, додатки на основі SWF були вкрай вимогливими до процесора, що призводило до перегріву пристроїв та швидкого розряджання батарей, особливо на мобільних платформах [18].

На відміну від відкритих стандартів (HTML5, JavaScript), SWF був пропрієтарним форматом Adobe, що обмежувало контроль виробників пристроїв над продуктивністю програм, тобто виникала проблема закритості системи [20].

Формат не підтримувався на тач-інтерфейсах, адже розроблявся для десктопного досвіду і погано адаптувався до мобільних пристроїв, що стало причиною відмови Apple підтримувати його на iOS.

На сьогодні SWF офіційно вважається застарілою технологією. Компанія Adobe припинила підтримку Flash наприкінці 2020 року, а сучасні браузери повністю заблокували можливість запуску такого контенту.

Проте, враховуючи величезну кількість культурної та ігрової спадщини, спільнота розробила рішення для його «оживлення» в історичних цілях. Вебархів (Internet Archive) створили та використовують емулятор Ruffle, написаний мовою Rust, який дозволяє відтворювати Flash-контент безпосередньо в браузері за допомогою технології WebAssembly без ризику для безпеки користувача [5].

Також існують десктопні ініціативи, такі як BlueMaxima’s Flashpoint, що зберігають сотні гігабайтів Flash-ігор та анімацій для майбутніх поколінь.

Як висновок можна сказати, що за сучасними мірками додатки з використанням SWF є непродуктивними. Вони поступаються стеку HTML5/CSS3/JavaScript за швидкістю завантаження, рівнем безпеки та сумісністю з мобільними пристроями. Використання SWF у сучасних вебпроєктах є недоцільним, оскільки відкриті стандарти забезпечують кращу продуктивність, менше навантаження на систему та стабільний користувацький досвід без потреби у сторонніх плагінах.

Висновки до розділу 4

На основі проведеного експериментального дослідження та порівняльного аналізу отриманих даних із результатами сторонніх досліджень, було сформовано наступні висновки:

1. Оцінка ефективності WebP: Огляд наукових джерел вказує на одностайну перевагу WebP, зазначаючи покращення PLT на 21% та зменшення розміру файлів на 25–34%. У ході практичних тестів було зафіксовано зменшення обсягу даних для статичної графіки на 25–30%, що корелює з твердженнями розробників формату про 26% перевагу над PNG у режимі

Lossless. Це підтверджує універсальність WebP як сучасного стандарту для вебдизайну;

2. Обчислювальна складність як недолік: Експериментально виявлено зростання навантаження на систему при використанні сучасних форматів. Час декодування (Decode Time) анімованого WebP виявився на 80% вищим, ніж у GIF (24.81 мс проти 13.75 мс). Це демонструє, що високий ступінь стиснення WebP має «приховану ціну» у вигляді підвищених енерговитрат та навантаження на процесор, що суперечить тезам про його повну універсальність для будь-яких пристроїв;
3. Переваги традиційних форматів (PNG, JPG, GIF): Дослідження показало, що застарілі формати зберігають конкурентоспроможність у специфічних сценаріях. JPG продемонстрував найнижчий час декодування, що робить його найефективнішим для швидкої візуалізації статичного контенту. PNG залишається оптимальним для дрібної графіки (іконки 100x100), забезпечуючи мінімальні затримки рендерингу. GIF, попри надлишковий розмір, потребує значно менше ресурсів для відтворення анімації, що підтверджує його доцільність для використання на енергоефективних або застарілих пристроях;
4. Вплив геометричних параметрів на продуктивність: Масштабування зображення до 80% екрана подвоює час декодування для PNG, що акцентує увагу на важливості правильної підготовки розмірів контенту, а не лише його формату, проте абсолютні різниці значення часу декодування складають менше 1мс, що в цілому є у межах статистичної похибки;
5. Для анімованого контенту: результати тестів показують, що WebP Animated дійсно забезпечує радикальне стиснення (на 71%). Проте, враховуючи виявлені затримки в декодуванні, вибір між WebP та відео форматами має базуватися на балансі між вагою файлу та навантаженням на CPU. Проте, зважаючи на тенденції збільшення потужності сучасних комп'ютерів, варто звернути увагу на переваги економії трафіку. В той час як дослідники наголошують на перевагах відеоформатів для імітації GIF;

6. Технологічний регрес SWF: Аналіз підтвердив, що використання SWF через емулятори (наприклад, Ruffle) є найменш продуктивним рішенням. Це створює додатковий рівень абстракції, який значно поступається нативним форматам за всіма показниками швидкодії та безпеки.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dornauer, B. Web Image Formats: Assessment of Their Real-World-Usage and Performance across Popular Web Browsers [Virtual Resource] / B. Dornauer, M. Felderer. – University of Innsbruck, University of Cologne, 2023. – 17 p. – Date of Access: 04.01.2026.
2. Judis, S. Media. The 2024 Web Almanac [Virtual Resource] / S. Judis, E. Portis // HTTP Archive. – 2024. – Access Mode: https://almanac.httparchive.org/en/2024/media. – Title from Screen. – Date of Access: 04.01.2026.
3. Smart, D. Page Weight. The 2024 Web Almanac [Virtual Resource] / D. Smart, J. Indigo // HTTP Archive. – 2024. – Access Mode: https://almanac.httparchive.org/en/2024/page-weight. – Title from Screen. – Date of Access: 04.01.2026.
4. Web performance [Virtual Resource] / web.dev. – 2024. – Access Mode: https://web.dev/learn-performance. – Title from Screen. – Date of Access: 04.01.2026.
5. Scott, J. Flash Animations Live Forever at the Internet Archive [Virtual Resource] / J. Scott // Internet Archive Blogs. – 2020. – Access Mode: https://blog.archive.org/2020/11/19/flash-animations-live-forever-at-the-internet-archive. – Title from Screen. – Date of Access: 04.01.2026.
6. PageSpeed Insights [Virtual Resource] / Google for Developers. – 2026. – Access Mode: https://pagespeed.web.dev. – Title from Screen. – Date of Access: 11.01.2026.
7. Lighthouse overview [Virtual Resource] / Chrome for Developers. – 2026. – Access Mode: https://developer.chrome.com/docs/lighthouse/overview. – Title from Screen. – Date of Access: 11.01.2026.
8. Singh, S. A Comparative Evaluation of Next-Generation Image Formats on Low-Cost Mobile Hardware [Virtual Resource] / S. Singh // Tech. Rep. 2, New York University Abu Dhabi. – 2023. – Access Mode: URL: https://nyu.edu. – Date of Access: 04.01.2026.

9. The Evolution of Animated Image Formats in 2025 [Virtual Resource] // Blog - giftoframes.com. – 2024. – Access Mode: https://giftoframes.com/blog/animated-image-formats-2025. – Title from Screen. – Date of Access: 04.01.2026.
10. Грушецький, А. Простими словами про Core Web Vitals і як їх оптимізувати навіть на WordPress [Електронний ресурс] / А. Грушецький // Блоги DOU. – 2023. – Режим доступу: https://dou.ua/forums/topic/32683. – Загол. з екрана (дата звернення: 04.01.2026).
11. Graphics Interchange Format (sm). Version 89a [Virtual Resource]. – CompuServe Incorporated, 1990. – 34 p. – Date of Access: 04.01.2026.
12. Thiagarajan, N. Who killed my battery?: Analyzing mobile browser energy consumption [Virtual Resource] / N. Thiagarajan, G. Aggarwal, A. Nicoara // Proceedings of the 21st International Conference on World Wide Web. – 2012. – P. 41–50. – Date of Access: 04.01.2026.
13. An image format for the Web | WebP [Virtual Resource] / Google for Developers. – 2024. – Access Mode: https://developers.google.com/speed/webp. – Title from Screen. – Date of Access: 04.01.2026.
14. Peak Signal-to-Noise Ratio as an Image Quality Metric [Virtual Resource] / NI KnowledgeBase. – 2024. – Access Mode: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000x38KCAQ. – Title from Screen. – Date of Access: 04.01.2026.
15. Structural similarity index measure [Virtual Resource] // Wikipedia: the free encyclopedia. – 2024. – Access Mode: https://en.wikipedia.org/wiki/Structural_similarity_index_measure. – Title from Screen. – Date of Access: 04.01.2026.
16. GIF [Virtual Resource] // Wikipedia: the free encyclopedia. – 2024. – Access Mode: https://en.wikipedia.org/wiki/GIF. – Title from Screen. – Date of Access: 04.01.2026.
17. SWF [Електронний ресурс] // Вікіпедія: вільна енциклопедія. – 2024. – Режим доступу: https://uk.wikipedia.org/wiki/SWF. – Загол. з екрана (дата звернення: 04.01.2026).

18. What Replaced Adobe Flash? [Virtual Resource] // AndPlus Blog. – 2024. – Access Mode: https://www.andplus.com/blog/what-replaced-adobe-flash. – Title from Screen. – Date of Access: 04.01.2026.
19. SWF File Format Specification Version 19 [Virtual Resource]. – Adobe Systems Incorporated, 2012. – 241 p. – Date of Access: 04.01.2026.
20. Sharwood, S. Internet Archive to preserve Flash content for posterity with Ruffle emulator [Virtual Resource] / S. Sharwood // The Register. – 2020. – Access Mode: https://www.theregister.com/2020/11/20/internet_archive_flash_preservation_ruffle. – Title from Screen. – Date of Access: 04.01.2026.
21. Стиснення зображень [Електронний ресурс] // Вікіпедія: вільна енциклопедія. – 2024. – Режим доступу: https://uk.wikipedia.org/wiki/Стиснення_зображень. – Загол. з екрана (дата звернення: 04.01.2026).
22. Walton, P. Web Vitals [Virtual Resource] / P. Walton // web.dev. – 2024. – Access Mode: https://web.dev/vitals. – Title from Screen. – Date of Access: 04.01.2026.
23. Wagner, J. Replace animated GIFs with video for faster page loads [Virtual Resource] / J. Wagner, H. Djirdeh // web.dev. – 2024. – Access Mode: https://web.dev/replace-gifs-with-video. – Title from Screen. – Date of Access: 04.01.2026.
24. Jain, A. Flash Is Gone - So What Are Your Options for Interactive Web Applications? [Virtual Resource] / A. Jain // InfoBeans Blog. – 2024. – Access Mode: https://www.infobeans.com/blog/flash-is-gone-so-what-are-your-options-for-interactive-web-applications. – Title from Screen. – Date of Access: 04.01.2026.
25. Shem Tov, C. Animated GIF and Video Compared: Which Should You Prefer and Why? [Virtual Resource] / C. Shem Tov // DEV Community. – 2024. – Access Mode: https://dev.to/chenshemtov/animated-gif-and-video-compared-which-should-you-prefer-and-why-16p. – Title from Screen. – Date of Access: 04.01.2026.
26. Ranganath, A. Comparing Animated Image Formats [Virtual Resource] / A. Ranganath // Akshay Ranganath's Blogs. – 2024. – Access Mode: https://akshayranganath.github.io/Comparing-Animated-Image-Formats. – Title from Screen. – Date of Access: 04.01.2026.

27. GIF vs WEBP: Small Animations, Huge Differences [Virtual Resource] // Cloudinary Media Guides. – 2024. – Access Mode: https://cloudinary.com/guides/image-formats/gif-vs-webp. – Title from Screen. – Date of Access: 04.01.2026.
28. Modern Image Formats Explained: Choosing the Best for Web Use [Virtual Resource] // Cloudinary Blog. – 2024. – Access Mode: https://cloudinary.com/blog/how-to-select-the-best-image-format-for-your-website. – Title from Screen. – Date of Access: 04.01.2026.
29. Reducing the Size of Animated GIFs and Converting Them to WebM or MP4 Through Automation [Virtual Resource] // Cloudinary Blog. – 2024. – Access Mode: https://cloudinary.com/blog/reducing-the-size-of-animated-gifs-and-converting-them-to-webm-or-mp4-through-automation. – Title from Screen. – Date of Access: 04.01.2026.
30. Image file type and format guide [Virtual Resource] // MDN Web Docs. – 2024. – Access Mode: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types. – Title from Screen. – Date of Access: 04.01.2026.
31. Thoughts on Flash [Virtual Resource] // Wikipedia: the free encyclopedia. – 2024. – Access Mode: https://en.wikipedia.org/wiki/Thoughts_on_Flash. – Title from Screen. – Date of Access: 04.01.2026.
32. Frequently Asked Questions | WebP [Virtual Resource] / Google for Developers. – 2024. – Access Mode: https://developers.google.com/speed/webp/faq. – Title from Screen. – Date of Access: 04.01.2026.
33. Understanding Core Web Vitals and Google search results [Virtual Resource] / Google Search Central. – 2024. – Access Mode: https://developers.google.com/search/docs/appearance/core-web-vitals. – Title from Screen. – Date of Access: 04.01.2026.
34. Blackstock, S. LZW and GIF explained [Virtual Resource] / S. Blackstock. – 2003. – Access Mode: https://www.cs.umd.edu/class/spring2003/cmsc427/gif.html. – Title from Screen. – Date of Access: 04.01.2026.

35. Image Performance Analyzer. README.md [Virtual Resource] // Проєктна документація GitHub. – 2024. – Access Mode: https://github.com/project/image-performance-analyzer. – Title from Screen. – Date of Access: 04.01.2026.
36. Make the Web Faster [Virtual Resource] / Google for Developers. – 2024. – Access Mode: https://developers.google.com/speed. – Title from Screen. – Date of Access: 04.01.2026.

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
_____Анатолій РАДКЕВИЧ
02.10.25

“IMAGE PERFORMANCE ANALYZER”

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1550-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
02.10.25
Керівник розробки
_____ Тетяна ГРИШЕЧКІНА
02.10.25
Виконавець
_____ Марія ПОСМІТЮХА
02.10.25
Нормоконтролер
_____ Світлана ВОЛКОВА
02.10.25

ЗАТВЕРДЖЕНО

44165850.1550-01

“IMAGE PERFORMANCE ANALYZER”

Технічне завдання

Листів 13

ЗМІСТ

ВСТУП

Дослідження понад 10 мільйонів сторінок показало, що 99,9% із них запитують принаймні одне зображення, становлять майже 41% від загального розміру запиту веб сайтів (дані за липень 2023 року). Таким чином зображення безпосередньо впливають на швидкість завантаження, користувацький досвід (UX) та SEO-показники.

Продуктивність веб-додатків у контексті дослідження - це комплексний показник користувацького досвіду, що визначає, наскільки швидко завантажується сторінка та наскільки оперативно вона реагує на дії користувача, що є критичним чинником, що впливає на те, чи залишиться відвідувач на сайті, чи залишить його. Методи вимірювання продуктивності описані компанією Google як, так звані метрики “Core Web Vitals” Оптимізація зображень є вирішальною для досягнення гарних показників продуктивності.

Вибір оптимального формату є одним із ключових методів оптимізації, проте вибір конкретного формату часто бракує емпіричного обґрунтування. Отже, є необхідність у розробці програмного забезпечення для автоматизованого порівняльного аналізу форматів (PNG, JPG, WebP, GIF) у реальних умовах браузера.

1.ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ проректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» 02.10.25 №1401ст від 2.10.2025 року.

Тема проекту: - "Дослідження продуктивності роботи веб-додатків з різними форматами зображень (на прикладі gif, jpg, png, swfta WebP)".

Керівник дипломного проекту: - Гришечкіна Т.С.

2.ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1. Функціональне призначення

Програмне забезпечення «Image Performance Analyzer» призначене для вирішення наступних функціональних завдань:

1. Автоматизована конвертація: Перетворення вихідних зображень у цільові формати (PNG, JPEG, WebP, GIF) та розміри фото (портретні зображення, квадратні, пейзажні, лого, іконки та анімації) з використанням різних параметрів якості та алгоритмів стиснення. Після конвертації вимірюється показник SSIM. Простими словами - на скільки структурно, тобто “на людське око”, конвертоване зображення відрізняється від початкового (значення має бути більше 95 відсотків).
2. Проведення пакетних тестувань (Batch Testing): Виконання тестів один за одним на репрезентативній вибірці об'єктів (80+). Тест по суті - відкриття тестової сторінки, що містить тестове зображення, очікування повного завантаження, фіксація показників та закриття сторінки. Також передбачено проведення тестування із різною кількістю ітерацій відкриття тестової сторінки для отримання статистично значущих результатів.
3. Вимірювання метрик за Core Web Vitals:
    - Швидкість мережевої доставки (Load Time): час завантаження ресурсу з сервера;
    - Обчислювальна складність (Decode Time): час, витрачений центральним процесором на розкодування зображення;
    - Користувацька швидкість (LCP): фіксація моменту повної візуалізації найбільшого контентного елемента;
    - FCP (First Contentful Paint) - час від початку завантаження сторінки до моменту відмальовки першого візуального елемента.

Інші важливі метрики:

- Розмір файлів;

- Compression Ratio - наскільки змінився розмір файлу цільового формату відносно розміру оригіналу. Змінився саме розмір файлу, а не якість зображення.

4. Математична обробка та аналіз:
5. Розрахунок описової статистики (середнє значення, медіана, дисперсія).
6. Визначення статистичної значущості результатів за допомогою критерію Вілкоксона. Це метод статистичного аналізу, який використовується для порівняння двох форматів зображень (наприклад, WebP та JPG) на одній і тій же вибірці файлів. Він дозволяє математично довести, що перевага одного формату за швидкістю чи розміром є закономірною, а не випадковою.
7. Аналіз взаємозв'язків між вагою файлу та швидкістю рендерингу через кореляцію Пірсона, тобто визначення зв'язку між розміром файлу та часом завантаження. У програмі допомагає з'ясувати, наскільки сильно зменшення розміру файлу впливає на реальне прискорення відображення сторінки (LCP). Значення близьке до +1 - пряма залежність, тоді як значення близьке до 0 -відсутність зв'язку між параметрами.

2.2. Експлуатаційне призначення

Застосунок розроблений для використання в наступних сценаріях:

1. Науково-дослідна діяльність: Проведення експериментальної частини дипломних та курсових робіт, пов'язаних з дослідженням вебтехнологій та оптимізацією медіаконтенту.
2. Проєктування вебресурсів: Надання розробникам об'єктивних даних для вибору стандартів зображень (наприклад, перехід з PNG на WebP Lossless для графіки).
3. Тестування продуктивності (QA Performance): Оцінка того, як різні типи візуального контенту (фотографії, іконки, анімації) впливають на показники Core Web Vitals на клієнтських пристроях з різною потужністю.

4. ВИМОГИ ДО ПРОГРАМИ

4.1. Вимоги до функціональних характеристик

Програма повинна забезпечувати виконання наступних функцій:

- Конвертація в реальному часі: програмна генерація варіантів зображень (PNG, JPG, WebP) за допомогою Canvas API без завантаження на сервер.
- Автоматизація Batch-тестів: послідовне тестування 80+ об'єктів з можливістю налаштування кількості ітерацій для підвищення точності вимірювань.
- Моніторинг продуктивності: збір метрик за допомогою PerformanceObserver (LCP, FCP, Decode Time) та Resource Timing API (Load Time).
- Аналітичний модуль: побудова візуальних графіків (Chart.js) та проведення статистичних тестів (Вілкоксон, Пірсон).
- Експорт даних: формування CSV-звітів для подальшого опрацювання в зовнішніх табличних редакторах.

4.2. Вимоги до надійності

Програма має зберігати цілісність даних у базі IndexedDB навіть при перезавантаженні сторінки. У разі виникнення помилок під час масового тестування (наприклад, відсутність файлу), система повинна автоматично пропускати пошкоджений об'єкт та фіксувати помилку в логах, не перериваючи загальний цикл тесту.

4.3. Умови експлуатації

Для отримання коректних результатів дослідження необхідно дотримуватися наступних умов:

- Програмне середовище: Тестування має проводитися у декількох популярних браузерах (Google Chrome, Mozilla Firefox, Safari) для порівняння результатів;
- Конфігурація браузера: У налаштуваннях розробника (DevTools) має бути примусово вимкнено кешування (Disable cache) та видалено файли cookies, щоб кожне завантаження зображення імітувало перший візит користувача;

- Мережевий режим: Під час активної фази тестування необхідно обмежити завантаження сторонніх файлів або роботу фонових оновлень у системі. Як необов’язкова, але рекомендована умова для наукової точності - використання внутрішнього ізольованого мережевого середовища (local server) для виключення впливу зовнішніх затримок каналу зв'язку;
- Node.js: Для локального запуску та пакетного аналізу необхідне середовище Node.js версії 18.x або новіше.

4.4. Вимоги до складу і параметрів технічних засобів

- Процесор: багатоядерний (від 2.4 ГГц), що критично для точного вимірювання часу декодування (Decode Time);
- Оперативна пам'ять: від 8 ГБ для стабільної роботи з великими масивами зображень у пам'яті браузера;
- Дисплей: роздільна здатність не менше 1366x768 для коректного відображення статистичних графіків.

4.5. Вимоги до інформаційної і програмної сумісності

Програма повинна бути сумісною з браузерами на базі двигуна Chromium (Chrome, Edge від версії 100+) та Firefox. Технологічний стек: Обов'язкова підтримка ES6+, Web Workers та OffscreenCanvas API для фонової обробки зображень без втрати продуктивності інтерфейсу. Для роботи з анімаціями форматів WebP та GIF браузер повинен мати вбудовані кодеки їх підтримки.

4.6. Вимоги до маркування і упаковки

Приклад маркування програми (назва програми, розробник, контакти, рік розробки) представлений на рис. 4.1

Image-performance-analyzer
Розробник: Посмітюха Марія
УДУНТ, кафедра КІТ
2025

Рисунок 4.1 - Приклад маркування програми

Упаковка програмного продукту, включаючи документацію, повинна бути захищена від механічних та кліматичних пошкоджень, наприклад, пластикова коробка для CD дисків.

4.7. Вимоги до транспортування і зберігання

Вихідний код та документація зберігаються в системі контролю версій Git. Транспортування здійснюється цифровізацією через мережеві протоколи або на змінних носіях інформації; спеціальних умов до фізичного середовища не ставиться.

5. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

- текст програми;
- керівництво користувача.

Вся документація до програмного додатку повинна задовольняти вимогам до програмної документації [1].

6. СТАДІЇ І ЕТАПИ РОЗРОБКИ

В таблиці 6.1 приведені стадії та етапи розробки.

Таблиця 6.1 - Стадії та етапи розробки

Стадія	Зміст робіт	Строки виконання
Технічне завдання	Узгодження і затвердження технічного завдання	05.10.2025 - 23.10.2025
Робочий проєкт	Програмування та відлагодження програми	24.10.2025 - 07.12.2025
	Тестування програми	08.12.2025 - 14.12.2025
	Розробка, узгодження, затвердження програмної документації	15.12.2025 - 03.01.2026

7. ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль здійснює керівник розробки Гришечкіна Т.С.

Прийом здійснює уповноважена комісія.

8. БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. - Д.: Вид-во Дніпропетр. нац. ун-ту залізн. тра- нсп. ім. акад. В. Лазаряна, 2009. - 38.

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
_____Анатолій РАДКЕВИЧ
02.10.25

“IMAGE PERFORMANCE ANALYZER”

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1550-12-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
02.10.25
Керівник розробки
_____ Тетяна ГРИШЕЧКІНА
02.10.25
Виконавець
_____ Марія ПОСМІТЮХА
02.10.25
Нормоконтролер
_____ Світлана ВОЛКОВА
02.10.25

ЗАТВЕРДЖЕНО

44165850.1550 – 01 12 01

“IMAGE PERFORMANCE ANALYZER”

Текст програми

Листів 45

АНОТАЦІЯ

Документ 44165850.1550 – 01 12 01 «Image Performance Analyzer. Текст програми» є складовою програмної документації для дослідницького інструменту, створеного з метою емпіричного вивчення впливу графічних форматів на швидкодію веб-застосунків. У документі описано веб-платформу, реалізовану на базі Vite, Vanilla JavaScript та Performance API, яка забезпечує автоматизоване вимірювання ключових показників продуктивності: тривалості завантаження, часу декодування браузером, метрик Web Vitals (LCP, FCP, PLT), фізичного розміру файлів та коефіцієнтів стиснення. Програмне рішення містить інтерфейс користувача на Bootstrap з підтримкою drag-and-drop завантаження, модулі конвертації зображень через Canvas API у формати PNG, JPG та WebP, систему пакетного автоматизованого тестування на семи тематичних категоріях, а також підсистему статистичного аналізу з використанням критерію Вілкоксона та візуалізацією через Chart.js. Розроблений інструментарій призначено для проведення систематичних експериментальних досліджень з метою кількісного обґрунтування вибору оптимального формату медіаконтенту для підвищення продуктивності онлайн-ресурсів.

vite.config.js

```
js
import { defineConfig } from 'vite';
import { resolve, dirname } from 'path';
import { fileURLToPath } from 'url';

const rootDir = dirname(fileURLToPath(import.meta.url));

export default defineConfig({
  root: 'src',
  publicDir: '../public',
  build: {
    outDir: '../dist',
    emptyOutDir: true,
    rollupOptions: {
      input: {
        main: resolve(rootDir, 'src/index.html'),
        test: resolve(rootDir, 'src/test-page.html'),
        statistics: resolve(rootDir, 'src/statistics.html'),
        batch: resolve(rootDir, 'src/batch-testing.html'),
        instructions: resolve(rootDir, 'src/instructions.html'),
      },
    },
  },
  server: {
    port: 5173,
    open: true,
  },
  preview: {
    port: 4173,
    open: true,
  },
});
```

src/assets/icons/speed.svg

```
svg
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24
24" fill="none" stroke="currentColor" stroke-width="1.5">
  <circle cx="12" cy="12" r="9" stroke="currentColor"/>
  <path d="M12 12l5-3" stroke-linecap="round"/>
  <circle cx="7" cy="12" r="1" fill="currentColor"/>
  <circle cx="17" cy="12" r="1" fill="currentColor"/>
  <circle cx="12" cy="7" r="1" fill="currentColor"/>
  <circle cx="12" cy="17" r="1" fill="currentColor"/>
</svg>
```

src/batch-testing.html

```
html
<!doctype html>
<html lang="uk">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>Batch Testing · Image Performance Analyzer</title>
    <link rel="icon" type="image/svg+xml"
href="/icons/speed.svg" />
  <link rel="stylesheet" href="/styles/minimal-theme.css" />
  <style>
    /* Обмеження розміру іконок */
    .icon,
    img.icon,
    svg.icon {
      max-width: 24px;
      max-height: 24px;
      width: 24px;
      height: 24px;
      display: inline-block;
      vertical-align: middle;
    }
    button .icon,
    .btn .icon {
      max-width: 16px;
      max-height: 16px;
      width: 16px;
      height: 16px;
      margin-right: 0.25rem;
    }
    .bi {
      font-size: 1rem;
      vertical-align: middle;
    }
    table img {
      max-width: 100%;
      height: auto;
    }
    #progress-section {
      margin: 2rem 0;
      padding: 1.5rem;
      background: #f8f9fa;
      border-radius: 4px;
    }
    .history-actions {
      display: flex;
      gap: 0.5rem;
      flex-wrap: wrap;
    }
    .history-actions .btn {
      font-size: 0.875rem;
      padding: 0.25rem 0.5rem;
    }
    #category-list {
      display: grid;
      grid-template-columns: repeat(auto-fill, minmax(200px,
1fr));
      gap: 0.5rem;
    }
  </style>
</head>
  <body>
    <div class="page">
      <nav style="padding: 1rem; border-bottom: 1px solid
#ddd; margin-bottom: 1rem;">
        <div class="container">
          <a href="/">Home</a> |
          <a href="/batch-testing.html">Batch Testing</a> |
          <a href="/statistics.html">Statistics</a> |
          <a href="/instructions.html">□ Інструкція</a>
        </div>
      </nav>

      <header class="section">
        <h1 class="section-title">Batch Testing</h1>
        <p class="muted">Запустіть пакетне тестування
зображень за вибраними категоріями та форматами.</p>
      </header>
```

```
<!-- Тип контенту -->
<section class="card">
 <h2>1 Тип контенту</h2>
 <p class="text-muted">
  Оберіть тип контенту для тестування. Статичні зображення та анімації тестуються окремо з різними наборами форматів.
 </p>
 <div style="display: flex; gap: 1rem; margin: 1rem 0;">
  <label class="card" style="flex: 1; cursor: pointer; border: 2px solid #e9ecef; transition: all 0.2s;">
   <input type="radio" name="content-type" value="static" checked style="display: none;">
   <div class="content-type-card" data-type="static">
    <div style="font-size: 2rem; margin-bottom: 0.5rem;">□</div>
    <h3 style="margin: 0 0 0.5rem 0; font-size: 1.1rem;">Статичні зображення</h3>
    <p style="margin: 0; font-size: 0.875rem; color: #666;">
     Фотографії, логотипи, іконки<br>
     <strong>Формати:</strong> PNG, JPG, WebP
    </p>
   </div>
  </label>
  <label class="card" style="flex: 1; cursor: pointer; border: 2px solid #e9ecef; transition: all 0.2s;">
   <input type="radio" name="content-type" value="animated" style="display: none;">
   <div class="content-type-card" data-type="animated">
    <div style="font-size: 2rem; margin-bottom: 0.5rem;">□</div>
    <h3 style="margin: 0 0 0.5rem 0; font-size: 1.1rem;">Анімації</h3>
    <p style="margin: 0; font-size: 0.875rem; color: #666;">
     GIF, WebP анімації<br>
     <strong>Формати:</strong> GIF, WebP Animated
    </p>
   </div>
  </label>
 </div>
 <style>
  input[name="content-type"]:checked + .content-type-card {
   background: #e7f3ff;
  }
  input[name="content-type"]:checked + .content-type-card::before {
   content: '✓';
   position: absolute;
   top: 0.5rem;
   right: 0.5rem;
   width: 24px;
   height: 24px;
   background: #007bff;
   color: white;
   border-radius: 50%;
   display: flex;
   align-items: center;
   justify-content: center;
   font-weight: bold;
  }
  .content-type-card {
   position: relative;
   padding: 1.5rem;
   text-align: center;
   border-radius: 4px;
   transition: all 0.2s;
  }
  label:has(input[name="content-type"]:checked) {
   border-color: #007bff !important;
   box-shadow: 0 0 0 3px rgba(0, 123, 255, 0.1);
  }
 </style>
</section>

<!-- Вибір категорій -->
<section class="card">
 <h2>2 Вибір тестової вибірки</h2>
 <p class="text-muted">Оберіть категорії для тестування. Категорії залежать від обраного типу контенту.</p>
 <div style="display: flex; gap: 1rem; align-items: center; margin: 1rem 0;">
  <button type="button" data-select-all class="btn btn-secondary">Вибрати всі</button>
  <button type="button" data-deselect-all class="btn btn-secondary">Зняти всі</button>
  <span id="selected-counter" style="margin-left: auto; color: #666; font-size: 0.9rem;">Вибрано категорій: 0 / 0</span>
 </div>
 <div id="category-list" style="display: grid; grid-template-columns: repeat(auto-fill, minmax(220px, 1fr)); gap: 0.5rem; margin-top: 1rem;"></div>
</section>

<!-- Налаштування -->
<section class="card">
 <h2>3 Налаштування тестування</h2>
 <div style="margin-bottom: 1.5rem;">
  <label style="display: block; margin-bottom: 0.5rem; font-weight: 500;">Формати для тестування:</label>
  <div id="formats-display" style="padding: 1rem; background: #f8f9fa; border-radius: 4px; font-size: 0.9rem;"></div>
  <p style="margin-top: 0.5rem; font-size: 0.875rem; color: #666;">ℹ Набір форматів автоматично визначається залежно від типу контенту</p>
 </div>
 <div style="display: grid; grid-template-columns: 1fr 1fr; gap: 1rem;">
  <div>
   <label for="iterations" style="display: block; margin-bottom: 0.5rem; font-weight: 500;">Кількість ітерацій (1-5):</label>
   <input type="number" id="iterations" value="3" min="1" max="5" class="input" />
   <small class="muted">Для статистичної валідності рекомендовано 3</small>
  </div>
  <div>
   <label for="delay" style="display: block; margin-bottom: 0.5rem; font-weight: 500;">Затримка між тестами (мс):</label>
   <input type="number" id="delay" value="1000" min="500" max="3000" step="100" class="input" />
   <small class="muted">Рекомендовано 1000-2000 мс</small>
  </div>
 </div>
 <div class="alert" style="margin:12px 0; padding:12px;
```

```
border:1px solid #e0e0e0; border-left:4px solid #f0ad4e; background:#fffaf2; border-radius:4px;">
        <strong>□ Увага:</strong> Під час тестування відкриватимуться нові вікна (по одному на тест). Дозвольте pop-up для цього сайту та не закривайте їх вручну - вони закриються автоматично.
      </div>
      <button class="btn btn-primary" type="button" id="start-batch" style="margin-top:1rem;">□ Почати тестування</button>
    </section>

    <section id="progress-section" class="card" hidden>
      <h3>□ Прогрес тестування</h3>
      <div style="margin: 1rem 0;">
        <div style="background: #e9ecef; border-radius: 4px; height: 24px; overflow: hidden;">
          <div id="progress-bar" style="background: #007bff; height: 100%; width: 0%; transition: width 0.3s ease;"></div>
        </div>
      </div>
      <div style="display: flex; justify-content: space-between; align-items: center; margin-top: 0.5rem;">
        <div>
          <div id="progress-text" style="font-size: 0.9rem; color: #666;">Підготовка...</div>
          <div id="progress-count" style="font-size: 0.875rem; color: #999; margin-top: 0.25rem;">0 / 0</div>
        </div>
        <button id="cancel-batch" class="btn btn-secondary">□ Скасувати</button>
      </div>
    </section>

    <section class="card">
      <h2>□ Історія попередніх тестів</h2>
      <div id="results-empty" style="padding: 2rem; text-align: center; color: #999;">
        Ще немає збережених batch тестів. Запустіть перший тест вище.
      </div>
      <div id="results-table" hidden style="overflow-x: auto;">
        <table class="table table-striped">
          <thead>
            <tr>
              <th>Дата і час</th>
              <th>Категорії</th>
              <th>Зображень</th>
              <th>Тестів</th>
              <th>Статус</th>
              <th style="min-width: 240px;">Дії</th>
            </tr>
          </thead>
          <tbody id="results-body">
            <!-- Результати генеруються JavaScript -->
          </tbody>
        </table>
      </div>
    </section>
  </div>
  <script type="module" src="/batch-testing.js"></script>
 </body>
</html>
```

src/batch-testing.js

```
js
import { saveBatchResults, loadAllBatchResults, deleteBatchResult } from './modules/batchStorage.js';
import { convertImage } from './modules/imageConverter.js';
import { createNotification, clamp } from './modules/utils.js';

// Категорії розділені на статичні та анімації
const categoriesConfig = {
  static: {
    label: 'Статичні зображення',
    categories: [
      { id: 'photos/landscapes', label: 'Фотографії: Пейзажі' },
      { id: 'photos/portraits', label: 'Фотографії: Портрети' },
      { id: 'photos/products', label: 'Фотографії: Продукти' },
      { id: 'graphics/logos', label: 'Графіка: Логотипи' },
      { id: 'graphics/icons', label: 'Графіка: Іконки' },
    ],
    formats: [
      { name: 'PNG', type: 'png' },
      { name: 'JPG (90)', type: 'jpg', quality: 0.9 },
      { name: 'JPG (75)', type: 'jpg', quality: 0.75 },
      { name: 'WebP Lossy (90)', type: 'webp-lossy', quality: 0.9 },
      { name: 'WebP Lossy (75)', type: 'webp-lossy', quality: 0.75 },
      { name: 'WebP Lossless', type: 'webp-lossless' },
    ],
  },
  animated: {
    label: 'Анімації',
    categories: [
      { id: 'animations/gif', label: 'Анімації: GIF' },
      { id: 'animations/webp', label: 'Анімації: WebP' },
    ],
    formats: [
      { name: 'GIF (оригінал)', type: 'gif-original' },
      { name: 'WebP Animated (оригінал)', type: 'webp-animated-original' },
    ],
  },
};

let currentContentType = 'static';

let isTestingCancelled = false;
let lastProgressUpdate = 0;

document.addEventListener('DOMContentLoaded', init);

/**
 * Ініціалізація сторінки batch testing.
 */
function init() {
  updateContentType();
  bindContentTypeSwitch();
  bindControls();
  loadResultsTable();
}

function bindContentTypeSwitch() {
  document.querySelectorAll('input[name="content-type"]').forEach((radio) => {
    radio.addEventListener('change', updateContentType);
  });
}
```

```
function bindControls() {
  const selectAll = document.querySelector('[data-select-all]');
  const deselectAll = document.querySelector('[data-deselect-all]');
  selectAll?.addEventListener('click', () => toggleAllCategories(true));
  deselectAll?.addEventListener('click', () => toggleAllCategories(false));

  const startBtn = document.getElementById('start-batch');
  startBtn?.addEventListener('click', () => {
    void runBatch();
  });

  const cancelBtn = document.getElementById('cancel-batch');
  cancelBtn?.addEventListener('click', () => {
    isTestingCancelled = true;
    createNotification('Тестування скасовано', 'warning');
  });
}

function updateContentType() {
  const selectedType = document.querySelector('input[name="content-type"]:checked')?.value || 'static';
  currentContentType = selectedType;
  renderCategories();
  updateFormatsDisplay();
}

function updateFormatsDisplay() {
  const display = document.getElementById('formats-display');
  if (!display) return;
  const config = categoriesConfig[currentContentType];
  const formatsList = config.formats.map((f) => f.name).join(', ');
  display.innerHTML = `<strong>${formatsList}</strong>`;
}

function renderCategories() {
  const list = document.getElementById('category-list');
  const counter = document.getElementById('selected-counter');
  if (!list) return;
  const config = categoriesConfig[currentContentType];
  const cats = config.categories;
  list.innerHTML = cats
    .map(
      (cat) => `
      <label class="card" style="display:flex; gap:8px; align-items:center; padding: 0.75rem; cursor: pointer;">
        <input type="checkbox" name="category" value="${cat.id}" style="cursor:pointer;" />
        <span style="font-size:0.9rem;">${cat.label}</span>
      </label>
    `
    )
    .join('');
  list.onchange = () => updateCategoryCounter(counter, cats.length);
  updateCategoryCounter(counter, cats.length);
}

function toggleAllCategories(checked) {
  document.querySelectorAll('input[name="category"]').forEach((input) => {
    input.checked = checked;
  });
  const total = categoriesConfig[currentContentType]?.categories.length ?? 0;

  updateCategoryCounter(document.getElementById('selected-counter'), total);
}

function updateCategoryCounter(counter, total = 0) {
  if (!counter) return;
  const selected = [...document.querySelectorAll('input[name="category"]:checked')].length;
  counter.textContent = `Вибрано категорій: ${selected} / ${total}`;
}

/**
 * Основна функція запуску batch тесту.
 */
async function runBatch() {
  const selectedCategories = [...document.querySelectorAll('input[name="category"]:checked')].map(
    (input) => input.value
  );
  const selectedFormats = categoriesConfig[currentContentType]?.formats ?? [];
  const iterationsInput = document.getElementById('iterations');
  const delayInput = document.getElementById('delay');
  const iterations = clamp(Number(iterationsInput?.value) || 3, 1, 5);
  const delay = clamp(Number(delayInput?.value) || 1000, 500, 3000);

  if (!selectedCategories.length) {
    createNotification('Оберіть хоча б одну категорію', 'warning');
    return;
  }

  const progressSection = document.getElementById('progress-section');
  progressSection?.removeAttribute('hidden');
  isTestingCancelled = false;
  updateProgress(0, 'Підготовка до тестування...', '0 / 0');

  let images = [];
  try {
    images = await loadImagesFromCategories(selectedCategories);
  } catch (error) {
    createNotification(error.message, 'error');
    return;
  }

  if (!images.length) {
    updateProgress(0, 'Створіть metadata.json згідно з інструкцією', '0 / 0');
    createNotification('Створіть metadata.json згідно з інструкцією', 'warning');
```

```
    return;
  }

  const totalTests = images.length * selectedFormats.length * iterations;
  let completed = 0;
  const results = [];

  for (const image of images) {
    if (isTestingCancelled) break;
    let originalBlob;
    try {
      originalBlob = await loadImageBlob(image.path);
    } catch (error) {
      console.warn('Не вдалося завантажити зображення', image.path, error);
      continue;
    }

    for (const format of selectedFormats) {
      if (isTestingCancelled) break;
      let convertedBlob = originalBlob;
      if (format.type.includes('original')) {
        console.log(`Пропускаємо конвертацію для ${format.name} (оригінал)`);
      } else {
        try {
          convertedBlob = await convertImage(originalBlob, format.type, format.quality);
        } catch (error) {
          console.warn('Конвертація не вдалася', format, error);
          continue;
        }
      }
      for (let i = 0; i < iterations; i += 1) {
        if (isTestingCancelled) break;
        const statusText = `Тестування ${image.filename} → ${format.name} (${i + 1}/${iterations})`;
        const countText = `Виконано: ${completed} / ${totalTests} тестів`;
        updateProgress((completed / Math.max(1, totalTests)) * 100, statusText, countText);
        const metrics = await runSingleTest(convertedBlob, image, format, i);
        results.push({
          timestamp: new Date().toISOString(),
          imageId: image.id,
          category: image.category,
          format: format.name,
          quality: format.quality ?? null,
          iteration: i + 1,
          loadTime: metrics.loadTime,
          decodeTime: metrics.decodeTime,
          lcp: metrics.lcp,
          fcp: metrics.fcp,
          plt: metrics.plt,
          fileSize: convertedBlob.size,
          originalFileSize: originalBlob.size,
          compressionRatio: originalBlob.size > 0 ? convertedBlob.size / originalBlob.size : null,
          width: image.width,
          height: image.height,
        });
        completed += 1;
        await sleep(delay);
      }
    }
  }

  if (isTestingCancelled) {
    updateProgress(100, 'Тестування скасовано', '');
    return;
  }

  updateProgress(100, 'Тестування завершено!', `Виконано: ${completed} / ${totalTests} тестів`);
  const summary = calculateSummary(results);
  try {
    await saveBatchResults({
      contentType: currentContentType,
      config: { categories: selectedCategories, formats: selectedFormats, iterations, delay },
      results,
      summary,
      status: 'completed',
    });
    createNotification('Результати batch тесту збережено', 'success');
  } catch (error) {
    createNotification('Не вдалося зберегти результати', 'error');
  }
  await loadResultsTable();
}

/**
 * Завантажує список зображень з metadata.json для вибраних категорій.
 * @param {Array<string>} selectedCategories
 * @returns {Promise<Array>}
 */
async function loadImagesFromCategories(selectedCategories) {
  const images = [];
  for (const category of selectedCategories) {
    try {
      const response = await fetch(`/test-images/${category}/metadata.json`, { cache: 'no-store' });
      if (!response.ok) {
        console.warn(`metadata.json не знайдено для ${category}`);
        continue;
      }
      const metadata = await response.json();
      const items = metadata.images ?? [];
      for (const img of items) {
        images.push({
          id: img.id ?? `${category}-${img.filename}`,
          filename: img.filename,
          category,
          path: `/test-images/${category}/original/${img.filename}`,
          width: img.width ?? 0,
          height: img.height ?? 0,
          originalFormat: img.originalFormat ?? '',
          description: img.description ?? '',
          originalSize: img.size ?? 0,
        });
      }
    } catch (error) {
      console.warn(`Не вдалося прочитати metadata для ${category}`, error);
    }
  }
```

```
  return images;
}

/**
 * Завантажує зображення як Blob.
 * @param {string} path
 * @returns {Promise<Blob>}
 */
async function loadImageBlob(path) {
  const response = await fetch(path, { cache: 'no-store' });
  if (!response.ok) throw new Error(`Не вдалося завантажити
${path}`);
  return response.blob();
}

/**
 * Виконує один тест: завантажує зображення в DOM,
вимірює loadTime, decodeTime, LCP.
 * @param {Blob} blob
 * @param {object} imageInfo
 * @param {object} format
 * @param {number} iteration
 *                                                    @returns
{Promise<{loadTime:number|null,decodeTime:number|null,l
cp:number|null}>}
 */
async function runSingleTest(blob, imageInfo, format,
iteration) {
  return new Promise((resolve) => {
    const testWindow = window.open('/test-page.html',
'_blank', 'width=800,height=600');
    if (!testWindow) {
      console.error('Не вдалося відкрити тестове вікно');
      resolve({
        loadTime: null,
        decodeTime: null,
        lcp: null,
        fcp: null,
        plt: null,
      });
      return;
    }

    const timeout = setTimeout(() => {
      console.warn('Timeout: тест не завершився за 15
секунд');
      testWindow.close();
      resolve({
        loadTime: null,
        decodeTime: null,
        lcp: null,
        fcp: null,
        plt: null,
      });
    }, 15000);

    let testPageReady = false;

    const messageHandler = async (event) => {
      if (event.source !== testWindow) return;

      if (event.data.action === 'testPageReady') {
        testPageReady = true;
        const arrayBuffer = await blob.arrayBuffer();
        testWindow.postMessage(
          {
            action: 'runTest',
            blobData: arrayBuffer,
            imageInfo: {
              id: imageInfo.id,
              filename: imageInfo.filename,
              category: imageInfo.category,
              width: imageInfo.width,
              height: imageInfo.height,
              mimeType: blob.type,
            },
            format: {
              name: format.name,
              type: format.type,
              quality: format.quality,
            },
          },
          '*'
        );
      }

      if (event.data.action === 'testComplete') {
        clearTimeout(timeout);
        window.removeEventListener('message',
messageHandler);
        testWindow.close();
        resolve(event.data.metrics || { loadTime: null,
decodeTime: null, lcp: null, fcp: null, plt: null });
      }
    };

    window.addEventListener('message', messageHandler);

    const checkClosed = setInterval(() => {
      if (testWindow.closed) {
        clearInterval(checkClosed);
        clearTimeout(timeout);
        window.removeEventListener('message',
messageHandler);
        if (!testPageReady) {
          console.warn('Тестове вікно закрито передчасно');
        }
        resolve({
          loadTime: null,
          decodeTime: null,
          lcp: null,
          fcp: null,
          plt: null,
        });
      }
    }, 150);
  });
}

/**
 * Оновлює прогрес, з обмеженням частоти до ~100мс.
 * @param {number} percent
 * @param {string} text
 * @param {string} countText
 */
function updateProgress(percent, text, countText) {
  const now = performance.now();
  if (now - lastProgressUpdate < 100) return;
  lastProgressUpdate = now;
  const progressBar = document.getElementById('progress-
bar');
  const progressText = document.getElementById('progress-
text');
  const progressCount = document.getElementById('progress-
```

```
count');
  if (progressBar) progressBar.style.width = `${clamp(percent,
0, 100)}%`;
  if (progressText) progressText.textContent = text;
  if (progressCount) progressCount.textContent = countText;
}

/**
 * Підсумкова статистика для batch.
 * @param {Array} results
 *                                                    @returns
{{totalTests:number,categories:number,images:number,form
ats:number}}
 */
function calculateSummary(results) {
  const categoriesSet = new Set();
  const imagesSet = new Set();
  const formatsSet = new Set();
  results.forEach((r) => {
    categoriesSet.add(r.category);
    imagesSet.add(r.imageId);
    formatsSet.add(r.format);
  });
  return {
    totalTests: results.length,
    categories: categoriesSet.size,
    images: imagesSet.size,
    formats: formatsSet.size,
  };
}

/**
 * Експортує результати у CSV з BOM.
 * @param {Array} results
 * @param {string} filename
 */
function exportToCSV(results, filename = 'batch_results.csv')
{
  if (!results?.length) {
    createNotification('Немає даних для експорту', 'warning');
    return;
  }
  const header =
    [
      'timestamp',
      'imageId',
      'category',
      'format',
      'quality',
      'iteration',
      'loadTime_ms',
      'decodeTime_ms',
      'lcp_ms',
      'fcp_ms',
      'plt_ms',
      'fileSize_bytes',
      'originalSize_bytes',
      'compressionRatio',
      'width',
      'height',
    ].join(',') + '\n';
  const rows = results
    .map((r) =>
      [
        r.timestamp,
        r.imageId,
        r.category,
        r.format,
        r.quality ?? '',
        r.iteration,
        r.loadTime ?? '',
        r.decodeTime ?? '',
        r.lcp ?? '',
        r.fcp ?? '',
        r.plt ?? '',
        r.fileSize,
        r.originalFileSize ?? '',
        r.compressionRatio ? r.compressionRatio.toFixed(4) : '',
        r.width,
        r.height,
      ].join(',')
    )
    .join('\n');
  const csv = `\ufeff${header}${rows}`;
  const blob = new Blob([csv], { type: 'text/csv;charset=utf-8;'
});
  const link = document.createElement('a');
  link.href = URL.createObjectURL(blob);
  link.download = filename;
  link.style.display = 'none';
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
  URL.revokeObjectURL(link.href);
}

/**
 * Завантажує та відображає таблицю результатів.
 */
async function loadResultsTable() {
  const tbody = document.getElementById('results-body');
  const table = document.getElementById('results-table');
  const empty = document.getElementById('results-empty');
  try {
    const items = await loadAllBatchResults();
    const sorted = [...items].sort((a, b) => new
Date(b.timestamp) - new Date(a.timestamp));
    if (!sorted.length) {
      if (table) table.hidden = true;
      if (empty) empty.hidden = false;
      return;
    }
    if (table) table.hidden = false;
    if (empty) empty.hidden = true;
    if (tbody) {
      tbody.innerHTML = sorted
        .map((item) => {
          const categoriesCount = item.config?.categories?.length
?? 0;
          const images = item.summary?.images ?? '-';
          const tests = item.summary?.totalTests ?? '-';
          const status = item.status ?? '-';
          let statusIcon = '⊡';
          if (status === 'completed') statusIcon = '✓';
          else if (status === 'failed') statusIcon = '✗';
          return `
            <tr data-id="${item.id}">
              <td>${new
Date(item.timestamp).toLocaleString('uk-UA')}</td>
              <td>${categoriesCount} категорії</td>
              <td>${images}</td>
              <td>${tests}</td>
              <td>${statusIcon} ${status}</td>
              <td>
```

```
            <div class="history-actions">
              <button class="btn btn-ghost" data-view="${item.id}">Переглянути</button>
              <button class="btn btn-ghost" data-csv="${item.id}">Завантажити CSV</button>
              <button class="btn btn-ghost" data-delete="${item.id}">Видалити</button>
            </div>
          </td>
        </tr>
      `;
    })
    .join('');
  }
 } catch (error) {
  console.warn('Не вдалося завантажити результати batch', error);
  createNotification('Не вдалося завантажити результати batch', 'error');
 }
}

document.getElementById('results-body')?.addEventListener('click', async (event) => {
 const target = event.target;
 const row = target.closest('tr');
 const id = row?.dataset.id;
 if (!id) return;
 if (target.closest('[data-delete]')) {
  if (!confirm('Ви впевнені, що хочете видалити цей batch test?')) return;
  try {
   await deleteBatchResult(id);
   createNotification('Результат видалено', 'success');
   await loadResultsTable();
  } catch (error) {
   console.error('Помилка видалення:', error);
   createNotification('Не вдалося видалити результат', 'error');
  }
 } else if (target.closest('[data-csv]')) {
  try {
   const all = await loadAllBatchResults();
   const item = all.find((entry) => entry.id === id);
   if (!item || !item.results || item.results.length === 0) {
    createNotification('Немає даних для експорту', 'warning');
    return;
   }
   const filename = `batch_${id}_${new Date().toISOString().slice(0, 10)}.csv`;
   exportToCSV(item.results, filename);
   createNotification('CSV експортовано', 'success');
  } catch (error) {
   console.error('Помилка експорту CSV:', error);
   createNotification('Не вдалося експортувати CSV', 'error');
  }
 } else if (target.closest('[data-view]')) {
  window.location.href = `/statistics.html?batch=${id}`;
 }
});

/**
 * Допоміжна затримка.
 * @param {number} ms
 * @returns {Promise<void>}
 */
function sleep(ms) {
 return new Promise((resolve) => setTimeout(resolve, ms));
}
```

src/components/chart.js

```
js
import { createComparisonChart } from './charts/barChart.js';
import { createTimelineChart } from './charts/lineChart.js';
import { createSizeChart } from './charts/doughnutChart.js';
import { createRadarChart } from './charts/radarChart.js';

export const initCharts = () => {
 const section = document.querySelector('[data-charts]');
 const emptyState = section?.querySelector('[data-empty-charts]');
 const canvases = {
  comparison: document.getElementById('comparison-chart'),
  timeline: document.getElementById('timeline-chart'),
  size: document.getElementById('size-chart'),
  radar: document.getElementById('radar-chart'),
 };

 const wrappers = {
  comparison: document.querySelector('[data-chart-wrapper="comparison"]'),
  timeline: document.querySelector('[data-chart-wrapper="timeline"]'),
  size: document.querySelector('[data-chart-wrapper="size"]'),
  radar: document.querySelector('[data-chart-wrapper="radar"]'),
 };

 const instances = {};
 let hasRendered = false;

 const toggleEmptyState = (key, isEmpty) => {
  const wrapper = wrappers[key];
  const canvas = canvases[key];
  if (!wrapper || !canvas) return;
  const empty = wrapper.querySelector('.chart-empty');
  if (empty) empty.hidden = !isEmpty;
  canvas.hidden = isEmpty;
  if (isEmpty && instances[key]) {
   instances[key].destroy();
   delete instances[key];
  }
 };

 const render = (payloads) => {
  if (!payloads) return;
  renderComparison(payloads.comparison ?? { labels: [], values: [] });
  renderTimeline(payloads.timeline ?? { labels: [], values: [] });
  renderSize(payloads.size ?? { labels: [], values: [] });
  renderRadar(payloads.radar ?? { labels: [], datasets: [] });
  const hasData =
   Boolean(payloads?.comparison?.labels?.length) ||
   Boolean(payloads?.timeline?.labels?.length) ||
   Boolean(payloads?.size?.labels?.length) ||
   Boolean(payloads?.radar?.datasets?.length);
```

```
    if (emptyState) {
      emptyState.hidden = hasData;
    }
    if (!hasRendered && section) {
      section.removeAttribute('data-loading');
      hasRendered = true;
    }
  };

  const renderComparison = (data) => {
    const isEmpty = !data.labels?.length;
    toggleEmptyState('comparison', isEmpty);
    if (isEmpty || !canvases.comparison) return;
    instances.comparison?.destroy();
    instances.comparison =
createComparisonChart(canvases.comparison, data);
  };

  const renderTimeline = (data) => {
    const isEmpty = (data.labels?.length ?? 0) < 2;
    toggleEmptyState('timeline', isEmpty);
    if (isEmpty || !canvases.timeline) return;
    instances.timeline?.destroy();
    instances.timeline =
createTimelineChart(canvases.timeline, data);
  };

  const renderSize = (data) => {
    const isEmpty = !data.labels?.length;
    toggleEmptyState('size', isEmpty);
    if (isEmpty || !canvases.size) return;
    instances.size?.destroy();
    instances.size = createSizeChart(canvases.size, data);
  };

  const renderRadar = (data) => {
    const isEmpty = !data.datasets?.length;
    toggleEmptyState('radar', isEmpty);
    if (isEmpty || !canvases.radar) return;
    instances.radar?.destroy();
    instances.radar = createRadarChart(canvases.radar, data);
  };

  return { render };
};
```

src/components/charts/barChart.js

```
js
import { Chart } from './chartFactory.js';

export const createComparisonChart = (canvas, data) =>
  new Chart(canvas, {
    type: 'bar',
    data: {
      labels: data.labels,
      datasets: [
        {
          label: 'Час завантаження (мс)',
          data: data.values,
          backgroundColor: 'rgba(59, 130, 246, 0.85)',
          borderColor: 'rgb(59, 130, 246)',
          borderWidth: 2,
          borderRadius: 10,
          maxBarThickness: 42,
        },
      ],
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      plugins: {
        legend: { display: false },
        tooltip: {
          backgroundColor: 'rgba(17, 24, 39, 0.9)',
          padding: 12,
          cornerRadius: 8,
        },
      },
      scales: {
        x: {
          grid: { display: false },
          ticks: { color: '#6b7280' },
        },
        y: {
          beginAtZero: true,
          grid: { color: 'rgba(229, 231, 235, 0.6)' },
          ticks: { color: '#6b7280' },
        },
      },
    },
  });
```

src/components/charts/chartFactory.js

```
js
import { Chart } from 'chart.js/auto';

Chart.defaults.font.family = "Inter, -apple-system,
BlinkMacSystemFont, 'Segoe UI', sans-serif";
Chart.defaults.color = '#111827';
Chart.defaults.plugins.legend.labels = {
  color: '#6b7280',
};

export { Chart };
```

src/components/charts/doughnutChart.js

```
js
import { Chart } from './chartFactory.js';

export const createSizeChart = (canvas, data) =>
  new Chart(canvas, {
    type: 'doughnut',
    data: {
      labels: data.labels,
      datasets: [
        {
          label: 'Обсяг (MB)',
          data: data.values,
          backgroundColor: [
            'rgba(59, 130, 246, 0.85)',
            'rgba(99, 102, 241, 0.85)',
            'rgba(244, 114, 182, 0.85)',
            'rgba(16, 185, 129, 0.85)',
            'rgba(234, 179, 8, 0.85)',
          ],
```

```
        borderWidth: 0,
       },
      ],
    },
    options: {
     responsive: true,
     maintainAspectRatio: false,
     plugins: {
      legend: {
       position: 'bottom',
       labels: {
        usePointStyle: true,
       },
      },
     },
    },
  });
```

src/components/charts/lineChart.js

```
js
import { Chart } from './chartFactory.js';

export const createTimelineChart = (canvas, data) =>
  new Chart(canvas, {
    type: 'line',
    data: {
     labels: data.labels,
     datasets: [
      {
       label: 'Час завантаження (мс)',
       data: data.values,
       borderColor: 'rgb(16, 185, 129)',
       backgroundColor: 'rgba(16, 185, 129, 0.25)',
       fill: true,
       tension: 0.35,
       pointRadius: 4,
       pointBackgroundColor: '#ffffff',
       pointBorderColor: 'rgb(16, 185, 129)',
      },
     ],
    },
    options: {
     responsive: true,
     maintainAspectRatio: false,
     plugins: {
      legend: { display: false },
     },
     scales: {
      x: {
       grid: { color: 'rgba(243, 244, 246, 1)' },
       ticks: { color: '#6b7280' },
      },
      y: {
       beginAtZero: true,
       grid: { color: 'rgba(229, 231, 235, 0.6)' },
       ticks: { color: '#6b7280' },
      },
     },
    },
  });
```

src/components/charts/radarChart.js

```
js
import { Chart } from './chartFactory.js';

export const createRadarChart = (canvas, data) =>
  new Chart(canvas, {
    type: 'radar',
    data: {
     labels: data.labels,
     datasets: data.datasets,
    },
    options: {
     responsive: true,
     maintainAspectRatio: false,
     scales: {
      r: {
       beginAtZero: true,
       grid: { color: 'rgba(209, 213, 219, 0.5)' },
       angleLines: { color: 'rgba(209, 213, 219, 0.5)' },
       ticks: { display: false },
      },
     },
     plugins: {
      legend: {
       position: 'bottom',
      },
     },
    },
  });
```

src/components/toast.js

```
js
import { getIcon } from '../modules/icons.js';

const TOAST_TYPES = {
  info: 'toast-info',
  success: 'toast-success',
  warning: 'toast-warning',
  error: 'toast-error',
};

const ensureRoot = () => {
  let root = document.getElementById('toast-root');
  if (!root) {
    root = document.createElement('div');
    root.id = 'toast-root';
    root.className = 'toast-stack';
    document.body.append(root);
  }
  return root;
};

export const showToast = (message, type = 'info') => {
  const root = ensureRoot();
  const toast = document.createElement('div');
  toast.className = `toast ${TOAST_TYPES[type] ??
TOAST_TYPES.info} fade-in`;
  toast.setAttribute('role', 'status');
  toast.innerHTML = `
    <span class="toast-icon" aria-
hidden="true">${getIcon(type)}</span>
    <span class="toast-message">${message}</span>
    <button class="toast-close" type="button" aria-
label="Закрити сповіщення">&times;</button>
```

```
  `;
  toast.querySelector('.toast-close').addEventListener('click',
() => toast.remove());
  root.append(toast);
  setTimeout(() => {
    toast.style.opacity = '0';
    toast.style.transform = 'translateY(-6px)';
    setTimeout(() => toast.remove(), 300);
  }, 3500);
};
```

src/components/uploader.js

```
js
import { loadImageWithVariants } from
'../modules/imageLoader.js';
import { createNotification, isValidUrl, debounce } from
'../modules/utils.js';
import { icons } from '../modules/icons.js';

export const initUploader = ({ container, onResult, onProcess
}) => {
  if (!container) return null;
  enhanceIcons(container);
  const dropzone = container.querySelector('[data-dropzone]');
  const fileInput = container.querySelector('[data-file-input]');
  const urlInput = container.querySelector('[data-url-input]');
  const analyzeBtn = container.querySelector('[data-url-
submit]');

  const state = { loading: false };

  const setLoading = (loading, label = 'Опрацювання...') => {
    state.loading = loading;
    dropzone.dataset.loading = loading ? 'true' : 'false';
    analyzeBtn.disabled = loading ||
!isValidUrl(urlInput.value.trim());
    if (loading) {
      dropzone.classList.add('is-loading');
      dropzone.querySelector('[data-dropzone-
text]').textContent = label;
    } else {
      dropzone.classList.remove('is-loading');
      dropzone.querySelector('[data-dropzone-
text]').textContent = 'Перетягніть зображення або
натисніть';
    }
  };

  const handleFiles = async (files) => {
    if (!files?.length) return;
    setLoading(true, 'Генерую формати...');
    for (const file of files) {
      try {
        if (onProcess) {
          await onProcess(file);
        } else {
          const entry = await loadImageWithVariants(file);
          onResult?.(entry);
          createNotification(`Готово: ${file.name}`, 'success');
        }
      } catch (error) {
        createNotification(error.message, 'error');
      }
    }
    setLoading(false);
  };

  const handleUrl = async () => {
    const url = urlInput.value.trim();
    if (!isValidUrl(url)) {
      createNotification('Введіть коректний URL', 'warning');
      return;
    }
    setLoading(true, 'Завантажую з URL...');
    try {
      if (onProcess) {
        await onProcess(url);
      } else {
        const entry = await loadImageWithVariants(url);
        onResult?.(entry);
        createNotification('Аналіз завершено', 'success');
      }
    } catch (error) {
      createNotification(error.message, 'error');
    } finally {
      setLoading(false);
    }
  };

  const updateButton = debounce(() => {
    analyzeBtn.disabled = !isValidUrl(urlInput.value.trim()) ||
state.loading;
  }, 150);

  ['dragenter', 'dragover'].forEach((eventName) =>
    dropzone.addEventListener(eventName, (event) => {
      event.preventDefault();
      dropzone.classList.add('active');
    })
  );
  ['dragleave', 'drop'].forEach((eventName) =>
    dropzone.addEventListener(eventName, (event) => {
      event.preventDefault();
      if (eventName === 'drop') {
        const { files } = event.dataTransfer ?? {};
        if (files?.length) handleFiles(files);
      }
      dropzone.classList.remove('active');
    })
  );

  dropzone.addEventListener('click', () => fileInput.click());
  fileInput.addEventListener('change', (event) => {
    handleFiles(event.target.files);
    event.target.value = '';
  });

  urlInput.addEventListener('input', updateButton);
  analyzeBtn.addEventListener('click', handleUrl);

  updateButton();
  return { setLoading };
};

const enhanceIcons = (container) => {
  const dropIcon = container.querySelector('.dropzone-icon');
  if (dropIcon) dropIcon.innerHTML = icons.upload;
};
```

src/index.html

html

```
<!doctype html>
<html lang="uk">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Image Performance Analyzer</title>
  <link rel="icon" type="image/svg+xml" href="/icons/speed.svg" />
 </head>
 <body>
  <div class="page">
   <nav style="padding: 1rem; border-bottom: 1px solid #ddd; margin-bottom: 1rem;">
    <div class="container">
     <a href="/">Home</a> |
     <a href="/batch-testing.html">Batch Testing</a> |
     <a href="/statistics.html">Statistics</a> |
     <a href="/instructions.html">□ Інструкція</a>
    </div>
   </nav>

   <section
    class="info-banner"
    style="background: #f8f9fa; padding: 1.5rem; margin-bottom: 2rem; border-left: 4px solid #007bff;"
   >
    <h3>i Для дипломного дослідження</h3>
    <p>Цей застосунок призначений для статистичного дослідження продуктивності форматів зображень.</p>
    <ul>
     <li><strong>Швидке тестування:</strong> Завантажте зображення нижче для миттєвого аналізу</li>
     <li>
      <strong>Batch testing:</strong> Перейдіть на <a href="/batch-testing.html">Batch Testing</a> для автоматизованого тестування 100+ зображень
     </li>
     <li><strong>Статистика:</strong> Переглядайте результати на <a href="/statistics.html">Statistics</a></li>
    </ul>
   </section>

   <header class="section" aria-label="Intro">
    <h1 class="section-title">Image Performance Analyzer</h1>
    <p class="muted">Завантажуйте зображення, генеруйте альтернативні формати та вимірюйте їхню продуктивність.</p>
   </header>

   <section class="section" data-uploader aria-label="Завантаження">
    <h2 class="section-title">1. Завантаження</h2>
    <div class="grid two">
     <div class="dropzone" id="drop-zone" data-dropzone tabindex="0" style="transition: all 0.2s ease;">
      <div class="dropzone-icon" aria-hidden="true"></div>
      <p data-dropzone-text>Перетягніть зображення або натисніть</p>
      <p class="muted">Підтримка GIF/JPG/PNG/WebP/SWF до 30 МБ</p>
      <input
       type="file"
       id="file-input"
       data-file-input
       accept="image/gif,image/png,image/jpeg,image/webp,application/x-shockwave-flash"
       multiple
       hidden
      />
     </div>
     <div class="card">
      <label class="muted" for="url-input">Публічний URL</label>
      <input id="url-input" class="input" type="url" placeholder="https://cdn.example.com/image.webp" data-url-input />
      <div style="margin-top: 12px; display: flex; gap: 8px;">
       <button class="btn btn-primary" type="button" data-url-submit disabled>Аналізувати</button>
      </div>
     </div>
    </div>
    <div class="alert alert-info" style="margin-top: 1rem;">
     <strong>i Важливо:</strong> Для точного вимірювання LCP, FCP та PLT відкриватимуться тестові вікна. Дозвольте pop-up для цього сайту.
    </div>
    <div class="alert alert-info" style="margin-top: 1rem;">
     <strong>i Інформація:</strong>
     <ul style="margin: 0.5rem 0 0 1.5rem; font-size: 0.9rem;">
      <li>Зображення буде автоматично конвертовано у 6 форматів (PNG, JPG ×2, WebP ×3)</li>
      <li>Для кожного формату будуть виміряні метрики продуктивності</li>
      <li>Процес займає близько 30-60 секунд залежно від розміру зображення</li>
      <li>Будуть відкриватися тестові вікна - не закривайте їх вручну</li>
     </ul>
    </div>
   </section>

   <!-- Прогрес -->
   <section id="progress-section" class="card" style="display: none;">
    <h2>□ Обробка зображення</h2>
    <div style="margin: 1rem 0;">
     <div style="background: #e9ecef; border-radius: 4px; height: 24px; overflow: hidden;">
      <div id="progress-bar" style="background: #007bff; height: 100%; width: 0%; transition: width 0.3s ease;"></div>
     </div>
    </div>
    <div style="display: flex; justify-content: space-between; align-items: center; margin-top: 0.5rem;">
     <div>
      <div id="progress-text" style="font-size: 0.9rem; color: #666;">Підготовка...</div>
      <div id="progress-count" style="font-size: 0.875rem; color: #999; margin-top: 0.25rem;">0 / 0</div>
     </div>
    </div>
   </section>

   <section class="section" aria-label="Поточний
```

```
результат" id="results-section" style="display:none;">
      <div style="display: flex; justify-content: space-between; align-items: center; margin-bottom: 12px;">
       <h2 class="section-title">2. Поточне зображення</h2>
       <button class="btn btn-primary" type="button" data-test-current>Перевірити на сторінці</button>
      </div>
      <div class="grid one">
       <div class="card">
        <table class="variants-table" id="results">
         <thead>
          <tr>
           <th>Формат</th>
           <th>Якість</th>
           <th>Розмір</th>
           <th>Стиснення</th>
           <th>Load Time</th>
           <th>Decode Time</th>
           <th>LCP</th>
           <th>FCP</th>
           <th>PLT</th>
          </tr>
         </thead>
         <tbody data-variants-body></tbody>
        </table>
        <div class="muted" style="margin-top:12px; font-size:0.9rem;">
         LCP &lt; 2500 мс і FCP &lt; 1800 мс вважаються добрими показниками.
        </div>
        <div style="display:flex; gap:1rem; margin-top:1rem;">
         <button id="export-results" class="btn btn-secondary">□ Експортувати результати (CSV)</button>
         <button id="clear-results" class="btn btn-secondary">□ Очистити результати</button>
        </div>
       </div>
       <div class="card" data-preview>
        <img src="" alt="" style="width: 50%; border-radius: 8px; border: 1px solid var(--color-border);" />
        <p><strong data-name>-</strong></p>
        <p class="muted">Формат: <span data-format>-</span> · Розмір: <span data-size>-</span> · Роздільна здатність: <span data-dimensions>-</span></p>
       </div>
      </div>
     </section>

     <section class="section" aria-label="Історія">
      <div style="display: flex; justify-content: space-between; align-items: center; gap: 12px; flex-wrap: wrap;">
       <h2 class="section-title">3. Історія</h2>
       <div style="display: flex; gap: 8px; align-items: center;">
        <select class="input" data-history-filter>
         <option value="all">Усі формати</option>
         <option value="gif">GIF</option>
         <option value="jpg">JPG</option>
         <option value="png">PNG</option>
         <option value="webp">WEBP</option>
         <option value="swf">SWF</option>
        </select>
        <select class="input" data-history-sort>
         <option value="newest">Новіші</option>
         <option value="oldest">Старіші</option>
         <option value="size">Найменший розмір</option>
        </select>
        <button class="btn" type="button" data-history-clear>Очистити історію</button>
       </div>
      </div>
      <div class="card">
       <table class="table">
        <thead>
         <tr>
          <th>Thumbnail</th>
          <th>Name</th>
          <th>Original Format</th>
          <th>Date/Time</th>
          <th>Actions</th>
         </tr>
        </thead>
        <tbody data-history-body></tbody>
       </table>
      </div>
     </section>
    </div>
    <div id="toast-root" class="toast-stack" aria-live="polite" aria-atomic="true"></div>
    <script type="module" src="/main.js"></script>
   </body>
</html>
```

src/instructions.html

```
html
<!DOCTYPE html>
<html lang="uk">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Інструкція користувача - Image Performance Analyzer</title>
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.0/font/bootstrap-icons.css">
 <link rel="stylesheet" href="/styles/minimal-theme.css">
 <style>
  .step-card {
   margin-bottom: 2rem;
   border-left: 4px solid #007bff;
  }
  .step-number {
   width: 40px;
   height: 40px;
   background: #007bff;
   color: white;
   border-radius: 50%;
   display: flex;
   align-items: center;
   justify-content: center;
   font-weight: bold;
   font-size: 1.2rem;
  }
  .checklist-item {
   padding: 0.5rem;
```

```
  margin: 0.25rem 0;
  background: #f8f9fa;
  border-radius: 4px;
 }
 .checklist-item input[type="checkbox"] {
  margin-right: 0.5rem;
 }
 .code-block {
  background: #f4f4f4;
  border: 1px solid #ddd;
  border-radius: 4px;
  padding: 1rem;
  margin: 1rem 0;
  font-family: monospace;
  white-space: pre-wrap;
 }
 .alert-box {
  padding: 1rem;
  margin: 1rem 0;
  border-left: 4px solid;
  border-radius: 4px;
 }
 .alert-info {
  background: #e7f3ff;
  border-color: #2196F3;
 }
 .alert-warning {
  background: #fff4e5;
  border-color: #ff9800;
 }
 .alert-success {
  background: #e8f5e9;
  border-color: #4caf50;
 }
 .category-table {
  width: 100%;
  border-collapse: collapse;
  margin: 1rem 0;
 }
 .category-table th,
 .category-table td {
  padding: 0.75rem;
  border: 1px solid #ddd;
  text-align: left;
 }
 .category-table th {
  background: #f8f9fa;
  font-weight: 600;
 }
 .folder-tree {
  font-family: monospace;
  background: #f8f9fa;
  padding: 1rem;
  border-radius: 4px;
  margin: 1rem 0;
 }
 .folder-tree .folder {
  color: #ff9800;
  font-weight: bold;
 }
 .folder-tree .file {
  color: #2196F3;
 }
 .progress-indicator {
  display: flex;
  justify-content: space-between;
  margin: 2rem 0;
  position: relative;
 }
 .progress-indicator::before {
  content: '';
  position: absolute;
  top: 20px;
  left: 0;
  right: 0;
  height: 2px;
  background: #ddd;
  z-index: 0;
 }
 .progress-step {
  display: flex;
  flex-direction: column;
  align-items: center;
  position: relative;
  z-index: 1;
 }
 .progress-step-circle {
  width: 40px;
  height: 40px;
  border-radius: 50%;
  background: white;
  border: 2px solid #ddd;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: bold;
  margin-bottom: 0.5rem;
 }
 .progress-step.active .progress-step-circle {
  background: #007bff;
  color: white;
  border-color: #007bff;
 }
 .progress-step-label {
  font-size: 0.875rem;
  text-align: center;
 }
 </style>
</head>
<body>
 <nav style="padding: 1rem; border-bottom: 1px solid #ddd;">
  <div class="container">
   <a href="/">Home</a> |
   <a href="/batch-testing.html">Batch Testing</a> |
   <a href="/statistics.html">Statistics</a> |
   <strong> Інструкція</strong>
  </div>
 </nav>

 <div class="container py-5">
  <div class="text-center mb-5">
   <h1 class="display-4">📖 Інструкція користувача</h1>
   <p class="lead text-muted">Покрокове керівництво для проведення дослідження</p>
  </div>

  <div class="progress-indicator">
   <div class="progress-step active">
    <div class="progress-step-circle">1</div>
    <div class="progress-step-label">Збір зображень</div>
   </div>
   <div class="progress-step">
    <div class="progress-step-circle">2</div>
```

```
      <div class="progress-step-label">Організація файлів</div>
    </div>
    <div class="progress-step">
      <div class="progress-step-circle">3</div>
      <div class="progress-step-label">Metadata.json</div>
    </div>
    <div class="progress-step">
      <div class="progress-step-circle">4</div>
      <div class="progress-step-label">Тестування</div>
    </div>
    <div class="progress-step">
      <div class="progress-step-circle">5</div>
      <div class="progress-step-label">Аналіз</div>
    </div>
  </div>

  <div class="alert-box alert-info">
    <h4><i class="bi bi-info-circle"></i> Мета дослідження</h4>
    <p class="mb-0">Для проведення статистично валідного дослідження потрібно зібрати <strong>мінімум 80 зображень</strong> різних категорій. Кожне зображення буде протестовано у <strong>6 форматах</strong> з <strong>3 ітераціями</strong>, загалом <strong>1440 тестів</strong>.</p>
  </div>

  <div class="card step-card">
    <div class="card-body">
      <div class="d-flex align-items-center mb-3">
        <div class="step-number me-3">1</div>
        <h2 class="mb-0">Збір зображень</h2>
      </div>

      <p>Зберіть <strong>80 зображень</strong> за такими категоріями:</p>

      <table class="category-table">
        <thead>
          <tr>
            <th>Категорія</th>
            <th>Кількість</th>
            <th>Де взяти</th>
            <th>Примітки</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td><strong>Фотографії: Пейзажі</strong></td>
            <td>10</td>
            <td><a href="https://unsplash.com/s/photos/landscape" target="_blank">Unsplash <i class="bi bi-box-arrow-up-right"></i></a></td>
            <td>Гори, ліси, моря</td>
          </tr>
          <tr>
            <td><strong>Фотографії: Портрети</strong></td>
            <td>10</td>
            <td><a href="https://unsplash.com/s/photos/portrait" target="_blank">Unsplash <i class="bi bi-box-arrow-up-right"></i></a></td>
            <td>Різні ракурси</td>
          </tr>
          <tr>
            <td><strong>Фотографії: Продукти</strong></td>
            <td>10</td>
            <td><a href="https://unsplash.com/s/photos/product" target="_blank">Unsplash <i class="bi bi-box-arrow-up-right"></i></a></td>
            <td>Електроніка, їжа, одяг</td>
          </tr>
          <tr>
            <td><strong>Графіка: Логотипи</strong></td>
            <td>10</td>
            <td><a href="https://www.flaticon.com/" target="_blank">Flaticon <i class="bi bi-box-arrow-up-right"></i></a></td>
            <td>PNG з прозорістю, ~500x500px</td>
          </tr>
          <tr>
            <td><strong>Графіка: Іконки</strong></td>
            <td>10</td>
            <td><a href="https://www.flaticon.com/" target="_blank">Flaticon <i class="bi bi-box-arrow-up-right"></i></a></td>
            <td>256x256 або 512x512px</td>
          </tr>
          <tr>
            <td><strong>Анімації: GIF</strong></td>
            <td>15</td>
            <td><a href="https://giphy.com/" target="_blank">GIPHY <i class="bi bi-box-arrow-up-right"></i></a></td>
            <td>Розмір до 2MB</td>
          </tr>
          <tr>
            <td><strong>Анімації: WebP</strong></td>
            <td>15</td>
            <td>-</td>
            <td>Можна пропустити - програма конвертує GIF</td>
          </tr>
        </tbody>
        <tfoot>
          <tr>
            <th>РАЗОМ:</th>
            <th>80</th>
            <th colspan="2"></th>
          </tr>
        </tfoot>
      </table>

      <div class="alert-box alert-warning mt-4">
        <strong><i class="bi bi-exclamation-triangle"></i> Важливо:</strong>
        <ul class="mb-0 mt-2">
          <li>Завантажуйте зображення у <strong>високій якості</strong> (Full resolution)</li>
          <li>Уникайте дублікатів - кожне зображення має бути унікальним</li>
          <li>Розмір файлів: від 100 KB до 5 MB</li>
        </ul>
      </div>

      <div class="mt-4">
        <h5>Чеклист збору зображень:</h5>
        <div class="checklist-item">
          <input type="checkbox" id="check-landscapes">
          <label for="check-landscapes">✓ Завантажено 10 пейзажів</label>
        </div>
```

```
      <div class="checklist-item">
        <input type="checkbox" id="check-portraits">
        <label for="check-portraits">✓ Завантажено 10 портретів</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="check-products">
        <label for="check-products">✓ Завантажено 10 продуктів</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="check-logos">
        <label for="check-logos">✓ Завантажено 10 логотипів (PNG)</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="check-icons">
        <label for="check-icons">✓ Завантажено 10 іконок</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="check-gifs">
        <label for="check-gifs">✓ Завантажено 15 GIF анімацій</label>
      </div>
    </div>
  </div>

  <div class="card" style="margin-top: 2rem;">
    <div class="card-body">
      <h2>□ Розуміння метрик</h2>
      <p class="text-muted">Детальне пояснення кожної метрики та як їх інтерпретувати для диплому.</p>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ Load Time (час завантаження)</summary>
        <div style="margin-top: 1rem; padding-left: 1rem;">
          <p><strong>Що це:</strong> Час від початку HTTP-запиту до завершення завантаження файлу.</p>
          <p><strong>Як вимірюємо:</strong> Resource Timing API.</p>
          <p><strong>Що впливає:</strong> розмір файлу, швидкість мережі, стиснення.</p>
          <p><strong>Орієнтири:</strong> □ &lt; 100 мс; □ 100–500 мс; □ 500–1000 мс; □ &gt; 1000 мс.</p>
          <p><strong>Для диплому:</strong> “WebP Lossy (90%) показав Load Time 98 мс, що на 60% швидше за PNG (245 мс).”</p>
        </div>
      </details>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ Decode Time (час декодування)</summary>
        <div style="margin-top: 1rem; padding-left: 1rem;">
          <p><strong>Що це:</strong> Час, потрібний браузеру для розпаковки зображення в bitmap.</p>
          <p><strong>Як вимірюємо:</strong> <code>img.decode()</code>.</p>
          <p><strong>Орієнтири:</strong> □ &lt; 50 мс; □ 50–150 мс; □ 150–300 мс; □ &gt; 300 мс.</p>
        </div>
      </details>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ LCP (Largest Contentful Paint)</summary>
        <div style="margin-top: 1rem; padding-left: 1rem;">
          <p><strong>Що це:</strong> Час появи найбільшого елемента (головне зображення).</p>
          <p><strong>Web Vitals:</strong> □ &lt; 2.5 с; □ 2.5–4.0 с; □ &gt; 4.0 с.</p>
          <div class="alert alert-info" style="margin-top: 1rem;">
            <strong>Порада:</strong> Ми вимірюємо LCP на окремій test-page з одним зображенням, тому результат “чистий”.
          </div>
        </div>
      </details>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ FCP (First Contentful Paint)</summary>
        <div style="margin-top: 1rem; padding-left: 1rem;">
          <p><strong>Що це:</strong> Час до появи першого контенту.</p>
          <p><strong>Web Vitals:</strong> □ &lt; 1.8 с; □ 1.8–3.0 с; □ &gt; 3.0 с.</p>
          <p><strong>Різниця з LCP:</strong> FCP - перший контент, LCP - найбільший/основний.</p>
        </div>
      </details>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ PLT (Page Load Time)</summary>
        <div style="margin-top: 1rem; padding-left: 1rem;">
          <p><strong>Що це:</strong> Загальний час завантаження тест-сторінки.</p>
          <p><strong>Орієнтири для нашої сторінки:</strong> □ &lt; 500 мс; □ 500–1500 мс; □ 1500–3000 мс; □ &gt; 3000 мс.</p>
        </div>
      </details>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ File Size</summary>
        <div style="margin-top: 1rem; padding-left: 1rem;">
          <p><strong>Що це:</strong> Розмір файлу після конвертації (B/KB/MB).</p>
          <p><strong>Приклад:</strong> PNG 5.2 MB → WebP 1.1 MB (−78.8%).</p>
        </div>
      </details>

      <details style="margin: 1rem 0; padding: 1rem; border: 1px solid #e9ecef; border-radius: 4px;">
        <summary style="cursor: pointer; font-weight: 600; font-size: 1.1rem;">□ Compression Ratio</summary>
```

```
        <div style="margin-top: 1rem; padding-left: 1rem;">
         <p><strong>Що це:</strong> Зміна розміру відносно оригіналу.</p>
         <p><strong>Діапазони:</strong> −80%+ відмінно; −60…−80% добре; −40…−60% помірно; −20…−40% слабко; 0…+50% погано.</p>
        </div>
       </details>
     </div>
   </div>

   <div class="card" style="margin-top: 2rem;">
    <div class="card-body">
     <h2>□ Статистична валідність</h2>

      <h3 style="margin-top: 1.5rem;">Чому 3 ітерації?</h3>
      <p>Одне вимірювання може бути неточним (фонові процеси, мережа). Кожне зображення тестується <strong>3 рази</strong>, беремо <strong>медіану</strong> (стійка до викидів).</p>

      <h3 style="margin-top: 1.5rem;">Тест Вілкоксона</h3>
      <p>Відповідає на питання: <strong>"Чи є різниця між форматами статистично значущою?"</strong></p>

      <div style="background: #f8f9fa; padding: 1.5rem; border-radius: 4px; margin: 1rem 0;">
       <h4>Як інтерпретувати:</h4>
       <p><strong>p-value:</strong></p>
       <ul>
         <li>p &lt; 0.05 - різниця значуща ✓</li>
         <li>p &gt; 0.05 - різниця може бути випадковою ✗</li>
       </ul>
       <p><strong>Effect Size (r):</strong></p>
       <ul>
        <li>r &lt; 0.3 - малий ефект</li>
        <li>0.3 ≤ r &lt; 0.5 - середній</li>
        <li>r ≥ 0.5 - великий ефект</li>
       </ul>
      </div>

      <div class="alert alert-success" style="margin-top: 1rem;">
       <strong>✓Для диплому пишіть:</strong><br />
       "Тест Вілкоксона показав статистично значущу різницю (p &lt; 0.001) у Load Time між PNG та WebP. Effect Size r = 0.89 свідчить про дуже великий ефект, що підтверджує практичну значущість покращення продуктивності."
      </div>
     </div>
   </div>

   <div class="card step-card">
    <div class="card-body">
     <div class="d-flex align-items-center mb-3">
      <div class="step-number me-3">2</div>
      <h2 class="mb-0">Організація файлів</h2>
     </div>

     <p>Розмістіть зображення у папці <code>/public/test-images/</code> за такою структурою:</p>

     <div class="folder-tree">
      <pre style="margin:0;">
<span class="folder">public/test-images/</span>
├── <span class="folder">photos/</span>
│   ├── <span class="folder">landscapes/</span>
│   │   ├── <span class="folder">original/</span>
│   │   │   ├── <span class="file">landscape_001.jpg</span>
│   │   │   ├── <span class="file">landscape_002.jpg</span>
│   │   │   └── ... (10 файлів)
│   │   └── <span class="file">metadata.json</span>
│   ├── <span class="folder">portraits/</span>
│   │   ├── <span class="folder">original/</span>
│   │   └── <span class="file">metadata.json</span>
│   └── <span class="folder">products/</span>
│       ├── <span class="folder">original/</span>
│       └── <span class="file">metadata.json</span>
├── <span class="folder">graphics/</span>
│   ├── <span class="folder">logos/</span>
│   └── <span class="folder">icons/</span>
└── <span class="folder">animations/</span>
    ├── <span class="folder">gif/</span>
    └── <span class="folder">webp/</span>
      </pre>
     </div>

     <div class="alert-box alert-info mt-4">
      <strong><i class="bi bi-lightbulb"></i> Порада:</strong> Перейменовуйте файли за схемою <code>категорія_номер.розширення</code>:
      <ul class="mb-0 mt-2">
       <li><code>landscape_001.jpg</code>, <code>landscape_002.jpg</code>, ...</li>
       <li><code>portrait_001.jpg</code>, <code>portrait_002.jpg</code>, ...</li>
       <li><code>logo_001.png</code>, <code>logo_002.png</code>, ...</li>
      </ul>
     </div>

     <div class="mt-4">
      <h5>Чеклист організації файлів:</h5>
      <div class="checklist-item">
       <input type="checkbox" id="check-folders">
       <label for="check-folders">✓ Створено всі папки за структурою</label>
      </div>
      <div class="checklist-item">
       <input type="checkbox" id="check-renamed">
       <label for="check-renamed">✓ Файли перейменовано за схемою category_NNN</label>
      </div>
      <div class="checklist-item">
       <input type="checkbox" id="check-placed">
       <label for="check-placed">✓ Всі файли розміщено у відповідних /original/ папках</label>
      </div>
     </div>
    </div>
   </div>

   <div class="card step-card">
    <div class="card-body">
     <div class="d-flex align-items-center mb-3">
      <div class="step-number me-3">3</div>
      <h2 class="mb-0">Створення metadata.json</h2>
     </div>

     <p>Для <strong>КОЖНОЇ</strong> з 7 підпапок
```

створіть файл <code>metadata.json</code>.

<h5 class="mt-4">Приклад для <code>/public/test-images/photos/landscapes/metadata.json</code>:</h5>

{
"category": "photos/landscapes",
"description": "Пейзажні фотографії для тестування",
"images": [
{
"id": "landscape_001",
"filename": "landscape_001.jpg",
"description": "Гірський пейзаж зі снігом",
"width": 1920,
"height": 1080,
"originalFormat": "jpg",
"notes": "Висока деталізація"
},
{
"id": "landscape_002",
"filename": "landscape_002.jpg",
"description": "Морський берег на заході сонця",
"width": 1920,
"height": 1280,
"originalFormat": "jpg",
"notes": "Багато градієнтів"
}
]
}

<h5 class="mt-4">Як дізнатися розміри зображення (width × height)?</h5>

Windows: Правий клік → Властивості → Деталі
Mac: Правий клік → Get Info
Альтернатива: Відкрийте зображення в будь-якому редакторі

 Важливо:
<ul class="mb-0 mt-2">
Файл має бути валідним JSON (перевірте на jsonlint.com)
<code>filename</code> має точно відповідати назві файлу в папці <code>original/</code>
<code>category</code> має відповідати шляху (наприклад, "photos/landscapes")

<h5>Чеклист metadata.json:</h5>

<input type="checkbox" id="meta-landscapes">
<label for="meta-landscapes">✓ photos/landscapes/metadata.json</label>

<input type="checkbox" id="meta-portraits">
<label for="meta-portraits">✓ photos/portraits/metadata.json</label>

<input type="checkbox" id="meta-products">
<label for="meta-products">✓ photos/products/metadata.json</label>

<input type="checkbox" id="meta-logos">
<label for="meta-logos">✓ graphics/logos/metadata.json</label>

<input type="checkbox" id="meta-icons">
<label for="meta-icons">✓ graphics/icons/metadata.json</label>

<input type="checkbox" id="meta-gif">
<label for="meta-gif">✓ animations/gif/metadata.json</label>

<input type="checkbox" id="meta-webp">
<label for="meta-webp">✓ animations/webp/metadata.json</label>

4
<h2 class="mb-0">Запуск batch тестування</h2>

<h5>4.1. Тестовий запуск (рекомендовано)</h5>
Спочатку протестуйте на 3-5 зображеннях, щоб переконатися, що все працює:

Створіть тільки одну категорію (наприклад, landscapes) з 3 файлами
Відкрийте Batch Testing
Виберіть тільки цю категорію
Виберіть 2-3 формати (PNG, JPG 90, WebP Lossy 90)
Встановіть iterations: 1
Натисніть "Почати тестування"
Зачекайте ~2-3 хвилини
Перевірте результати на Statistics

 Якщо тест пройшов успішно: переходьте до повного тестування

<h5 class="mt-4">4.2. Повне тестування</h5>

Відкрийте Batch Testing
Натисніть "Вибрати всі" (7

```
категорій)</li>
        <li>Виберіть формати:
          <ul>
            <li>☑ PNG</li>
            <li>☑ JPG (quality: 90)</li>
            <li>☑ JPG (quality: 75)</li>
            <li>☑ WebP Lossy (quality: 90)</li>
            <li>☑ WebP Lossy (quality: 75)</li>
            <li>☑ WebP Lossless</li>
          </ul>
        </li>
        <li>Встановіть <strong>iterations: 3</strong> (для статистичної валідності)</li>
        <li>Затримка: <strong>1000 мс</strong></li>
        <li>Натисніть <strong>"Почати тестування"</strong></li>
        <li><strong>Зачекайте 2-4 години</strong> (80 зображень × 6 форматів × 3 ітерації = 1440 тестів)</li>
      </ol>

      <div class="alert-box alert-warning mt-4">
        <strong><i class="bi bi-hourglass-split"></i> Під час тестування:</strong>
        <ul class="mb-0 mt-2">
          <li>Не закривайте браузер</li>
          <li>Не переходьте на інші сторінки</li>
          <li>Можна мінімізувати вікно браузера</li>
          <li>Progress bar показує поточний прогрес</li>
        </ul>
      </div>

      <div class="mt-4">
        <h5>Чеклист тестування:</h5>
        <div class="checklist-item">
          <input type="checkbox" id="test-trial">
          <label for="test-trial">✓ Тестовий запуск виконано успішно</label>
        </div>
        <div class="checklist-item">
          <input type="checkbox" id="test-full">
          <label for="test-full">✓ Повне тестування запущено</label>
        </div>
        <div class="checklist-item">
          <input type="checkbox" id="test-complete">
          <label for="test-complete">✓ Тестування завершено (1440 тестів)</label>
        </div>
      </div>
    </div>
  </div>

  <div class="card step-card">
    <div class="card-body">
      <div class="d-flex align-items-center mb-3">
        <div class="step-number me-3">5</div>
        <h2 class="mb-0">Аналіз результатів</h2>
      </div>

      <h5>5.1. Перегляд статистики</h5>
      <ol>
        <li>Відкрийте <a href="/statistics.html">Statistics</a></li>
        <li>Виберіть ваш batch test з dropdown (дата + кількість тестів)</li>
        <li>Переглядайте:
          <ul>
            <li><strong>Описову статистику:</strong> таблиці з mean, median, stdDev для кожного формату</li>
            <li><strong>Графіки:</strong> bar charts, scatter plots, heatmap</li>
            <li><strong>Матрицю кореляцій:</strong> зв'язок між метриками</li>
          </ul>
        </li>
      </ol>

      <h5 class="mt-4">5.2. Експорт даних для диплому</h5>
      <ol>
        <li>Натисніть <strong>"Завантажити CSV"</strong></li>
        <li>Відкрийте файл у Excel/Google Sheets</li>
        <li>Зробіть скріншоти графіків (PrintScreen або Snipping Tool)</li>
        <li>Скопіюйте таблиці описової статистики в Word</li>
      </ol>

      <div class="alert-box alert-success mt-4">
        <strong><i class="bi bi-check-circle"></i> Що використати в дипломі:</strong>
        <ul class="mb-0 mt-2">
          <li><strong>Таблиці:</strong> Описова статистика (mean, median, stdDev)</li>
          <li><strong>Графіки:</strong> Bar chart (Load Time), Scatter plot (Size vs Time)</li>
          <li><strong>Статистика:</strong> p-value, Effect Size з тесту Вілкоксона</li>
          <li><strong>CSV:</strong> Для додаткового аналізу в R/Python</li>
        </ul>
      </div>

      <h5 class="mt-4">5.3. Інтерпретація результатів</h5>
      <p><strong>Приклад висновку для диплому:</strong></p>
      <div class="code-block">"WebP Lossy (quality 90) показав статистично значуще (p &amp;lt; 0.001)
покращення Load Time на 60.2% порівняно з PNG при великому
розмірі ефекту (r = 0.84 за Cohen). File Size зменшився на 76.8% при незначній втраті якості. Результати підтверджують
доцільність використання WebP для веб-додатків."</div>

      <div class="mt-4">
        <h5>Чеклист для диплому:</h5>
        <div class="checklist-item">
          <input type="checkbox" id="diploma-csv">
          <label for="diploma-csv">✓ CSV експортовано</label>
        </div>
        <div class="checklist-item">
          <input type="checkbox" id="diploma-screenshots">
          <label for="diploma-screenshots">✓ Скріншоти графіків зроблено</label>
        </div>
        <div class="checklist-item">
          <input type="checkbox" id="diploma-tables">
          <label for="diploma-tables">✓ Таблиці скопійовано в Word</label>
        </div>
        <div class="checklist-item">
```

```
        <input type="checkbox" id="diploma-pvalue">
        <label for="diploma-pvalue">✓ p-value &lt; 0.05 підтверджено</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="diploma-effect">
        <label for="diploma-effect">✓ Effect Size інтерпретовано</label>
      </div>
    </div>
  </div>

  <div class="card mb-5" style="border-left: 4px solid #ff9800;">
    <div class="card-body">
      <h3><i class="bi bi-exclamation-triangle text-warning"></i> Поширені помилки та рішення</h3>

      <div class="mt-4">
        <h5>❌"Failed to load metadata.json"</h5>
        <p><strong>Рішення:</strong> Переконайтеся, що файл <code>metadata.json</code> існує в кожній підпапці і валідний (перевірте на <a href="https://jsonlint.com/" target="_blank">jsonlint.com</a>)</p>
      </div>

      <div class="mt-4">
        <h5>❌"Test page not loading"</h5>
        <p><strong>Рішення:</strong> Переконайтеся, що dev server запущений (<code>npm run dev</code>)</p>
      </div>

      <div class="mt-4">
        <h5>❌"No data in Statistics"</h5>
        <p><strong>Рішення:</strong> Спочатку запустіть Batch Testing, результати автоматично з'являться</p>
      </div>

      <div class="mt-4">
        <h5>❌ "CSV відкривається з кракозябрами"</h5>
        <p><strong>Рішення:</strong> Файл містить BOM. У Excel: Data → From Text/CSV → Encoding: UTF-8</p>
      </div>
    </div>
  </div>

  <div class="card" style="border-left: 4px solid #4caf50;">
    <div class="card-body">
      <h3><i class="bi bi-clipboard-check text-success"></i> Фінальний чеклист</h3>
      <p class="text-muted">Перевірте перед здачею диплому:</p>

      <div class="checklist-item">
        <input type="checkbox" id="final-images">
        <label for="final-images">✓ Зібрано 80 зображень у 7 категоріях</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-metadata">
        <label for="final-metadata">✓ Створено 7 файлів metadata.json</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-batch">
        <label for="final-batch">✓ Batch testing виконано (1440 тестів)</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-statistics">
        <label for="final-statistics">✓ Результати переглянуто на Statistics</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-export">
        <label for="final-export">✓ CSV експортовано та перевірено</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-graphs">
        <label for="final-graphs">✓ Скріншоти графіків для диплому готові</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-pvalue">
        <label for="final-pvalue">✓ p-value &lt; 0.05 підтверджено для ключових порівнянь</label>
      </div>
      <div class="checklist-item">
        <input type="checkbox" id="final-readme">
        <label for="final-readme">✓ README.md прочитано</label>
      </div>
    </div>
  </div>

  <div class="text-center mt-5 mb-4">
    <p class="text-muted">Якщо виникли питання - перевірте консоль браузера (F12 → Console) або зверніться до розробника</p>
    <p><strong>Успіхів у дослідженні! □</strong></p>
  </div>
 </div>

 <script>
  document.querySelectorAll('.checklist-item input[type="checkbox"]').forEach((checkbox) => {
    const savedState = localStorage.getItem(checkbox.id);
    if (savedState === 'true') {
      checkbox.checked = true;
    }
    checkbox.addEventListener('change', (e) => {
      localStorage.setItem(e.target.id, e.target.checked);
    });
  });
 </script>
</body>
</html>
```

src/main.js

```
js
import 'bootstrap/dist/css/bootstrap-grid.min.css';
import './styles/minimal-theme.css';
import './styles/main.css';
import './styles/components.css';

import { initUploader } from './components/uploader.js';
import {
  addHistoryEntry,
```

```
  loadHistory,
  removeHistoryEntry,
  clearHistory,
  sortHistory,
  filterHistory,
  saveHistory,
  getVariantBlob,
} from './modules/history.js';
import { formatBytes, formatDuration, createNotification, blobToDataUrl } from './modules/utils.js';
import { openTestPageWindow } from './modules/testPage.js';
import { convertImage } from './modules/imageConverter.js';

const state = {
  entries: [],
  selectedEntryId: null,
  testResults: [],
};

document.addEventListener('DOMContentLoaded', bootstrap);

async function bootstrap() {
  state.entries = await loadHistory();
  state.selectedEntryId = state.entries[0]?.id ?? null;

  initUploader({
    container: document.querySelector('[data-uploader]'),
    onProcess: handleImageUpload,
  });

  bindFilters();
  bindHistoryActions();
  bindTestButton();
  bindResultActions();
  window.addEventListener('message', handleTestMessage);
  render();
}

const elements = {
  preview: document.querySelector('[data-preview]'),
  name: document.querySelector('[data-name]'),
  format: document.querySelector('[data-format]'),
  size: document.querySelector('[data-size]'),
  dimensions: document.querySelector('[data-dimensions]'),
  variantsBody: document.querySelector('[data-variants-body]'),
  testButton: document.querySelector('[data-test-current]'),
  historyBody: document.querySelector('[data-history-body]'),
  historyFilter: document.querySelector('[data-history-filter]'),
  historySort: document.querySelector('[data-history-sort]'),
  clearHistoryBtn: document.querySelector('[data-history-clear]'),
};

function render() {
  const filtered = filterHistory(state.entries, { format: elements.historyFilter?.value ?? 'all' });
  const sorted = sortHistory(filtered, elements.historySort?.value ?? 'newest');
  renderHistory(sorted);
  const active = state.entries.find((item) => item.id === state.selectedEntryId) ?? sorted[0];
  state.selectedEntryId = active?.id ?? null;
  renderCurrent(active);
}

function renderCurrent(entry) {
  if (!entry) {
    elements.preview?.setAttribute('hidden', 'true');
    elements.variantsBody.innerHTML = '<tr><td colspan="9">Додайте зображення, щоб побачити результати</td></tr>';
    return;
  }
  const resultsSection = document.getElementById('results-section');
  if (resultsSection) resultsSection.style.display = 'block';
  elements.preview?.removeAttribute('hidden');
  const previewImg = elements.preview.querySelector('img');
  if (previewImg && entry.previewUrl) {
    previewImg.src = entry.previewUrl;
    previewImg.alt = entry.originalName;
  }
  elements.name.textContent = entry.originalName;
  elements.format.textContent = entry.originalFormat.toUpperCase();
  elements.size.textContent = formatBytes(entry.originalSize);
  elements.dimensions.textContent = `${entry.width} × ${entry.height}`;
  const formatTime = (value) => (value === null || value === undefined ? '-' : formatDuration(value));
  const formatQuality = (q) => (typeof q === 'number' ? `${Math.round(q * 100)}%` : '-');
  const formatCompression = (ratio) => {
    if (ratio === null || ratio === undefined) return '-';
    const percent = (1 - ratio) * 100;
    if (percent > 0) return `−${percent.toFixed(1)}%`;
    if (percent < 0) return `+${Math.abs(percent).toFixed(1)}%`;
    return '0%';
  };
  const vitalsClass = (metric, value) => {
    if (value === null || value === undefined) return '';
    if (metric === 'lcp') {
      if (value < 2500) return 'vitals-good';
      if (value < 4000) return 'vitals-warn';
      return 'vitals-poor';
    }
    if (metric === 'fcp') {
      if (value < 1800) return 'vitals-good';
      if (value < 3000) return 'vitals-warn';
      return 'vitals-poor';
    }
    return '';
  };
  if (!elements.variantsBody) return;
  elements.variantsBody.innerHTML = entry.variants
    .map(
      (variant) => `
        <tr>
          <td><span class="pill">${variant.format}</span></td>
          <td>${formatQuality(variant.quality)}</td>
          <td>${formatBytes(variant.size)}</td>
          <td>${formatCompression(variant.compressionRatio)}</td>
          <td>${formatTime(variant.loadTime)}</td>
          <td>${formatTime(variant.decodeTime)}</td>
          <td class="${vitalsClass('lcp', variant.lcp)}">${formatTime(variant.lcp)}</td>
          <td class="${vitalsClass('fcp', variant.fcp)}">${formatTime(variant.fcp)}</td>
          <td>${formatTime(variant.plt)}</td>
```

```
      </tr>
    `
  )
  .join('');
}

/**
 * Обробляє завантажене зображення: конвертує у всі формати та тестує.
 * @param {File|Blob|string} source
 */
async function handleImageUpload(source) {
 try {
  clearResults();
  const blob = typeof source === 'string' ? await fetchBlob(source) : source;
  if (!blob) {
   createNotification('Не вдалося завантажити файл', 'error');
   return;
  }
  if (blob.size > 30 * 1024 * 1024) {
   createNotification('Файл занадто великий (макс. 30 МБ)', 'error');
   return;
  }

  const isAnimated = await isAnimatedImage(blob);
  if (isAnimated) createNotification('Виявлено анімацію. Тестуємо оригінал.', 'info');

  const originalInfo = await getImageInfo(blob, blob.name || 'image');
  const previewUrl = await blobToDataUrl(blob);
  const formats = getFormatsForImage(isAnimated);

  showProgress();
  await convertAndTestAllFormats(blob, originalInfo, formats, isAnimated);
  hideProgress();

  // Створюємо запис для поточного перегляду/історії
  const entry = {
   id: `entry-${Date.now()}`,
   timestamp: Date.now(),
   originalName: originalInfo.filename,
   originalFormat: (blob.type?.split('/')[1] || 'unknown').toUpperCase(),
   originalSize: blob.size,
   width: originalInfo.width,
   height: originalInfo.height,
   previewUrl,
   variants: state.testResults.map((r, idx) => ({
    id: `${r.format}-${idx}`,
    format: r.format,
    size: r.fileSize,
    width: originalInfo.width,
    height: originalInfo.height,
    compressionRatio: r.compressionRatio,
    loadTime: r.loadTime,
    decodeTime: r.decodeTime,
    lcp: r.lcp,
    fcp: r.fcp,
    plt: r.plt,
    quality: r.quality,
   })),
  };
  state.entries = [entry, ...state.entries];
  state.selectedEntryId = entry.id;
  render();
 } catch (error) {
  console.error('Помилка обробки зображення', error);
  createNotification(error.message ?? 'Помилка обробки зображення', 'error');
  hideProgress();
 }
}

async function fetchBlob(url) {
 const res = await fetch(url);
 if (!res.ok) throw new Error('Не вдалося завантажити URL');
 return res.blob();
}

async function getImageInfo(blob, filename) {
 return new Promise((resolve, reject) => {
  const img = new Image();
  const url = URL.createObjectURL(blob);
  img.onload = () => {
   URL.revokeObjectURL(url);
   resolve({ filename, width: img.width, height: img.height, originalSize: blob.size, mimeType: blob.type });
  };
  img.onerror = () => {
   URL.revokeObjectURL(url);
   reject(new Error('Не вдалося завантажити зображення'));
  };
  img.src = url;
 });
}

function renderHistory(entries) {
 if (!elements.historyBody) return;
 if (!entries.length) {
  elements.historyBody.innerHTML = '<tr><td colspan="5">Історія порожня</td></tr>';
  return;
 }
 elements.historyBody.innerHTML = entries
  .map((entry) => {
   const thumbDataUrl =
    entry.variants.find((variant) => Boolean(variant.dataUrl || variant.url))?.dataUrl ??
    entry.previewUrl ??
    'data:image/svg+xml,<svg xmlns="http://www.w3.org/2000/svg" width="48" height="48"><rect width="48" height="48" fill="%23f8f9fa" stroke="%23e0e0e0"/><text x="50%" y="55%" dominant-baseline="middle" text-anchor="middle" fill="%236c757d" font-size="12">img</text></svg>';
   return `
    <tr data-entry-id="${entry.id}">
     <td><img src="${thumbDataUrl}" alt="" style="width:48px;height:48px;object-fit:cover;border-radius:6px;border:1px solid #e0e0e0;" /></td>
     <td>${entry.originalName}</td>
     <td>${entry.originalFormat.toUpperCase()}</td>
     <td>${new Date(entry.timestamp).toLocaleString()}</td>
     <td>
      <div class="history-actions">
       <button class="btn btn-ghost" data-
```

```
view="${entry.id}">View</button>
          <button class="btn btn-ghost" data-test="${entry.id}">Test Page</button>
          <button class="btn btn-ghost" data-delete="${entry.id}">Delete</button>
        </div>
       </td>
      </tr>
    `;
   })
   .join('');
}

function bindFilters() {
  elements.historyFilter?.addEventListener('change', render);
  elements.historySort?.addEventListener('change', render);
}

function bindHistoryActions() {
  elements.historyBody?.addEventListener('click', async (event) => {
    const viewBtn = event.target.closest('[data-view]');
    const deleteBtn = event.target.closest('[data-delete]');
    const testBtn = event.target.closest('[data-test]');
    if (viewBtn) {
      state.selectedEntryId = viewBtn.dataset.view;
      render();
      return;
    }
    if (testBtn) {
      const entry = state.entries.find((item) => item.id === testBtn.dataset.test);
      if (entry) void openTestPage(entry);
      return;
    }
    if (deleteBtn) {
      const id = deleteBtn.dataset.delete;
      state.entries.find((item) => item.id === id)?.cleanup?.();
      state.entries = await removeHistoryEntry(id);
      if (state.selectedEntryId === id) {
        state.selectedEntryId = state.entries[0]?.id ?? null;
      }
      render();
    }
  });
  elements.clearHistoryBtn?.addEventListener('click', async () => {
    state.entries.forEach((entry) => entry.cleanup?.());
    await clearHistory();
    state.entries = [];
    state.selectedEntryId = null;
    render();
  });
}

function bindTestButton() {
  elements.testButton?.addEventListener('click', () => {
    const entry = state.entries.find((item) => item.id === state.selectedEntryId);
    if (entry) void openTestPage(entry);
  });
}

function bindResultActions() {
  document.getElementById('export-results')?.addEventListener('click', exportResultsToCSV);
  document.getElementById('clear-results')?.addEventListener('click', () => {
    if (confirm('Ви впевнені, що хочете очистити результати?')) {
      clearResults();
      createNotification('Результати очищено', 'info');
    }
  });
}

async function openTestPage(entry) {
  if (!entry) {
    createNotification('Немає запису для тесту', 'warning');
    return;
  }
  const variants = [];
  for (const variant of entry.variants) {
    try {
      const blob = variant.blob || (await getVariantBlob(entry.id, variant.id));
      if (!blob) continue;
      const dataUrl = await blobToDataUrl(blob);
      variants.push({
        id: variant.id,
        format: variant.format,
        dataUrl,
        size: variant.size,
        width: variant.width,
        height: variant.height,
        compressionRatio: variant.compressionRatio,
        quality: variant.quality,
      });
    } catch (error) {
      console.warn('Не вдалося підготувати варіант для тесту', error);
    }
  }
  if (!variants.length) {
    createNotification('Немає доступних варіантів для тесту', 'warning');
    return;
  }
  const targetVariant = variants[0];
  if (!targetVariant) {
    createNotification('Немає доступних варіантів для тесту', 'warning');
    return;
  }

  const testWindow = openTestPageWindow({
    entryId: entry.id,
    imageName: entry.originalName,
    imageFormat: targetVariant.format,
  });

  if (!testWindow) {
    createNotification('Браузер заблокував вікно тесту', 'error');
    return;
  }

  const payloadBuffer = await (await getVariantBlob(entry.id, targetVariant.id))?.arrayBuffer();
  if (!payloadBuffer) {
    createNotification('Не вдалося підготувати файл для тесту', 'error');
    return;
  }
```

```
  const handleReady = (event) => {
    if (event.source !== testWindow) return;
    if (event.data?.action !== 'testPageReady') return;
    testWindow.postMessage(
      {
        action: 'runTest',
        blobData: payloadBuffer,
        imageInfo: {
          id: entry.id,
          filename: entry.originalName,
          category: 'manual',
          width: targetVariant.width,
          height: targetVariant.height,
          mimeType: 'image/' + targetVariant.format,
        },
        format: { name: targetVariant.format, id: targetVariant.id
},
      },
      '*'
    );
  };

  window.addEventListener('message', handleReady, { once:
true });
}

async function handleTestMessage(event) {
  if (event.origin !== window.location.origin) return;
  const { type, entryId, variants, action, metrics, format } =
event.data ?? {};

  // Підтримка старого формату
  if (type === 'TEST_COMPLETE' && entryId &&
Array.isArray(variants)) {
    const target = state.entries.find((item) => item.id ===
entryId);
    if (!target) return;
    const updatedVariants = target.variants.map((variant) => {
      const incoming = variants.find((v) => v.id === variant.id);
      if (!incoming) return variant;
      return {
        ...variant,
        loadTime: incoming.loadTime ?? variant.loadTime,
        decodeTime:          incoming.decodeTime          ??
variant.decodeTime,
        lcp: incoming.lcp ?? variant.lcp,
        compressionRatio:    incoming.compressionRatio    ??
variant.compressionRatio,
      };
    });
    const updatedEntry = { ...target, variants: updatedVariants
};
    state.entries = state.entries.map((item) => (item.id ===
updatedEntry.id ? updatedEntry : item));
    await saveHistory(state.entries);
    render();
    createNotification('Результати тесту збережено',
'success');
    return;
  }

  // Новий формат від test-page.html
  if (action === 'testComplete' && metrics) {
    const target =
      (format?.name      &&      state.entries.find((item)      =>
item.variants.some((v) => v.format === format.name))) ||
      state.entries.find((item)          =>          item.id          ===
state.selectedEntryId);
    if (!target) return;
    const targetFormat = format?.name;
    const updatedVariants = target.variants.map((variant) => {
      if (targetFormat && variant.format !== targetFormat)
return variant;
      return {
        ...variant,
        loadTime: metrics.loadTime ?? variant.loadTime,
        decodeTime:          metrics.decodeTime          ??
variant.decodeTime,
        lcp: metrics.lcp ?? variant.lcp,
        fcp: metrics.fcp ?? variant.fcp,
        plt: metrics.plt ?? variant.plt,
        compressionRatio:    metrics.compressionRatio    ??
variant.compressionRatio,
      };
    });
    const updatedEntry = { ...target, variants: updatedVariants
};
    state.entries = state.entries.map((item) => (item.id ===
updatedEntry.id ? updatedEntry : item));
    await saveHistory(state.entries);
  render();
  createNotification('Результати тесту збережено', 'success');
  }
}

/**
 * Визначає, чи є зображення анімацією.
 * @param {Blob} blob
 * @returns {Promise<boolean>}
 */
async function isAnimatedImage(blob) {
  if (blob.type === 'image/gif') {
    try {
      const arrayBuffer = await blob.arrayBuffer();
      const uint8Array = new Uint8Array(arrayBuffer);
      const                text                =                new
TextDecoder('ascii').decode(uint8Array.slice(0, 1024));
      return text.includes('NETSCAPE2.0');
    } catch (error) {
      console.warn('Не вдалося перевірити GIF на анімацію:',
error);
      return false;
    }
  }
  if (blob.type === 'image/webp') {
    return blob.size > 500 * 1024;
  }
  return false;
}

const getFormatsForImage = (animated) => {
  if (animated) {
    return [{ name: 'Оригінал (GIF/WebP)', type: 'original',
quality: null }];
  }
  return [
    { name: 'PNG', type: 'png', quality: null },
    { name: 'JPG (90)', type: 'jpg', quality: 0.9 },
    { name: 'JPG (75)', type: 'jpg', quality: 0.75 },
    { name: 'WebP Lossy (90)', type: 'webp-lossy', quality: 0.9
},
    { name: 'WebP Lossy (75)', type: 'webp-lossy', quality: 0.75
},
```

```
    { name: 'WebP Lossless', type: 'webp-lossless', quality: null },
  ];
};

const showProgress = () => {
  const section = document.getElementById('progress-section');
  if (section) section.style.display = 'block';
};
const hideProgress = () => {
  const section = document.getElementById('progress-section');
  if (section) setTimeout(() => (section.style.display = 'none'), 1500);
};
const updateProgress = (percent, text, countText) => {
  const bar = document.getElementById('progress-bar');
  const txt = document.getElementById('progress-text');
  const cnt = document.getElementById('progress-count');
  if (bar) bar.style.width = `${Math.max(0, Math.min(100, percent))}%`;
  if (txt) txt.textContent = text;
  if (cnt) cnt.textContent = countText;
};

const sleep = (ms) => new Promise((resolve) => setTimeout(resolve, ms));

/**
 * Відкриває test-page для одного формату.
 * @param {Blob} blob
 * @param {string} formatName
 * @param {number|null} quality
 * @param {object} originalInfo
 *                                                  @returns {Promise<{loadTime:number|null,decodeTime:number|null,lcp:number|null,fcp:number|null,plt:number|null}>}
 */
async function testConvertedImage(blob, formatName, quality, originalInfo) {
  return new Promise((resolve) => {
    const testWindow = window.open('/test-page.html', '_blank', 'width=800,height=600');
    if (!testWindow) {
      createNotification('Дозвольте pop-up для вимірювання метрик', 'warning');
      resolve({ loadTime: null, decodeTime: null, lcp: null, fcp: null, plt: null });
      return;
    }

    const timeout = setTimeout(() => {
      testWindow.close();
      resolve({ loadTime: null, decodeTime: null, lcp: null, fcp: null, plt: null });
    }, 15000);

    const handler = async (event) => {
      if (event.source !== testWindow) return;
      if (event.data?.action === 'testPageReady') {
        const arrayBuffer = await blob.arrayBuffer();
        testWindow.postMessage(
          {
            action: 'runTest',
            blobData: arrayBuffer,
            imageInfo: {
              id: originalInfo.filename || 'manual',
              filename: originalInfo.filename || 'image',
              category: 'manual',
              width: originalInfo.width,
              height: originalInfo.height,
              mimeType: blob.type,
            },
            format: { name: formatName, type: blob.type, quality },
          },
          '*'
        );
      }
      if (event.data?.action === 'testComplete') {
        clearTimeout(timeout);
        window.removeEventListener('message', handler);
        testWindow.close();
        resolve(event.data.metrics || { loadTime: null, decodeTime: null, lcp: null, fcp: null, plt: null });
      }
    };
    window.addEventListener('message', handler);
  });
}

/**
 * Конвертує та тестує всі формати
 */
async function convertAndTestAllFormats(originalBlob, originalInfo, formats, isAnimated) {
  const totalFormats = formats.length;
  let completed = 0;
  state.testResults = [];

  for (const format of formats) {
    try {
      updateProgress((completed / totalFormats) * 100, `Обробка ${format.name}...`, `${completed}/${totalFormats}`);

      let convertedBlob = originalBlob;
      if (format.type === 'original' || isAnimated) {
        // залишаємо як є
      } else {
        convertedBlob = await convertImage(originalBlob, format.type, format.quality);
      }

      const metrics = await testConvertedImage(convertedBlob, format.name, format.quality, originalInfo);
      const blobUrl = URL.createObjectURL(convertedBlob);

      state.testResults.push({
        format: format.name,
        quality: format.quality,
        fileSize: convertedBlob.size,
        compressionRatio: originalBlob.size > 0 ? convertedBlob.size / originalBlob.size : null,
        loadTime: metrics.loadTime,
        decodeTime: metrics.decodeTime,
        lcp: metrics.lcp,
        fcp: metrics.fcp,
        plt: metrics.plt,
        blobUrl,
        filename: `${(originalInfo.filename || 'image').split('.')[0]}_${format.name.replace(/\\s+/g, '_').toLowerCase()}`,
```

```
    });

    completed += 1;
    updateProgress((completed / totalFormats) * 100, `Завершено ${format.name}`, `${completed}/${totalFormats}`);
    await sleep(800);
   } catch (error) {
    console.error(`Помилка при обробці ${format.name}:`, error);
    createNotification(`Помилка: ${format.name}`, 'warning');
    completed += 1;
   }
  }

  // заповнюємо таблицю
  clearResultsTable();
  state.testResults.forEach((res) => addResultToTable(res));
  updateProgress(100, 'Завершено!', `${totalFormats}/${totalFormats}`);
}

function clearResultsTable() {
  const tbody = document.querySelector('#results tbody');
  if (tbody) tbody.innerHTML = '';
}

const formatFileSizeInline = (bytes) => {
  if (!bytes) return '-';
  if (bytes < 1024) return `${bytes} B`;
  if (bytes < 1024 * 1024) return `${(bytes / 1024).toFixed(2)} KB`;
  return `${(bytes / (1024 * 1024)).toFixed(2)} MB`;
};

function addResultToTable(result) {
  const tbody = document.querySelector('#results tbody');
  const resultsSection = document.getElementById('results-section');
  if (!tbody || !resultsSection) return;
  resultsSection.style.display = 'block';

  const formatTime = (ms) => (ms === null || ms === undefined ? '-' : `${ms.toFixed(2)} мс`);
  const formatCompression = (ratio) => {
   if (ratio === null || ratio === undefined) return '-';
   const percent = (1 - ratio) * 100;
   if (percent > 0) return `−${percent.toFixed(1)}%`;
   if (percent < 0) return `+${Math.abs(percent).toFixed(1)}%`;
   return '0%';
  };
  const getVitalsColor = (metric, value) => {
   if (value === null || value === undefined) return '';
   if (metric === 'lcp') {
    if (value < 2500) return 'color:#28a745;';
    if (value < 4000) return 'color:#ffc107;';
    return 'color:#dc3545;';
   }
   if (metric === 'fcp') {
    if (value < 1800) return 'color:#28a745;';
    if (value < 3000) return 'color:#ffc107;';
    return 'color:#dc3545;';
   }
   return '';
  };

  const row = document.createElement('tr');
  row.dataset.format = result.format;
  row.innerHTML = `
   <td><strong>${result.format}</strong></td>
   <td>${result.quality ? `${Math.round(result.quality * 100)}%` : '-'}</td>
   <td>${formatFileSizeInline(result.fileSize)}</td>

<td>${formatCompression(result.compressionRatio)}</td>
   <td>${formatTime(result.loadTime)}</td>
   <td>${formatTime(result.decodeTime)}</td>
   <td style="${getVitalsColor('lcp', result.lcp)}">${formatTime(result.lcp)}</td>
   <td style="${getVitalsColor('fcp', result.fcp)}">${formatTime(result.fcp)}</td>
   <td>${formatTime(result.plt)}</td>
   <td>
    <button class="btn btn-ghost btn-sm" data-download="${result.format}">□ Завантажити</button>
   </td>
  `;
  tbody.appendChild(row);
}

function clearResults() {
  clearResultsTable();
  const resultsSection = document.getElementById('results-section');
  if (resultsSection) resultsSection.style.display = 'none';
  if (state.testResults?.length) {
   state.testResults.forEach((r) => r.blobUrl && URL.revokeObjectURL(r.blobUrl));
  }
  state.testResults = [];
}

function exportResultsToCSV() {
  if (!state.testResults?.length) {
   createNotification('Немає результатів для експорту', 'warning');
   return;
  }
  const header =
   [
    'format',
    'quality',
    'fileSize_bytes',
    'compressionRatio',
    'loadTime_ms',
    'decodeTime_ms',
    'lcp_ms',
    'fcp_ms',
    'plt_ms',
   ].join(',') + '\\n';
  const rows = state.testResults
   .map((r) =>
    [
     r.format,
     r.quality ? (r.quality * 100).toFixed(0) : '',
     r.fileSize,
     r.compressionRatio ? r.compressionRatio.toFixed(4) : '',
     r.loadTime ?? '',
     r.decodeTime ?? '',
     r.lcp ?? '',
     r.fcp ?? '',
     r.plt ?? '',
```

```
    ].join(',')
   )
   .join('\\n');
  const csv = `\\ufeff${header}${rows}`;
  const blob = new Blob([csv], { type: 'text/csv;charset=utf-8;'
});
  const link = document.createElement('a');
  link.href = URL.createObjectURL(blob);
  link.download = `manual_test_results_${Date.now()}.csv`;
  link.style.display = 'none';
  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
  URL.revokeObjectURL(link.href);
  createNotification('CSV експортовано', 'success');
}

document.addEventListener('click', (event) => {
  const downloadBtn = event.target.closest('[data-download]');
  if (downloadBtn) {
    const formatName = downloadBtn.dataset.download;
    const result = state.testResults.find((r) => r.format ===
formatName);
    if (!result || !result.blobUrl) {
      createNotification('Файл не знайдено', 'error');
      return;
    }
    const link = document.createElement('a');
    link.href = result.blobUrl;
    let extension = 'png';
    if (formatName.includes('JPG')) extension = 'jpg';
    if (formatName.includes('WebP')) extension = 'webp';
    if (formatName.includes('GIF')) extension = 'gif';
    link.download = `${result.filename ||
formatName}.${extension}`;
    link.style.display = 'none';
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    createNotification('Файл завантажено', 'success');
  }
});
```

src/modules/batchStorage.js

```
js
/**
 * Відкриває IndexedDB для збереження batch результатів.
 * @returns {Promise<IDBDatabase>}
 */
export const openBatchDB = () =>
  new Promise((resolve, reject) => {
    const request = indexedDB.open('ImagePerformanceDB',
2);
    request.onupgradeneeded = (event) => {
      const db = event.target.result;
      if (!db.objectStoreNames.contains('batchResults')) {
        const store = db.createObjectStore('batchResults', {
keyPath: 'id' });
        store.createIndex('timestamp', 'timestamp', { unique:
false });
      }
    };
    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });

/**
 * Зберігає результати batch тестування.
 * @param
{{config:object,results:Array,summary:object,status?:string}
} batchData
 * @returns {Promise<string>} ID збереженого тесту
 */
export const saveBatchResults = async (batchData) => {
  const db = await openBatchDB();
  const id = batchData.id || `batch_${Date.now()}`;
  const payload = {
    id,
    timestamp: new Date().toISOString(),
    config: batchData.config,
    results: batchData.results,
    summary: batchData.summary,
    status: batchData.status || 'completed',
  };
  return new Promise((resolve, reject) => {
    const tx = db.transaction('batchResults', 'readwrite');
    const store = tx.objectStore('batchResults');
    const request = store.put(payload);
    request.onsuccess = () => resolve(id);
    request.onerror = () => reject(request.error);
  });
};

/**
 * Завантажує всі batch результати.
 * @returns {Promise<Array>}
 */
export const loadAllBatchResults = async () => {
  const db = await openBatchDB();
  return new Promise((resolve, reject) => {
    const tx = db.transaction('batchResults', 'readonly');
    const store = tx.objectStore('batchResults');
    const request = store.getAll();
    request.onsuccess = () => resolve(request.result || []);
    request.onerror = () => reject(request.error);
  });
};

/**
 * Видаляє batch результат за ID.
 * @param {string} id
 * @returns {Promise<void>}
 */
export const deleteBatchResult = async (id) => {
  const db = await openBatchDB();
  return new Promise((resolve, reject) => {
    const tx = db.transaction('batchResults', 'readwrite');
    const store = tx.objectStore('batchResults');
    const request = store.delete(id);
    request.onsuccess = () => resolve();
    request.onerror = () => reject(request.error);
  });
};
```

src/modules/history.js

```
js
import { storage, createNotification } from './utils.js';
```

```
const HISTORY_KEY = 'image-perf-history-v2';
const DB_NAME = 'image-perf-db';
const STORE_NAME = 'variants';
const hasIndexedDB = typeof indexedDB !== 'undefined';

const openDb = () =>
  new Promise((resolve, reject) => {
    if (!hasIndexedDB) {
      reject(new Error('IndexedDB недоступний'));
      return;
    }
    const request = indexedDB.open(DB_NAME, 1);
    request.onupgradeneeded = () => {
      const db = request.result;
      if (!db.objectStoreNames.contains(STORE_NAME)) {
        db.createObjectStore(STORE_NAME);
      }
    };
    request.onsuccess = () => resolve(request.result);
    request.onerror = () => reject(request.error);
  });

const putBlob = async (key, blob) => {
  if (!hasIndexedDB) return;
  if (!blob) return;
  try {
    const db = await openDb();
    const tx = db.transaction(STORE_NAME, 'readwrite');
    tx.objectStore(STORE_NAME).put(blob, key);
    return tx.complete;
  } catch (error) {
    console.warn('IndexedDB недоступний, blob не
збережено', error);
  }
};

const getBlob = async (key) => {
  if (!hasIndexedDB) return null;
  try {
    const db = await openDb();
    return await new Promise((resolve, reject) => {
      const tx = db.transaction(STORE_NAME, 'readonly');
      const request = tx.objectStore(STORE_NAME).get(key);
      request.onsuccess = () => resolve(request.result);
      request.onerror = () => reject(request.error);
    });
  } catch (error) {
    console.warn('IndexedDB недоступний, blob не
завантажено', error);
    return null;
  }
};

const deleteBlob = async (key) => {
  if (!hasIndexedDB) return;
  try {
    const db = await openDb();
    const tx = db.transaction(STORE_NAME, 'readwrite');
    tx.objectStore(STORE_NAME).delete(key);
    return tx.complete;
  } catch (error) {
    console.warn('IndexedDB недоступний, blob не
видалено', error);
  }
};

const buildVariantKey = (entryId, variantId) =>
`${entryId}:${variantId}`;
export const getVariantBlob = (entryId, variantId) =>
getBlob(buildVariantKey(entryId, variantId));

export const getHistoryEntry = async (id) => {
  const history = await loadHistory();
  return history.find((item) => item.id === id) ?? null;
};

const stripTransient = (entry) => {
  const clone = {
    ...entry,
    variants: entry.variants.map((variant) => ({ ...variant })),
  };
  delete clone.previewUrl;
  delete clone.cleanup;
  clone.variants = clone.variants.map((variant) => {
    const next = { ...variant };
    delete next.url;
    delete next.blob;
    return next;
  });
  return clone;
};

export const saveHistory = async (entries) => {
  try {
    const sanitized = entries.map(stripTransient);
    storage.setItem(HISTORY_KEY,
JSON.stringify(sanitized));
    await Promise.all(
      entries.flatMap((entry) =>
        entry.variants.map((variant)                    =>
putBlob(buildVariantKey(entry.id, variant.id), variant.blob))
      )
    );
  } catch (error) {
    console.error('Не вдалось зберегти історію', error);
    createNotification('Не вдалося зберегти історію', 'error');
  }
};

export const loadHistory = async () => {
  try {
    const raw = storage.getItem(HISTORY_KEY);
    if (!raw) return [];
    const parsed = JSON.parse(raw);
    const restored = [];
    for (const entry of parsed) {
      const variants = [];
      for (const variant of entry.variants) {
        const blob = await getBlob(buildVariantKey(entry.id,
variant.id));
        const url = blob ? URL.createObjectURL(blob) : null;
        variants.push({ ...variant, blob, url });
      }
      const previewUrl = variants[0]?.url ?? null;
      restored.push({
        ...entry,
        previewUrl,
        variants,
        cleanup: () => {
          variants.forEach((variant)     =>     variant.url     &&
URL.revokeObjectURL(variant.url));
        },
      });
    }
```

```
    return restored;
  } catch (error) {
    console.warn('Не вдалось прочитати історію', error);
    return [];
  }
};

export const addHistoryEntry = async (entry) => {
  const history = [entry, ...(await loadHistory())].slice(0, 100);
  await saveHistory(history);
  return history;
};

export const removeHistoryEntry = async (id) => {
  const history = await loadHistory();
  const next = history.filter((item) => item.id !== id);
  await saveHistory(next);
  await Promise.all(
    history
      .find((item) => item.id === id)
      ?.variants.map((variant)                                    =>
deleteBlob(buildVariantKey(id, variant.id))) ?? []
  );
  return next;
};

export const clearHistory = async () => {
  storage.removeItem(HISTORY_KEY);
  try {
    const db = await openDb();
    const tx = db.transaction(STORE_NAME, 'readwrite');
    tx.objectStore(STORE_NAME).clear();
  } catch (error) {
    console.warn('Не вдалося очистити IndexedDB', error);
  }
  createNotification('Історію очищено', 'success');
};

export const sortHistory = (entries, sort = 'newest') => {
  const list = [...entries];
  switch (sort) {
    case 'oldest':
      return list.sort((a, b) => a.timestamp - b.timestamp);
    case 'size':
      return list.sort((a, b) => a.originalSize - b.originalSize);
    default:
      return list.sort((a, b) => b.timestamp - a.timestamp);
  }
};

export const filterHistory = (entries, { format = 'all' } = {}) =>
{
  if (format === 'all') return entries;
  return  entries.filter((entry)  =>  entry.originalFormat  ===
format);
};
```

src/modules/icons.js

```
js
export const icons = {
  upload:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M4 16v2a2 2 0 0 0 2 2h12a2 2 0 0 0 2-
2v-2"/><path stroke-linecap="round" stroke-linejoin="round"
d="M12 4v12m0 0 4-4m-4 4-4-4"/></svg>',
  link:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M10.5 13.5 9 12m3-1.5L15 9m-3 7.5L9
15"/><path  stroke-linecap="round"  stroke-linejoin="round"
d="M13.5  6H17a4  4  0  0  1  0  8h-3.5"/><path  stroke-
linecap="round" stroke-linejoin="round" d="M10.5 18H7a4 4
0 0 1 0-8h3.5"/></svg>',
  refresh:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M4 4v6h6M20 20v-6h-6"/><path stroke-
linecap="round" stroke-linejoin="round" d="M5 15a7 7 0 0 0
11.9 2.1M19 9a7 7 0 0 0 0-11.9-2.1"/></svg>',
  download:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M4 16v3a1 1 0 0 0 1 1h14a1 1 0 0 0 1-
1v-3"/><path stroke-linecap="round" stroke-linejoin="round"
d="M7 10l5 5 5-5M12 4v11"/></svg>',
  trash:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M9 4h6m-8 4h10m-1 0-.4 9.2a1 1 0 0 1-
1 .8H9.4a1 1 0 0 1-1-.8L8 8"/></svg>',
  info:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><circle cx="12" cy="12" r="9"/><path stroke-
linecap="round" stroke-linejoin="round" d="M12 8h.01M11
12h1v4h1"/></svg>',
  success:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="m5 13 4 4L19 7"/></svg>',
  warning:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round"  d="M12  9v4m0  4h.01"/><path  stroke-
linecap="round"  stroke-linejoin="round"  d="m10.3  4.3-7
12A1 1 0 0 0 4.2 18h15.6a1 1 0 0 0 .9-1.7l-7-12a1 1 0 0 0-1.7
0z"/></svg>',
  error:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M15 9 9 15m0-6 6 6"/><circle cx="12"
cy="12" r="9"/></svg>',
  chart:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
linejoin="round" d="M4 19h16M7 16V8m5 8V5m5 11v-
6"/></svg>',
  file:
    '<svg  viewBox="0  0  24  24"  fill="none"
stroke="currentColor"  stroke-width="1.5"  aria-
hidden="true"><path  stroke-linecap="round"  stroke-
```

```
linejoin="round" d="M15 3H9a2 2 0 0 0-2 2v14a2 2 0 0 0 2 2h6a2 2 0 0 0 2-2V7z"/><path stroke-linecap="round" stroke-linejoin="round" d="M15 3v4h4"/></svg>',
};

export const getIcon = (name) => icons[name] ?? '';
```

src/modules/imageConverter.js

```
js
import { getFileExtension } from './utils.js';

const MIME_MAP = {
  png: 'image/png',
  jpg: 'image/jpeg',
  jpeg: 'image/jpeg',
  webp: 'image/webp',
  'webp-lossy': 'image/webp',
  'webp-lossless': 'image/webp',
  gif: 'image/gif',
  swf: 'application/x-shockwave-flash',
};

const loadImageBitmap = async (blob) => {
  if (typeof createImageBitmap === 'function') {
    const bitmap = await createImageBitmap(blob);
    return { bitmap, width: bitmap.width, height: bitmap.height };
  }
  const url = URL.createObjectURL(blob);
  try {
    const img = await new Promise((resolve, reject) => {
      const element = new Image();
      element.onload = () => resolve(element);
      element.onerror = reject;
      element.src = url;
    });
    return { bitmap: img, width: img.naturalWidth, height: img.naturalHeight };
  } finally {
    URL.revokeObjectURL(url);
  }
};

const buildCanvas = (width, height) => {
  if (typeof OffscreenCanvas !== 'undefined') {
    return new OffscreenCanvas(width, height);
  }
  const canvas = document.createElement('canvas');
  canvas.width = width;
  canvas.height = height;
  return canvas;
};

/**
 * Грубо визначає, чи є зображення анімацією.
 * Для GIF/SWF повертає true; для WebP намагається знайти ANMF chunk.
 */
export const detectImageType = async (fileOrBlob) => {
  const type = fileOrBlob.type || MIME_MAP[getFileExtension(fileOrBlob.name)];
  if (!type) return { isAnimated: false, mime: 'application/octet-stream' };
  if (type.includes('gif') || type.includes('shockwave-flash')) {
    return { isAnimated: true, mime: type };
  }
  if (type.includes('webp')) {
    const buffer = await fileOrBlob.arrayBuffer();
    const view = new Uint8Array(buffer);
    // Пошук підрядка "ANMF" (animated webp frame marker)
    const marker = [0x41, 0x4e, 0x4d, 0x46];
    const hasAnmf = view.some((_, index) => marker.every((m, offset) => view[index + offset] === m));
    return { isAnimated: hasAnmf, mime: type };
  }
  return { isAnimated: false, mime: type };
};

/**
 * Конвертує статичне зображення через Canvas API.
 * @param {Blob} sourceBlob
 * @param {'png'|'jpg'|'webp'|'webp-lossy'|'webp-lossless'} targetFormat
 * @param {number} [quality=0.85]
 * @returns {Promise<{blob:Blob,width:number,height:number,format:string,quality?:number}>}
 */
export const convertStaticImage = async (sourceBlob, targetFormat, quality = 0.85) => {
  const { bitmap, width, height } = await loadImageBitmap(sourceBlob);
  const canvas = buildCanvas(width, height);
  const ctx = canvas.getContext('2d');
  ctx.drawImage(bitmap, 0, 0, width, height);
  const mime = MIME_MAP[targetFormat] ?? 'image/png';
  const isLosslessWebp = targetFormat === 'webp-lossless';
  const blob = await new Promise((resolve, reject) => {
    const q = mime === 'image/jpeg' ? quality : isLosslessWebp ? 1 : quality;
    const encoderOptions = Number.isFinite(q) ? Math.min(Math.max(q, 0.05), 1) : undefined;
    canvas.convertToBlob
      ? canvas
          .convertToBlob({ type: mime, quality: encoderOptions })
          .then(resolve)
          .catch(reject)
      : canvas.toBlob(
          (result) => {
            if (!result) {
              reject(new Error('Неможливо конвертувати зображення.'));
              return;
            }
            resolve(result);
          },
          mime,
          encoderOptions
        );
  });
  if (bitmap.close) bitmap.close();
  return { blob, width, height, format: targetFormat, quality: mime === 'image/jpeg' ? quality : undefined };
};

/**
 * Заглушка для анімацій: наразі повертає null для непідтримуваних форматів.
 * @param {Blob} sourceBlob
```

```
 * @param {'gif'|'webp'} targetFormat
 */
export const convertAnimatedImage = async (sourceBlob, targetFormat) => {
  // Повноцінна підтримка потребує WebAssembly/FFmpeg; наразі повертаємо null.
  console.warn('Анімована конвертація обмежена, повертаю вихідний blob');
  return { blob: sourceBlob, width: null, height: null, format: targetFormat };
};

/**
 * Генерує набір форматів для файлу
 * @param {File|Blob} originalFile
 * @returns {Promise<{isAnimated:boolean, variants:Array<object>}>}
 */
export const generateFormatVariants = async (originalFile) => {
  const { isAnimated, mime } = await detectImageType(originalFile);
  const variants = [];
  const targetFormats = isAnimated
    ? ['gif', 'webp']
    : ['png', 'jpg', 'webp-lossy', 'webp-lossless'];
  for (const target of targetFormats) {
    try {
      if (isAnimated && (target === 'gif' || target === 'webp')) {
        const converted = await convertAnimatedImage(originalFile, target);
        variants.push({
          ...converted,
          mime: MIME_MAP[target] ?? mime,
          size: converted.blob?.size ?? originalFile.size,
          compressionRatio: originalFile.size ? (converted.blob?.size ?? originalFile.size) / originalFile.size : 1,
        });
        continue;
      }
      const quality = target === 'jpg' ? 0.85 : target === 'webp-lossy' ? 0.8 : 1;
      const converted = await convertStaticImage(originalFile, target, quality);
      variants.push({
        ...converted,
        mime: MIME_MAP[target] ?? mime,
        size: converted.blob.size,
        compressionRatio: originalFile.size ? converted.blob.size / originalFile.size : 1,
      });
    } catch (error) {
      console.warn(`Конвертація у ${target} недоступна`, error);
    }
  }
  return { isAnimated, variants };
};

/**
 * Конвертує зображення у зазначений формат (статичні формати).
 * @param {Blob} sourceBlob
 * @param {string} targetFormat
 * @param {number} [quality]
 * @returns {Promise<Blob>}
 */
export const convertImage = async (sourceBlob, targetFormat, quality) => {
  const supported = ['png', 'jpg', 'jpeg', 'webp', 'webp-lossy', 'webp-lossless'];
  if (!supported.includes(targetFormat)) {
    throw new Error(`Формат ${targetFormat} не підтримується конвертацією`);
  }
  const converted = await convertStaticImage(sourceBlob, targetFormat, quality ?? 0.85);
  return converted.blob;
};
```

src/modules/imageLoader.js

```
js
import { uid, getFileExtension, createNotification } from './utils.js';
import { generateFormatVariants } from './imageConverter.js';

const MAX_SIZE_BYTES = 30 * 1024 * 1024;
const MIME_BY_EXT = {
  gif: 'image/gif',
  jpg: 'image/jpeg',
  jpeg: 'image/jpeg',
  png: 'image/png',
  webp: 'image/webp',
  avif: 'image/avif',
  bmp: 'image/bmp',
  svg: 'image/svg+xml',
  apng: 'image/apng',
  jfif: 'image/jpeg',
  swf: 'application/x-shockwave-flash',
};

const measureDimensions = async (blob) => {
  if (typeof createImageBitmap === 'function') {
    const bitmap = await createImageBitmap(blob);
    const width = bitmap.width;
    const height = bitmap.height;
    if (bitmap.close) bitmap.close();
    return { width, height };
  }
  const url = URL.createObjectURL(blob);
  try {
    const img = await new Promise((resolve, reject) => {
      const element = new Image();
      element.onload = () => resolve(element);
      element.onerror = reject;
      element.src = url;
    });
    return { width: img.naturalWidth, height: img.naturalHeight };
  } finally {
    URL.revokeObjectURL(url);
  }
};

const validateFile = (file) => {
  if (!file) throw new Error('Файл не передано.');
  if (file.size > MAX_SIZE_BYTES) throw new Error('Файл завеликий. Максимум 30 МБ.');
};
```

```
const buildEntry = async ({ blob, name, sourceType }) => {
  const extension = getFileExtension(name);
  let dimensions = { width: 0, height: 0 };
  try {
    dimensions = await measureDimensions(blob);
  } catch (error) {
    console.warn('Не вдалося зчитати розміри зображення', error);
  }
  const { width, height } = dimensions;
  const originalFormat = extension || (blob.type && blob.type.split('/').pop()) || 'unknown';
  const { isAnimated, variants } = await generateFormatVariants(blob);
  const id = uid('entry');
  const previewUrl = URL.createObjectURL(blob);
  const normalizedVariants = variants.map((variant, index) => {
    const objectUrl = variant.blob ? URL.createObjectURL(variant.blob) : null;
    return {
      id: `${id}-var-${index}`,
      format: variant.format,
      mime: variant.mime,
      size: variant.size,
      width: variant.width ?? width,
      height: variant.height ?? height,
      compressionRatio: variant.compressionRatio ?? 1,
      loadTime: null,
      decodeTime: null,
      quality: variant.quality ?? null,
      url: objectUrl,
      blob: variant.blob,
    };
  });

  return {
    id,
    timestamp: Date.now(),
    originalName: name,
    originalFormat,
    originalSize: blob.size,
    width,
    height,
    isAnimated,
    sourceType,
    previewUrl,
    variants: normalizedVariants,
    testPageMetrics: null,
    cleanup: () => {
      URL.revokeObjectURL(previewUrl);
      normalizedVariants.forEach((variant) => variant.url && URL.revokeObjectURL(variant.url));
    },
  };
};

const processFile = async (file) => {
  validateFile(file);
  return buildEntry({ blob: file, name: file.name, sourceType: 'file' });
};

const processUrl = async (url) => {
  if (!url) throw new Error('URL не може бути порожнім.');
  const response = await fetch(url, { cache: 'no-store' });
  if (!response.ok) throw new Error(`Не вдалося завантажити ресурс (${response.status}).`);
  const arrayBuffer = await response.arrayBuffer();
  const extension = getFileExtension(new URL(url).pathname);
  const blob = new Blob([arrayBuffer], { type: response.headers.get('content-type') || MIME_BY_EXT[extension] });
  const name = new URL(url).pathname.split('/').pop() || 'remote-resource';
  return buildEntry({ blob, name, sourceType: 'url' });
};

export const loadImageWithVariants = async (source) => {
  try {
    if (source instanceof File) {
      return await processFile(source);
    }
    if (typeof source === 'string') {
      return await processUrl(source);
    }
    throw new Error('Невідомий тип даних.');
  } catch (error) {
    createNotification(error.message, 'error');
    throw error;
  }
};
```

src/modules/statistics.js

```
js
/**
 * Обчислює описову статистику для числового масиву.
 * @param {number[]} values
 * @returns {{mean:number,median:number,stdDev:number,min:number,max:number,p25:number,p75:number,p95:number}}
 */
export const calculateDescriptiveStats = (values = []) => {
  const numeric = values.filter((v) => Number.isFinite(v));
  const n = numeric.length;
  if (!n) {
    return { mean: 0, median: 0, stdDev: 0, min: 0, max: 0, p25: 0, p75: 0, p95: 0 };
  }
  const sorted = [...numeric].sort((a, b) => a - b);
  const sum = sorted.reduce((acc, v) => acc + v, 0);
  const mean = sum / n;
  const median = sorted[Math.floor(n / 2)];
  const variance = sorted.reduce((acc, v) => acc + (v - mean) ** 2, 0) / Math.max(1, n - 1);
  const stdDev = Math.sqrt(variance);
  const pick = (p) => sorted[Math.min(sorted.length - 1, Math.max(0, Math.floor(sorted.length * p)))];
  return {
    mean,
    median,
    stdDev,
    min: sorted[0],
    max: sorted[sorted.length - 1],
    p25: pick(0.25),
    p75: pick(0.75),
    p95: pick(0.95),
  };
};
```

```
/**
 * Коефіцієнт кореляції Пірсона.
 * @param {number[]} x
 * @param {number[]} y
 * @returns {number}
 */
export const pearsonCorrelation = (x = [], y = []) => {
  const n = Math.min(x.length, y.length);
  if (n === 0) return 0;
  const xs = x.slice(0, n);
  const ys = y.slice(0, n);
  const meanX = xs.reduce((a, b) => a + b, 0) / n;
  const meanY = ys.reduce((a, b) => a + b, 0) / n;
  let num = 0;
  let sumSqX = 0;
  let sumSqY = 0;
  for (let i = 0; i < n; i += 1) {
    const dx = xs[i] - meanX;
    const dy = ys[i] - meanY;
    num += dx * dy;
    sumSqX += dx ** 2;
    sumSqY += dy ** 2;
  }
  const denom = Math.sqrt(sumSqX * sumSqY);
  return denom === 0 ? 0 : num / denom;
};

/**
 * Обчислює Cohen's d для двох вибірок.
 * @param {number[]} group1
 * @param {number[]} group2
 * @returns {number}
 */
export const calculateEffectSize = (group1 = [], group2 = [])
=> {
  if (!group1.length || !group2.length) return 0;
  const mean1 = group1.reduce((a, b) => a + b, 0) /
group1.length;
  const mean2 = group2.reduce((a, b) => a + b, 0) /
group2.length;
  const var1 =
    group1.reduce((acc, v) => acc + (v - mean1) ** 2, 0) /
Math.max(1, group1.length - 1);
  const var2 =
    group2.reduce((acc, v) => acc + (v - mean2) ** 2, 0) /
Math.max(1, group2.length - 1);
  const pooledSD = Math.sqrt((var1 + var2) / 2);
  return pooledSD === 0 ? 0 : (mean2 - mean1) / pooledSD;
};

/**
 * Вілкоксон для парних спостережень (наближення
нормальним розподілом).
 * @param {Array<object>} pairs - масив об'єктів з двома
значеннями
 * @param {string} [metricKey] - необов'язковий ключ,
якщо дані вкладені
 *                                                  @returns
{{W:number,pValue:number|null,effectSize:number,isSignifi
cant:boolean,interpretation:string}}
 */
export const wilcoxonSignedRank = (pairs = [], metricKey)
=> {
  const parsed = pairs
    .map((item) => extractPair(item, metricKey))
    .filter(Boolean)
    .map(({ a, b }) => ({ diff: b - a }));
  const filtered = parsed.filter((item) => item.diff !== 0);
  const n = filtered.length;
  if (!n) return { W: 0, pValue: null, effectSize: 0, isSignificant:
false, interpretation: 'Немає пар для аналізу' };

  const ranked = rankAbsolute(filtered.map((item) =>
Math.abs(item.diff)));
  let Wpos = 0;
  let Wneg = 0;
  ranked.forEach((rank, index) => {
    if (filtered[index].diff > 0) Wpos += rank;
    else Wneg += rank;
  });
  const W = Math.min(Wpos, Wneg);
  const meanW = (n * (n + 1)) / 4;
  const varW = (n * (n + 1) * (2 * n + 1)) / 24;
  const z = (W - meanW) / Math.sqrt(varW || 1);
  const pValue = 2 * (1 - normalCdf(Math.abs(z)));
  const effectSize = Math.abs(z) / Math.sqrt(n);
  const isSignificant = pValue < 0.05;
  const interpretation = isSignificant ? 'Статистично значуща
різниця' : 'Значущої різниці не виявлено';
  return { W, pValue, effectSize, isSignificant, interpretation
};
};

const extractPair = (item, metricKey) => {
  if (metricKey && item?.[metricKey]) {
    const val = item[metricKey];
    if (Array.isArray(val) && val.length >= 2) return { a:
val[0], b: val[1] };
    if (val && typeof val === 'object' && 'a' in val && 'b' in
val) return { a: val.a, b: val.b };
    if (val && typeof val === 'object' && 'before' in val &&
'after' in val)
      return { a: val.before, b: val.after };
  }
  if ('a' in item && 'b' in item) return { a: item.a, b: item.b };
  if ('before' in item && 'after' in item) return { a: item.before,
b: item.after };
  return null;
};

const rankAbsolute = (values) => {
  const pairs = values.map((v, index) => ({ v, index })).sort((a,
b) => a.v - b.v);
  const ranks = Array(values.length).fill(0);
  let i = 0;
  while (i < pairs.length) {
    let j = i;
    while (j < pairs.length && pairs[j].v === pairs[i].v) j += 1;
    const avgRank = (i + j + 1) / 2;
    for (let k = i; k < j; k += 1) {
      ranks[pairs[k].index] = avgRank;
    }
    i = j;
  }
  return ranks;
};

const normalCdf = (z) => {
  // Апроксимація через erf
  return (1 + erf(z / Math.SQRT2)) / 2;
};

const erf = (x) => {
  // Чисельне наближення Гауса
```

```
  const sign = Math.sign(x);
  const a1 = 0.254829592;
  const a2 = -0.284496736;
  const a3 = 1.421413741;
  const a4 = -1.453152027;
  const a5 = 1.061405429;
  const p = 0.3275911;
  const t = 1 / (1 + p * Math.abs(x));
  const y =
    1 -
    (((((a5 * t + a4) * t + a3) * t + a2) * t + a1) * t + Math.exp(-
x * x));
  return sign * y;
};
```

src/modules/testPage.js

```
js
/**
 * Відкриває тестову сторінку для entryId
 *                                                    @param
{{entryId:string,imageName?:string,imageFormat?:string}}
params
 * @returns {Window|null}
 */
export const openTestPageWindow = ({ entryId, imageName,
imageFormat } = {}) => {
  const base = new URL('/test-page.html',
window.location.origin);
  if (entryId) base.searchParams.set('entryId', entryId);
  if (imageName) base.searchParams.set('imageName',
imageName);
  if (imageFormat) base.searchParams.set('imageFormat',
imageFormat);
  base.searchParams.set('ts', Date.now().toString());
  return window.open(base.toString(), 'image-page-test',
'width=1200,height=900');
};
```

src/modules/utils.js

```
js
import { showToast } from '../components/toast.js';

/**
 * Генерує унікальний ідентифікатор для вимірювань
 * @param {string} [prefix]
 * @returns {string}
 */
export const uid = (prefix = 'perf') => `${prefix}-
${Math.random().toString(36).slice(2,                8)}-
${Date.now().toString(36)}`;

/**
 * Форматує байти у читабельний вигляд
 * @param {number} bytes
 * @returns {string}
 */
export const formatBytes = (bytes) => {
  if (!Number.isFinite(bytes)) return '0 B';
  if (bytes === 0) return '0 B';
  const units = ['B', 'KB', 'MB', 'GB'];
  const i = Math.floor(Math.log(bytes) / Math.log(1024));
  return `${(bytes / 1024 ** i).toFixed(2)} ${units[i]}`;
};

/**
 * Форматує тривалість у мілісекундах
 * @param {number} ms
 * @returns {string}
 */
export const formatDuration = (ms) => {
  if (!Number.isFinite(ms)) return '-';
  if (ms < 1) {
    return `${(ms * 1000).toFixed(1)} µs`;
  }
  if (ms > 1000) {
    return `${(ms / 1000).toFixed(2)} c`;
  }
  return `${ms.toFixed(2)} мс`;
};

/**
 * Розраховує швидкість завантаження
 * @param {number} bytes
 * @param {number} ms
 * @returns {string}
 */
export const formatSpeed = (bytes, ms) => {
  if (!Number.isFinite(bytes) || !Number.isFinite(ms) || ms ===
0) {
    return '-';
  }
  const kbPerSec = bytes / 1024 / (ms / 1000);
  return `${kbPerSec.toFixed(1)} KB/s`;
};

/**
 * Приводить значення до діапазону
 * @param {number} value
 * @param {number} min
 * @param {number} max
 * @returns {number}
 */
export const clamp = (value, min, max) =>
Math.min(Math.max(value, min), max);

/**
 * Перевіряє підтримуваний формат файлу
 * @param {string} name
 * @returns {boolean}
 */
export const isSupportedFormat = (name = '') =>
/\.(gif|jpe?g|png|webp|swf|svg|bmp|apng|avif)$/i.test(name);

/**
 * Повертає розширення файлу у нижньому регістрі
 * @param {string} name
 * @returns {string}
 */
export const getFileExtension = (name = '') =>
name.split('.').pop()?.toLowerCase() ?? '';

/**
 * Показує toast-повідомлення
 * @param {string} message
 * @param {'info'|'success'|'warning'|'error'} [type]
 */
export const createNotification = (message, type = 'info') => {
  showToast(message, type);
```

```
};

/**
 * Обмежена робота з localStorage із запасним варіантом
 */
export const storage = (() => {
  try {
    const testKey = '__perf-test__';
    window.localStorage.setItem(testKey, '1');
    window.localStorage.removeItem(testKey);
    return window.localStorage;
  } catch (error) {
    console.warn('LocalStorage недоступний, використовую тимчасовий об’єкт', error);
    return {
      store: new Map(),
      getItem(key) {
        return this.store.get(key) ?? null;
      },
      setItem(key, value) {
        this.store.set(key, value);
      },
      removeItem(key) {
        this.store.delete(key);
      },
    };
  }
})();

/**
 * Завантажує файл користувачу
 * @param {string} filename
 * @param {string|Blob} data
 * @param {string} mime
 */
export const downloadFile = (filename, data, mime = 'application/json') => {
  const blob = data instanceof Blob ? data : new Blob([data], { type: mime });
  const url = URL.createObjectURL(blob);
  const link = document.createElement('a');
  link.href = url;
  link.download = filename;
  link.click();
  setTimeout(() => URL.revokeObjectURL(url), 1000);
};

/**
 * Дебаунсер для дорогих операцій
 * @param {Function} fn
 * @param {number} delay
 * @returns {Function}
 */
export const debounce = (fn, delay = 200) => {
  let timer;
  return (...args) => {
    clearTimeout(timer);
    timer = setTimeout(() => fn(...args), delay);
  };
};

/**
 * Копіює текст до буфера обміну
 * @param {string} text
 * @returns {Promise<void>}
 */
export const copyToClipboard = async (text) => {
  if (!navigator?.clipboard) {
    createNotification('Буфер обміну недоступний у цьому браузері', 'error');
    return;
  }
  await navigator.clipboard.writeText(text);
  createNotification('Скопійовано до буфера обміну', 'success');
};

/**
 * Форматує timestamp у локальний рядок
 * @param {number} value
 * @returns {string}
 */
export const formatTimestamp = (value) => {
  if (!value) return '-';
  return new Date(value).toLocaleString();
};

/**
 * Перевіряє валідність URL
 * @param {string} value
 * @returns {boolean}
 */
export const isValidUrl = (value = '') => {
  try {
    const parsed = new URL(value);
    return ['http:', 'https:'].includes(parsed.protocol);
  } catch (error) {
    return false;
  }
};

/**
 * Конвертує Blob у data URL (base64)
 * @param {Blob} blob
 * @returns {Promise<string>}
 */
export const blobToDataUrl = (blob) =>
  new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onloadend = () => resolve(reader.result);
    reader.onerror = reject;
    reader.readAsDataURL(blob);
  });
```

src/statistics.html

```
html
<!doctype html>
<html lang="uk">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Статистика · Image Performance Analyzer</title>
    <link rel="icon" type="image/svg+xml" href="/icons/speed.svg" />
    <link rel="stylesheet" href="/styles/minimal-theme.css" />
    <script src="https://cdn.jsdelivr.net/npm/chart.js@4.4.0/dist/chart.umd.min.js"></script>
    <style>
      .visualization-section {
```

```
      margin-top: 3rem;
      margin-bottom: 3rem;
    }
    .chart-container {
      position: relative;
      height: 400px;
      margin: 1rem 0;
    }
    .chart-container canvas {
      max-height: 100%;
    }
  </style>
 </head>
 <body>
  <div class="page">
   <nav style="padding: 1rem; border-bottom: 1px solid #ddd; margin-bottom: 1rem;">
    <div class="container">
     <a href="/">Home</a> |
     <a href="/batch-testing.html">Batch Testing</a> |
     <a href="/statistics.html">Statistics</a> |
     <a href="/instructions.html">□ Інструкція</a>
    </div>
   </nav>

   <header class="section">
    <h1 class="section-title">Статистика</h1>
    <p class="muted">Оберіть batch тест та перегляньте описову статистику і порівняльний аналіз.</p>
   </header>

   <section class="section">
    <h2 class="section-title">Вибір batch тесту</h2>
    <select class="input" id="batch-selector" aria-label="Вибрати batch test">
     <option value="">Немає доступних batch тестів</option>
    </select>
   </section>

   <section class="section" id="descriptive-stats-section">
    <div id="descriptive-stats-container">
     <p class="muted">Завантажте результати batch тестування</p>
    </div>
   </section>

   <section class="card">
    <h2>□ Порівняльний аналіз (Тест Вілкоксона)</h2>
    <p class="text-muted">
     Непараметричний тест для парних вибірок. Перевіряє статистичну значущість різниць між форматами.
    </p>

    <div style="display: grid; grid-template-columns: 1fr 1fr auto; gap: 1rem; align-items: end; margin: 1.5rem 0;">
     <div>
      <label for="format-a" style="display: block; margin-bottom: 0.5rem; font-weight: 500;">
       Формат A (базовий):
      </label>
      <select id="format-a" class="input">
       <option value="">-- Оберіть формат --</option>
      </select>
     </div>
     <div>
      <label for="format-b" style="display: block; margin-bottom: 0.5rem; font-weight: 500;">
       Формат B (порівнюваний):
      </label>
      <select id="format-b" class="input">
       <option value="">-- Оберіть формат --</option>
      </select>
     </div>

     <div>
      <button id="run-wilcoxon" class="btn btn-primary" disabled>
       Виконати тест
      </button>
     </div>
    </div>

    <div id="wilcoxon-results" style="display: none;">
     <!-- Результати з'являться тут -->
    </div>

    <div id="wilcoxon-empty" class="text-muted" style="padding: 2rem; text-align: center;">
     Оберіть два формати для порівняння
    </div>
   </section>

   <section class="section visualization-section">
    <h2>□ Візуалізація: Load Time</h2>
    <div class="chart-container" id="chart-loadtime-boxplot-container">
     <canvas id="chart-loadtime-boxplot"></canvas>
    </div>
    <p class="muted" style="font-size:0.9rem;">Box plot показує розподіл Load Time для кожного формату (мс)</p>
   </section>

   <section class="section visualization-section">
    <h2>□ File Size vs Load Time</h2>
    <div class="chart-container" id="chart-scatter-size-time-container">
     <canvas id="chart-scatter-size-time"></canvas>
    </div>
    <p class="muted" style="font-size:0.9rem;">Scatter plot: залежність Load Time від розміру файлу</p>
   </section>

   <section class="section visualization-section">
    <h2>□ Heatmap: Порівняння форматів</h2>
    <div class="chart-container" id="chart-heatmap-container">
     <canvas id="chart-heatmap"></canvas>
    </div>
    <p class="muted" style="font-size:0.9rem;">Нормалізовані значення метрик (0 = найгірше, 100 = найкраще)</p>
   </section>

   <section class="section visualization-section">
    <h2>□ Матриця кореляцій</h2>
    <div class="chart-container" id="chart-correlation-container">
     <canvas id="chart-correlation"></canvas>
    </div>
    <p class="muted" style="font-size:0.9rem;">Кореляція
```

```
Пірсона між метриками (-1 до +1)</p>
      </section>
    </div>
    <script type="module" src="/statistics.js"></script>
  </body>
</html>
```

src/statistics.js

```
js
import { loadAllBatchResults } from './modules/batchStorage.js';
import { calculateDescriptiveStats, pearsonCorrelation, wilcoxonSignedRank } from './modules/statistics.js';

const charts = {};

const formatFileSize = (bytes) => {
  if (bytes === null || bytes === undefined) return '-';
  if (bytes < 1024) return `${bytes.toFixed(0)} B`;
  if (bytes < 1024 * 1024) return `${(bytes / 1024).toFixed(2)} KB`;
  return `${(bytes / (1024 * 1024)).toFixed(2)} MB`;
};

const formatTime = (ms) => {
  if (ms === null || ms === undefined) return '-';
  return `${ms.toFixed(2)} мс`;
};

const formatCompressionRatio = (ratio) => {
  if (ratio === null || ratio === undefined) return '-';
  const percent = (1 - ratio) * 100;
  if (percent > 0) return `−${percent.toFixed(1)}%`;
  if (percent < 0) return `+${Math.abs(percent).toFixed(1)}%`;
  return '0%';
};

const getMetricLabel = (metric) => {
  const labels = {
    loadTime: 'Load Time (мс)',
    decodeTime: 'Decode Time (мс)',
    lcp: 'LCP (мс)',
    fcp: 'FCP (мс)',
    plt: 'PLT (мс)',
    fileSize: 'File Size',
    compressionRatio: 'Compression Ratio',
  };
  return labels[metric] || metric;
};

document.addEventListener('DOMContentLoaded', () => {
  void loadStatistics();
});

/**
 * Завантажує доступні batch results і налаштовує дропдаун.
 */
async function loadStatistics() {
  const selector = document.getElementById('batch-selector');
  const descContainer = document.getElementById('descriptive-stats-container');
  const wilcoxonContainer = document.getElementById('wilcoxon-container');
  const emptyMessage = 'Завантажте результати batch тестування';
  try {
    const batches = await loadAllBatchResults();
    if (!batches.length) {
      if (selector) selector.innerHTML = '<option value="">Немає доступних batch тестів</option>';
      setContainerPlaceholder(descContainer, emptyMessage);
      setContainerPlaceholder(wilcoxonContainer, emptyMessage);
      clearCharts();
      return;
    }
    populateBatchDropdown(selector, batches);
    selector?.addEventListener('change', (event) => {
      const batch = batches.find((item) => item.id === event.target.value);
      if (batch) renderAllVisualizations(batch.results);
    });

    // Автовибір з URL ?batch=
    const urlParams = new URLSearchParams(window.location.search);
    const batchIdParam = urlParams.get('batch');
    const initialBatch = batchIdParam
      ? batches.find((b) => b.id === batchIdParam) ?? batches[0]
      : batches[0];

    if (initialBatch && selector) {
      selector.value = initialBatch.id;
      renderAllVisualizations(initialBatch.results);
      // прокрутка до статистики
      setTimeout(() => {
        document.getElementById('descriptive-stats-container')?.scrollIntoView({ behavior: 'smooth', block: 'start' });
      }, 300);
    }
  } catch (error) {
    setContainerPlaceholder(descContainer, 'Не вдалося завантажити статистику');
    setContainerPlaceholder(wilcoxonContainer, 'Не вдалося завантажити статистику');
    console.warn('loadStatistics error', error);
  }
}

function populateBatchDropdown(selector, batches) {
  if (!selector) return;
  const staticBatches = batches.filter((b) => b.contentType === 'static' || !b.contentType);
  const animatedBatches = batches.filter((b) => b.contentType === 'animated');
  const fragment = document.createDocumentFragment();
  const defaultOpt = document.createElement('option');
  defaultOpt.value = '';
  defaultOpt.textContent = '-- Оберіть batch test --';
  fragment.appendChild(defaultOpt);

  const buildGroup = (label, list) => {
    if (!list.length) return;
    const optgroup = document.createElement('optgroup');
    optgroup.label = label;
    list.forEach((batch) => {
      const option = document.createElement('option');
      option.value = batch.id;
```

```
    const info = `${batch.summary?.images ?? 0} зображень, ${batch.summary?.totalTests ?? 0} тестів`;
    option.textContent = `${new Date(batch.timestamp).toLocaleString('uk-UA')} - ${info}`;
    optgroup.appendChild(option);
   });
   fragment.appendChild(optgroup);
  };

  buildGroup('□ Статичні зображення', staticBatches);
  buildGroup('□ Анімації', animatedBatches);
  selector.innerHTML = '';
  selector.appendChild(fragment);
}

function renderAllVisualizations(results = []) {
  renderDescriptiveStatsTable(results);
  initWilcoxonComparison(results);
  renderLoadTimeBoxPlot(results);
  renderScatterPlot(results);
  renderHeatmap(results);
  renderCorrelationMatrix(results);
}

function setContainerPlaceholder(container, message) {
  if (!container) return;
  container.innerHTML = `<p class="muted">${message}</p>`;
}

function clearCharts() {
  Object.values(charts).forEach((chart) => chart?.destroy());
}

function initWilcoxonComparison(results) {
  const formatA = document.getElementById('format-a');
  const formatB = document.getElementById('format-b');
  const runBtn = document.getElementById('run-wilcoxon');
  const resultsBlock = document.getElementById('wilcoxon-results');
  const emptyBlock = document.getElementById('wilcoxon-empty');
  if (!formatA || !formatB || !runBtn || !resultsBlock || !emptyBlock) return;

  const formats = [...new Set(results.map((r) => r.format))].sort();
  const options = ['<option value="">-- Оберіть формат --</option>', ...formats.map((f) => `<option value="${f}">${f}</option>`)].join('');
  formatA.innerHTML = options;
  formatB.innerHTML = options;

  const updateButton = () => {
    const hasSelection = formatA.value && formatB.value && formatA.value !== formatB.value;
    runBtn.disabled = !hasSelection;
  };

  formatA.onchange = updateButton;
  formatB.onchange = updateButton;

  runBtn.onclick = () => {
    const fa = formatA.value;
    const fb = formatB.value;
    if (!fa || !fb || fa === fb) {
      alert('Оберіть два різні формати');
      return;
    }
    runWilcoxonAnalysis(results, fa, fb);
  };

  // Reset state
  resultsBlock.style.display = 'none';
  emptyBlock.style.display = 'block';
}

function runWilcoxonAnalysis(results, formatA, formatB) {
  const container = document.getElementById('wilcoxon-results');
  const empty = document.getElementById('wilcoxon-empty');
  if (!container || !empty) return;

  empty.style.display = 'none';
  container.style.display = 'block';

  const metrics = ['loadTime', 'decodeTime', 'lcp', 'fcp', 'plt', 'fileSize'];
  let html = `
    <h3 style="margin-top: 1.5rem;">
      Порівняння: <strong>${formatA}</strong> vs <strong>${formatB}</strong>
    </h3>
    <table class="table table-striped" style="margin-top: 1rem;">
      <thead>
        <tr>
          <th>Метрика</th>
          <th>Median A</th>
          <th>Median B</th>
          <th>Δ (%)</th>
          <th>p-value</th>
          <th>Effect Size (r)</th>
          <th>Інтерпретація</th>
        </tr>
      </thead>
      <tbody>
  `;

  metrics.forEach((metric) => {
    const dataA = results
      .filter((r) => r.format === formatA && Number.isFinite(r[metric]))
      .map((r) => r[metric]);
    const dataB = results
      .filter((r) => r.format === formatB && Number.isFinite(r[metric]))
      .map((r) => r[metric]);

    if (!dataA.length || !dataB.length) {
      html += `
        <tr>
          <td><strong>${getMetricLabel(metric)}</strong></td>
          <td colspan="6" class="text-muted">Недостатньо даних</td>
        </tr>
      `;
      return;
    }

    const n = Math.min(dataA.length, dataB.length);
    const pairs = [];
```

```
  for (let i = 0; i < n; i += 1) {
    pairs.push({ a: dataA[i], b: dataB[i] });
  }

  const testResult = wilcoxonSignedRank(pairs);
  const statsA = calculateDescriptiveStats(dataA);
  const statsB = calculateDescriptiveStats(dataB);
  const delta = ((statsB.median - statsA.median) / (statsA.median || 1)) * 100;

  const formatter =
    metric === 'fileSize'
      ? formatFileSize
      : ['loadTime', 'decodeTime', 'lcp', 'fcp', 'plt'].includes(metric)
        ? formatTime
        : (v) => (Number.isFinite(v) ? v.toFixed(2) : '-');

  let pValueText;
  if (testResult.pValue === null) pValueText = '-';
  else if (testResult.pValue < 0.001) pValueText = '< 0.001';
  else if (testResult.pValue < 0.01) pValueText = testResult.pValue.toFixed(3);
  else pValueText = testResult.pValue.toFixed(4);

  const pValueColor = testResult.isSignificant ? '#28a745' : '#6c757d';
  const r = Math.abs(testResult.effectSize);
  let effectInterpretation = 'Малий ефект';
  if (r >= 0.5) effectInterpretation = 'Великий ефект';
  else if (r >= 0.3) effectInterpretation = 'Середній ефект';

  let interpretation;
  if (testResult.isSignificant) {
    const direction = delta < 0 ? 'краще' : 'гірше';
    interpretation = `<span style="color:${pValueColor};"><strong>Статистично значуще:</strong> ${formatB} ${direction} на ${Math.abs(delta).toFixed(1)}%</span>`;
  } else {
    interpretation = `<span style="color:${pValueColor};">Різниця статистично не значуща</span>`;
  }

  html += `
    <tr>
      <td><strong>${getMetricLabel(metric)}</strong></td>
      <td>${formatter(statsA.median)}</td>
      <td>${formatter(statsB.median)}</td>
      <td style="color:${delta < 0 ? '#28a745' : '#dc3545'};">${delta > 0 ? '+' : ''}${delta.toFixed(1)}%</td>
      <td style="color:${pValueColor}; font-weight:600;">${pValueText}</td>
      <td>${Number.isFinite(testResult.effectSize) ? testResult.effectSize.toFixed(3) : '-'}<br><small class="muted">${effectInterpretation}</small></td>
      <td>${interpretation}</td>
    </tr>
  `;
 });

 html += `
    </tbody>
  </table>
  <div class="card" style="margin-top:1.5rem; border-left:4px solid #0d6efd;">
    <div class="card-body">
      <strong>ℹ Як інтерпретувати:</strong>
      <ul style="margin: 0.5rem 0 0 1.25rem;">
        <li><strong>p-value &lt; 0.05:</strong> різниця статистично значуща (95% впевненість)</li>
        <li><strong>Effect Size (r):</strong> &lt;0.3 малий, 0.3-0.5 середній, &gt;0.5 великий ефект</li>
        <li><strong>Δ (%):</strong> відсоткова зміна медіани (негативне = покращення для часових метрик)</li>
      </ul>
    </div>
  </div>
 `;

 container.innerHTML = html;
}

function renderDescriptiveStatsTable(results) {
 const container = document.getElementById('descriptive-stats-container');
 if (!container) return;
 if (!results.length) {
   setContainerPlaceholder(container, 'Недостатньо даних для побудови статистики');
   return;
 }
 const formats = [...new Set(results.map((r) => r.format))];
 const metrics = ['loadTime', 'decodeTime', 'lcp', 'fcp', 'plt', 'fileSize', 'compressionRatio'];
 let html = '<h3>□ Описова статистика</h3>';
 metrics.forEach((metric) => {
   html += `<h4>${getMetricLabel(metric)}</h4><table class="table table-sm table-striped">
     <thead>
       <tr>
         <th>Формат</th>
         <th>Mean</th>
         <th>Median</th>
         <th>StdDev</th>
         <th>Min</th>
         <th>Max</th>
       </tr>
     </thead>
     <tbody>`;
   formats.forEach((format) => {
     const values = results
       .filter((r) => r.format === format && r[metric] !== null && Number.isFinite(r[metric]))
       .map((r) => r[metric]);
     if (values.length) {
       const stats = calculateDescriptiveStats(values);
       const formatter =
         metric === 'fileSize'
           ? formatFileSize
           : metric === 'compressionRatio'
           ? formatCompressionRatio
           : formatTime;
       html += `<tr>
         <td><strong>${format}</strong></td>
         <td>${formatter(stats.mean)}</td>
         <td>${formatter(stats.median)}</td>
         <td>${formatter(stats.stdDev)}</td>
         <td>${formatter(stats.min)}</td>
         <td>${formatter(stats.max)}</td>
       </tr>`;
     } else {
```

```
      html += `<tr>
        <td><strong>${format}</strong></td>
        <td colspan="5" class="text-muted">Недостатньо даних</td>
      </tr>`;
    }
   });
   html += '</tbody></table>';
  });
  container.innerHTML = html || '<p class="muted">Недостатньо даних для побудови статистики</p>';
}

function renderLoadTimeBoxPlot(results) {
  const container = document.getElementById('chart-loadtime-boxplot-container');
  if (!container) return;
  container.innerHTML = '<canvas id="chart-loadtime-boxplot"></canvas>';
  const canvas = container.querySelector('canvas');
  const datasets = buildMedianByFormat(results, 'loadTime');
  if (!datasets.labels.length) {
    container.innerHTML = '';
    container.appendChild(messageNode('Недостатньо даних для побудови графіка'));
    return;
  }
  try {
    charts.boxplot?.destroy();
    charts.boxplot = new Chart(canvas, {
      type: 'bar',
      data: {
        labels: datasets.labels,
        datasets: [
          {
            label: 'Медіана Load Time (мс)',
            data: datasets.values,
            backgroundColor: 'rgba(54, 162, 235, 0.6)',
            borderColor: 'rgba(54, 162, 235, 1)',
            borderWidth: 1,
          },
        ],
      },
      options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
          y: { beginAtZero: true, title: { display: true, text: 'Час (мс)' } },
          x: { title: { display: true, text: 'Формат зображення' } },
        },
        plugins: {
          title: { display: true, text: 'Медіана Load Time по форматах' },
          legend: { display: false },
          tooltip: {
            callbacks: {
              label: (ctx) => `${ctx.parsed.y.toFixed(2)} мс`,
            },
          },
        },
      },
    });
  } catch (error) {
    console.warn('renderLoadTimeBoxPlot error', error);
  }
}

function renderScatterPlot(results) {
  const container = document.getElementById('chart-scatter-size-time-container');
  if (!container) return;
  container.innerHTML = '<canvas id="chart-scatter-size-time"></canvas>';
  const canvas = container.querySelector('canvas');
  const formats = [...new Set(results.map((r) => r.format))];
  const colors = [
    'rgba(255, 99, 132, 0.6)',
    'rgba(54, 162, 235, 0.6)',
    'rgba(255, 206, 86, 0.6)',
    'rgba(75, 192, 192, 0.6)',
    'rgba(153, 102, 255, 0.6)',
    'rgba(255, 159, 64, 0.6)',
    'rgba(99, 99, 99, 0.6)',
  ];
  const datasets = formats
    .map((format, idx) => {
      const data = results
        .filter((r) => r.format === format && Number.isFinite(r.loadTime) && Number.isFinite(r.fileSize))
        .map((r) => ({ x: r.fileSize / (1024 * 1024), y: r.loadTime }));
      if (!data.length) return null;
      const color = colors[idx % colors.length];
      return {
        label: format,
        data,
        backgroundColor: color,
        borderColor: color.replace('0.6', '1'),
        pointRadius: 4,
      };
    })
    .filter(Boolean);

  if (!datasets.length) {
    container.innerHTML = '';
    container.appendChild(messageNode('Недостатньо даних для побудови графіка'));
    return;
  }

  try {
    charts.scatter?.destroy();
    charts.scatter = new Chart(canvas, {
      type: 'scatter',
      data: { datasets },
      options: {
        responsive: true,
        maintainAspectRatio: false,
        scales: {
          x: {
            title: { display: true, text: 'File Size (MB)' },
            ticks: { callback: (value) => Number(value).toFixed(1) },
          },
          y: { title: { display: true, text: 'Load Time (мс)' } },
        },
        plugins: {
          title: { display: true, text: 'Залежність Load Time від File Size' },
          tooltip: {
            callbacks: {
```

```
            label: (context) => {
              const format = context.dataset.label;
              const size = context.parsed.x.toFixed(2);
              const time = context.parsed.y.toFixed(2);
              return `${format}: ${size} MB, ${time} мс`;
            },
          },
        },
      },
    },
  });
 } catch (error) {
  console.warn('renderScatterPlot error', error);
 }
}

function renderHeatmap(results) {
 const container = document.getElementById('chart-heatmap-container');
 if (!container) return;
 container.innerHTML = '<canvas id="chart-heatmap"></canvas>';
 const canvas = container.querySelector('canvas');
 const formats = [...new Set(results.map((r) => r.format))];
 const metrics = ['loadTime', 'decodeTime', 'lcp', 'fcp', 'plt', 'fileSize'];
 const allValues = {};
 metrics.forEach((m) => {
  const vals = results.map((r) => r[m]).filter((v) => Number.isFinite(v));
  allValues[m] = {
   min: vals.length ? Math.min(...vals) : null,
   max: vals.length ? Math.max(...vals) : null,
  };
 });
 const heatmapData = formats.map((format) =>
  metrics.map((metric) => {
   const values = results.filter((r) => r.format === format && Number.isFinite(r[metric])).map((r) => r[metric]);
   if (!values.length) return null;
   const mean = values.reduce((a, b) => a + b, 0) / values.length;
   const { min, max } = allValues[metric];
   if (!Number.isFinite(min) || !Number.isFinite(max) || min === max) return 50;
   return 100 - ((mean - min) / (max - min)) * 100;
  })
 );
 if (!heatmapData.length) {
  container.innerHTML = '';
  container.appendChild(messageNode('Недостатньо даних для побудови графіка'));
  return;
 }
 try {
  charts.heatmap?.destroy();
  charts.heatmap = new Chart(canvas, {
   type: 'bar',
   data: {
    labels: metrics.map((m) => getMetricLabel(m)),
    datasets: formats.map((format, idx) => ({
     label: format,
     data: heatmapData[idx],
     backgroundColor: `rgba(${100 + idx * 15}, ${150 - idx * 10}, 200, 0.7)`,
    })),
   },
   options: {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
     y: { beginAtZero: true, max: 100, title: { display: true, text: 'Нормалізоване значення (0-100)' } },
    },
    plugins: {
     title: { display: true, text: 'Порівняння форматів (100 = найкраще)' },
     tooltip: {
      callbacks: {
       label: (context) => `${context.dataset.label}: ${context.parsed.y.toFixed(1)} / 100`,
      },
     },
    },
   },
  });
 } catch (error) {
  console.warn('renderHeatmap error', error);
 }
}

function renderCorrelationMatrix(results) {
 const container = document.getElementById('chart-correlation-container');
 if (!container) return;
 const metrics = [
  { key: 'loadTime', label: 'Load Time' },
  { key: 'decodeTime', label: 'Decode Time' },
  { key: 'lcp', label: 'LCP' },
  { key: 'fcp', label: 'FCP' },
  { key: 'plt', label: 'PLT' },
  { key: 'fileSize', label: 'File Size' },
  { key: 'compressionRatio', label: 'Compression' },
 ];
 const matrix = metrics.map((m1) =>
  metrics.map((m2) => {
   const x = results.filter((r) => Number.isFinite(r[m1.key]) && Number.isFinite(r[m2.key])).map((r) => r[m1.key]);
   const y = results.filter((r) => Number.isFinite(r[m1.key]) && Number.isFinite(r[m2.key])).map((r) => r[m2.key]);
   if (x.length < 2 || y.length < 2) return 0;
   if (m1.key === m2.key) return 1;
   return pearsonCorrelation(x, y);
  })
 );

 const table = document.createElement('table');
 table.className = 'table';
 const thead = document.createElement('thead');
 const headerRow = document.createElement('tr');
 headerRow.innerHTML = '<th></th>' + metrics.map((m) => `<th>${m.label}</th>`).join('');
 thead.appendChild(headerRow);
 table.appendChild(thead);
 const tbody = document.createElement('tbody');
 matrix.forEach((row, i) => {
  const tr = document.createElement('tr');
  tr.innerHTML =
   `<th>${metrics[i].label}</th>` +
   row
    .map((val) => {
     let color = '';
     if (val > 0.7) color = 'style="color:#0066cc;"';
     else if (val < -0.7) color = 'style="color:#dc3545;"';
```

```
        return `<td ${color}>${val.toFixed(2)}</td>`;
       })
       .join('');
   tbody.appendChild(tr);
  });
  table.appendChild(tbody);
  container.innerHTML = '';
  container.appendChild(table);
}

function buildMedianByFormat(results, metricKey) {
  const labels = [];
  const values = [];
  const formats = [...new Set(results.map((r) => r.format))];
  formats.forEach((format) => {
   const vals = results.filter((r) => r.format === format &&
Number.isFinite(r[metricKey])).map((r) => r[metricKey]);
   if (!vals.length) return;
   const stats = calculateDescriptiveStats(vals);
   labels.push(format);
   values.push(stats.median);
  });
  return { labels, values };
}

function messageNode(text) {
  const p = document.createElement('p');
  p.className = 'muted';
  p.textContent = text;
  return p;
}
```

src/test-page.html

```
html
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Test Page - Performance Measurement</title>
  <style>
   * {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
   }
   body {
    width: 100vw;
    height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    background: #f5f5f5;
    overflow: hidden;
   }
   #test-image {
    max-width: 100%;
    max-height: 100%;
    object-fit: contain;
    display: block;
   }
   #loading {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    font-family: system-ui, -apple-system, sans-serif;
    color: #666;
    font-size: 14px;
   }
  </style>
</head>
<body>
  <div id="loading">Завантаження зображення...</div>
  <img id="test-image" style="display:none;" alt="Test
Image">

  <script>
   (function() {
    const metrics = {
     loadTime: null,
     decodeTime: null,
     lcp: null,
     fcp: null,
     plt: null
    };

    let lcpObserver = null;
    let fcpObserver = null;
    const pageLoadStart = performance.now();

    if (typeof PerformanceObserver !== 'undefined') {
     try {
      lcpObserver = new PerformanceObserver((list) => {
       const entries = list.getEntries();
       if (entries.length > 0) {
        const lastEntry = entries[entries.length - 1];
        metrics.lcp = lastEntry.renderTime ||
lastEntry.loadTime;
        console.log('LCP виміряно:', metrics.lcp);
       }
      });
      lcpObserver.observe({ type: 'largest-contentful-paint',
buffered: true });
     } catch (error) {
      console.warn('LCP observer не підтримується', error);
     }
    }

    if (typeof PerformanceObserver !== 'undefined') {
     try {
      fcpObserver = new PerformanceObserver((list) => {
       const entries = list.getEntries();
       for (const entry of entries) {
        if (entry.name === 'first-contentful-paint') {
         metrics.fcp = entry.startTime;
         console.log('FCP виміряно:', metrics.fcp);
         break;
        }
       }
      });
      fcpObserver.observe({ type: 'paint', buffered: true });
     } catch (error) {
      console.warn('FCP observer не підтримується', error);
     }
    }

    window.addEventListener('message', async (event) => {
     if (event.data.action === 'runTest') {
      const { blobData, imageInfo, format } = event.data;
```

```
        console.log('Отримано запит на тестування:', format.name);

        try {
          const blob = new Blob([blobData], { type: imageInfo.mimeType || 'image/png' });
          const url = URL.createObjectURL(blob);

          const img = document.getElementById('test-image');
          const loading = document.getElementById('loading');

          const loadStart = performance.now();

          img.onload = async () => {
            const loadEnd = performance.now();
            loading.style.display = 'none';
            img.style.display = 'block';

            const resourceEntries = performance.getEntriesByName(url);
            if (resourceEntries.length > 0) {
              const entry = resourceEntries[resourceEntries.length - 1];
              metrics.loadTime = entry.duration;
            } else {
              metrics.loadTime = loadEnd - loadStart;
            }

            const decodeStart = performance.now();
            try {
              await img.decode();
              metrics.decodeTime = performance.now() - decodeStart;
            } catch (error) {
              console.warn('Decode failed', error);
              metrics.decodeTime = null;
            }

            await new Promise(resolve => setTimeout(resolve, 300));
            metrics.plt = performance.now() - pageLoadStart;

            URL.revokeObjectURL(url);
            if (lcpObserver) lcpObserver.disconnect();
            if (fcpObserver) fcpObserver.disconnect();

            console.log('Метрики зібрано:', metrics);

            window.opener.postMessage(
              {
                action: 'testComplete',
                metrics: metrics,
                format: format || null,
                imageId: imageInfo?.id || null,
              },
              '*'
            );
            setTimeout(() => window.close(), 50);
          };

          img.onerror = () => {
            console.error('Помилка завантаження зображення');
            URL.revokeObjectURL(url);
            window.opener.postMessage(
              {
                action: 'testComplete',
                metrics: {
                  loadTime: null,
                  decodeTime: null,
                  lcp: null,
                  fcp: null,
                  plt: null
                },
                format: format || null,
                imageId: imageInfo?.id || null,
              },
              '*'
            );
            setTimeout(() => window.close(), 50);
          };

          img.src = url;
        } catch (error) {
          console.error('Помилка тестування:', error);
          window.opener.postMessage(
            {
              action: 'testComplete',
              metrics: {
                loadTime: null,
                decodeTime: null,
                lcp: null,
                fcp: null,
                plt: null
              }
            },
            '*'
          );
        }
      });

      if (window.opener) {
        window.opener.postMessage({ action: 'testPageReady' }, '*');
      }
    })();
  </script>
</body>
</html>
```

src/test-page.js

```
js
const params = new URLSearchParams(window.location.search);
const entryId = params.get('entryId');
const imageName = params.get('imageName') || 'Без назви';
const storageKey = entryId ? `ipa-test:${entryId}` : null;

const grid = document.getElementById('variants-grid');
const title = document.getElementById('image-name');
const saveButton = document.getElementById('save-results');

const lcpState = { value: null };

const observeLCP = () => {
  if (typeof PerformanceObserver === 'undefined') return () => {};
  try {
    const observer = new PerformanceObserver((list) => {
      const last = list.getEntries().at(-1);
```

```
    if (!last) return;
    lcpState.value = last.renderTime || last.loadTime || last.startTime;
   });
   observer.observe({ type: 'largest-contentful-paint', buffered: true });
   return () => observer.disconnect();
  } catch (error) {
   console.warn('LCP недоступний', error);
   return () => {};
  }
};

const formatDuration = (value) => {
  if (!Number.isFinite(value)) return '-';
  if (value < 1) return `${(value * 1000).toFixed(1)} µs`;
  if (value >= 1000) return `${(value / 1000).toFixed(2)} c`;
  return `${value.toFixed(0)} мс`;
};

const formatBytes = (bytes) => {
  if (!Number.isFinite(bytes) || bytes <= 0) return '-';
  const units = ['B', 'KB', 'MB'];
  const i = Math.min(units.length - 1, Math.floor(Math.log(bytes) / Math.log(1024)));
  return `${(bytes / 1024 ** i).toFixed(2)} ${units[i]}`;
};

const loadPayload = () => {
  if (!storageKey) return null;
  const raw = window.localStorage.getItem(storageKey);
  if (!raw) return null;
  try {
   return JSON.parse(raw);
  } catch (error) {
   console.warn('Неможливо прочитати payload', error);
   return null;
  }
};

const measureVariant = async (variant, originalSize) => {
  const source = variant.dataUrl || variant.url;
  if (!source) throw new Error('Джерело зображення відсутнє');
  const img = new Image();
  const start = performance.now();
  const loadPromise = new Promise((resolve, reject) => {
   img.onload = resolve;
   img.onerror = reject;
  });
  img.decoding = 'async';
  img.src = source;
  await loadPromise;
  const decodeStart = performance.now();
  await img.decode().catch(() => {});
  const decodeTime = performance.now() - decodeStart;
  const resource = performance.getEntriesByName(img.currentSrc).pop();
  const loadTime = resource?.duration ?? performance.now() - start;
  const compressionRatio = originalSize ? (variant.size ?? originalSize) / originalSize : null;
  return {
   id: variant.id,
   format: variant.format,
   loadTime,
   decodeTime,
   lcp: lcpState.value,
   size: variant.size,
   compressionRatio,
   width: variant.width,
   height: variant.height,
  };
};

const renderVariantCard = (variant, metrics) => {
  const source = variant.dataUrl || variant.url || '';
  const ratioText =
   Number.isFinite(metrics.compressionRatio) && metrics.compressionRatio > 0
    ? `${(metrics.compressionRatio * 100).toFixed(1)}%`
    : '-';
  const card = document.createElement('article');
  card.className = 'card format-card';
  card.innerHTML = `
   <h3 style="margin-top:0;">${variant.format.toUpperCase()} · ${formatBytes(variant.size)}</h3>
   <img src="${source}" alt="${variant.format}" loading="lazy" />
   <div class="metrics">
    <div class="metric"><div class="muted">Load Time</div><strong>${formatDuration(metrics.loadTime)}</strong></div>
    <div class="metric"><div class="muted">Decode Time</div><strong>${formatDuration(metrics.decodeTime)}</strong></div>
    <div class="metric"><div class="muted">LCP</div><strong>${formatDuration(metrics.lcp ?? 0)}</strong></div>
    <div class="metric"><div class="muted">Compression vs original</div><strong>${ratioText}</strong></div>
    <div class="metric"><div class="muted">File Size</div><strong>${formatBytes(variant.size)}</strong></div>
    <div class="metric"><div class="muted">Dimensions</div><strong>${variant.width} × ${variant.height}</strong></div>
   </div>
  `;
  return card;
};

async function main() {
  title.textContent = imageName;
  const payload = loadPayload();
  if (!payload || !payload.variants?.length) {  grid.innerHTML = '<p>Дані для тесту відсутні.</p>';
   return;
  }
  const disconnectLcp = observeLCP();
  const metrics = [];
  for (const variant of payload.variants) {
   const result = await measureVariant(variant, payload.originalSize);
   metrics.push(result);
   grid.appendChild(renderVariantCard(variant, result));
  }
  saveButton.addEventListener('click', () => {
   window.opener?.postMessage({ type: 'TEST_COMPLETE', entryId, variants: metrics }, window.location.origin);
   if (storageKey) window.localStorage.removeItem(storageKey);
```

```
    disconnectLcp();
    window.close();
  });
}

main()
```

ДОДАТОК В

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного університету
науки і технологій
_____Анатолій РАДКЕВИЧ
02.10.25

“IMAGE PERFORMANCE ANALYZER”

Керівництво користувача

ЛИСТ ЗАТВЕРДЖЕННЯ

44165850.1550– 01 І3 01

Представники
підприємства-розробника
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
02.10.25
Керівник розробки
_____ Тетяна
ГРИШЕЧКІНА
02.10.25
Виконавець
_____ Марія ПОСМІТЮХА
02.10.25
Нормоконтролер
_____ Світлана ВОЛКОВА
02.10.25

ЗАТВЕРДЖЕНО

44165850.1550– 01 І3 01

“IMAGE PERFORMANCE ANALYZER”

Керівництво користувача

Листів 9

ЗМІСТ

1 Вступ

1.1 Сфера застосування

Програмний комплекс "Image Performance Analyzer" розроблено для емпіричного вивчення того, як вибір графічного формату впливає на швидкість роботи веб-сторінок. Інструмент надає можливість зіставити часові характеристики завантаження й обробки зображень у форматах GIF, JPG, PNG та WebP.

Цільова аудиторія охоплює науковців, що займаються дослідженням веб-технологій, викладачів профільних дисциплін, а також розробників, які прагнуть підвищити швидкодію своїх онлайн-проектів через обґрунтований вибір медіаконтенту.

1.2 Короткий опис можливостей

Застосунок надає такі можливості:

- можливість швидкого імпорту графіки через перетягування файлів з автоматичним перекодуванням у шість варіантів форматів;
- реєстрація ключових показників продуктивності: часу отримання файлу з мережі, тривалості декодування браузером, метрик LCP і FCP, загального часу завантаження сторінки, фізичного розміру файлу та ступеня компресії; можливість організації пакетного тестування на наборах зображень семи категорій; інтегрований статистичний аналіз із застосуванням непараметричного критерію Вілкоксона для виявлення значущих відмінностей.

1.3 Рівень підготовки користувача

Користувач повинен мати базові навички роботи з комп’ютером та інтернетом. Для користувачів, які здійснюють введення та перевірку даних, достатньо базового рівня знань з роботи з ПК.

2 Призначення та умови застосування

2.1 Види діяльності та функції

- виконання експериментальних робіт з оцінювання швидкодії інтернет-ресурсів;
- педагогічна діяльність у межах освітніх курсів з веб-технологій і оптимізації клієнтської частини застосунків;

- практичне застосування при створенні високопродуктивних веб-сайтів через раціональний підбір медіаресурсів.

2.2 Умови застосування

- Доступність: Рішення реалізоване у вигляді веб-сервісу, доступ до якого здійснюється виключно через мережу Інтернет. Інсталяція додаткових програмних компонентів на клієнтську машину не потрібна;
- Платформенні вимоги: Робота забезпечується у будь-якому актуальному інтернет-оглядачі - Google Chrome версії 90 і вище, Mozilla Firefox 88+, Safari 14+, Microsoft Edge 90 і новіші;
- Апаратні характеристики: Двоядерний центральний процесор з тактовою частотою 1.5 ГГц, оперативна пам'ять обсягом 2 ГБ, монітор з роздільною здатністю від 1024×768 точок, надійне підключення до мережі зі швидкістю від 10 мегабіт за секунду.

3 Підготовка до роботи

Перевірка доступності: Запустіть інтернет-оглядач і введіть у адресний рядок наданий URL застосунку.

Тестування працездатності: Імпортуйте довільне демонстраційне зображення з файлової системи або вкажіть його URL. Процес аналізу ініціюється самостійно після отримання зображення. Перевірте, що порівняльна таблиця містить цифрові дані по всіх форматах.

4 Опис операцій

Виконання вимірювань

Помістіть файл зображення у спеціально відведену зону інтерфейсу методом drag-and-drop або виберіть його кнопкою завантаження (допустимі розширення: JPG, PNG, GIF, WebP; максимальний розмір: 30 мегабайт). Програма автоматично створить варіанти у шести форматах і запустить процес аналізу. По завершенні таблиця відобразить зіставлення результатів.

Вивантаження даних

Використовуйте опцію "Експортувати CSV" для збереження результатів у табличному файлі з правильним UTF-8 кодуванням кирилічних символів. Створений файл можна відкрити у Microsoft Excel або завантажити у статистичні пакети для подальшого аналізу.

5 Операції технологічного процесу обробки даних

Виконання вимірювань за показниками супроводжуваності:

● найменування: Реєстрація показників Web Vitals для спектру форматів зображень;

● умови: Доступність графічного файлу у одному з форматів JPG, PNG, GIF, WebP, розміром до 30 мегабайт;

● підготовчі дії: Перемістіть зображення у цільову область або оберіть через системний діалог вибору файлів;

● основні дії: Ініціюйте вимірювання натисканням "Почати тестування" для старту автоматичного процесу;

● заключні дії: Вивчіть отримані показники у таблиці, виконайте експорт у CSV або відкрийте розділ статистичної обробки;

● необхідні ресурси: Браузер із реалізацією Canvas API і PerformanceObserver, стабільне мережеве з'єднання швидкістю не менше 10 Мбіт/с.

6 Аварійні ситуації

6.1 Дії на випадок недотримання умов виконання технологічного процесу

Рестарт програмного модуля: При появі неполадок у функціонуванні виконайте перезавантаження веб-сторінки клавішею F5 або комбінацією Ctrl+R. Якщо симптоми повторюються, очистіть кеш оглядача і видаліть дані IndexedDB через панель інструментів розробника (натиснути F12, перейти до Application → Storage, виконати очищення).

Аналіз мережевого каналу: Коректність вимірювань часу завантаження критично залежить від швидкості та стабільності інтернет-з'єднання (мінімальна пропускна здатність 10 Мбіт/с). Нестабільний або повільний канал спричиняє значні викиди у зареєстрованих показниках.

6.2 Дії з відновлення програм і/або даних у разі відмови магнітних носіїв або виявлення помилок у даних

Відновлення втраченої інформації: Результати пакетного тестування зберігаються локально у IndexedDB веб-оглядача. У випадку випадкового видалення цих даних браузером доведеться повторно провести вимірювання. Рекомендується регулярно експортувати накопичені результати у CSV файли для створення резервних копій.

Контроль точності вимірювань: Протестуйте еталонне зображення з попередньо відомими характеристиками. Порівняйте зафіксований час завантаження з показаннями вкладки Network у інструментах розробника браузера (F12 → Network). Розбіжність не повинна перевищувати 10 відсотків від виміряного значення.

6.3 Дії у випадках виявлення несанкціонованого втручання в дані

Повідомлення про аномалії: У разі виявлення нетипових показників (наприклад, час завантаження понад 10 секунд для файлів малого розміру, від'ємні значення часу декодування) негайно сповістіть службу технічної підтримки. Збережіть аномальні результати через функцію експорту у CSV для подальшого аналізу.

Дослідження проблеми: Фахівці технічної служби вивчать описану ситуацію і встановлять першопричину збою. На основі результатів діагностики буде розроблено і впроваджено необхідні виправлення для усунення виявленої недосконалості.

Відновлення нормального функціонування: Після ідентифікації та усунення джерела проблеми виконайте очищення тимчасових файлів браузера, вилучіть cookies, пов'язані з даним застосунком, і здійсніть перезапуск операційної системи для повного відновлення штатного режиму експлуатації.

6.4 Дії в інших аварійних ситуаціях

Ліквідація програмних несправностей: У разі виникнення непрогнозованих помилок спробуйте використати інший веб-оглядач. Перевірте відповідність версії вашого браузера мінімальним технічним специфікаціям. Упевніться у наявності підтримки API PerformanceObserver у обраному оглядачі.

Підтримка актуальності ПЗ: Систематично встановлюйте свіжі версії веб-оглядача та операційної системи. Новітні випуски містять патчі безпеки, виправлення

виявлених дефектів та оптимізації швидкодії, що сприяє підвищенню стабільності та надійності роботи інструменту.

7 Рекомендації щодо засвоєння

7.1 Рекомендації щодо засвоєння та експлуатації

Вивчення супровідної документації: Перед початком практичної роботи ретельно прочитайте даний посібник експлуатації.

Попередня підготовка: Перевірте, що використовуваний веб-оглядач задовольняє мінімальні системні вимоги, а інтернет-канал забезпечує стійке з'єднання зі швидкістю принаймні 10 Мбіт/с. Переконайтеся, що у налаштуваннях браузера дозволено відкриття спливаючих вікон (pop-up) для тестової сторінки.

Поступове освоєння функціоналу: Рекомендується розпочати з ручного тестування одиничного зображення для знайомства з інтерфейсом та логікою роботи. Після опанування базових операцій перейдіть до пакетного автоматизованого тестування на невеликих наборах (5-10 графічних файлів). Детально вивчіть довідкові матеріали щодо інтерпретації кожної з реєстрованих метрик.

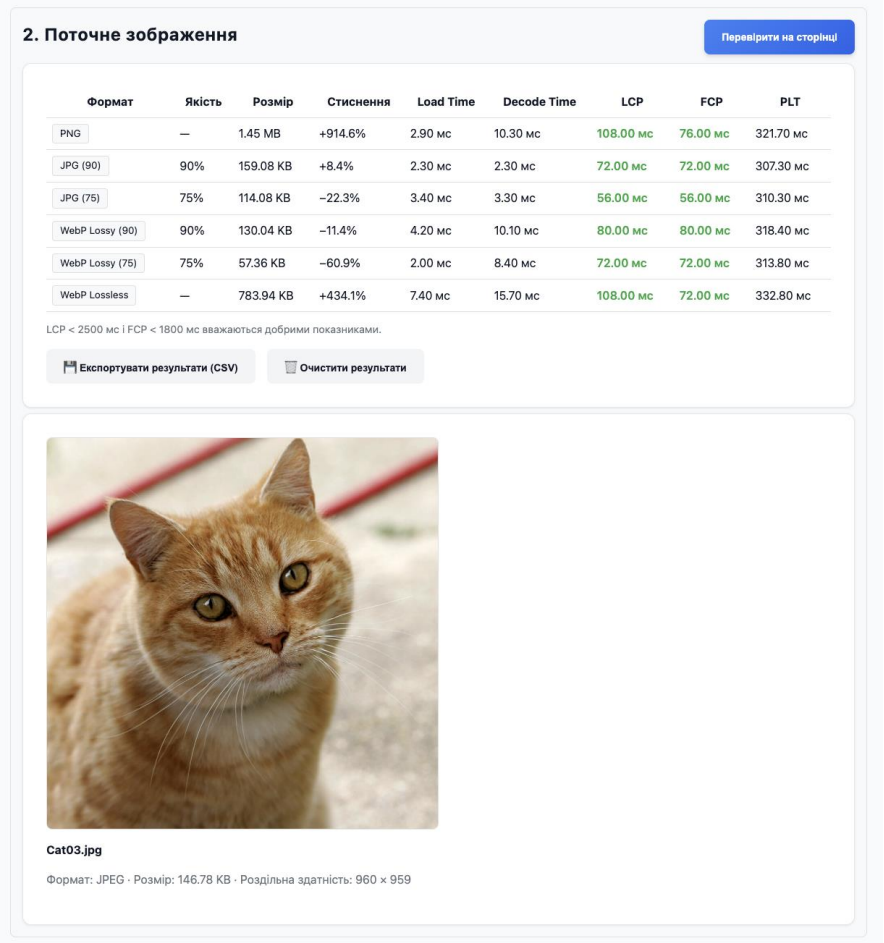


Рисунок 7.1 - Тестування одного файлу на головній сторінці

7.2 Опис еталонної перевірки

Проведення вимірювань: Після успішного завантаження файлу натисніть кнопку "Запустити тестування всіх форматів". Дозвольте браузеру відкрити допоміжне тестове вікно (pop-up). Зачекайте на завершення циклу вимірювань для всіх шести форматів: PNG, JPG з якістю 90%, JPG з якістю 75%, WebP Lossy 90%, WebP Lossy 75% та WebP Lossless.

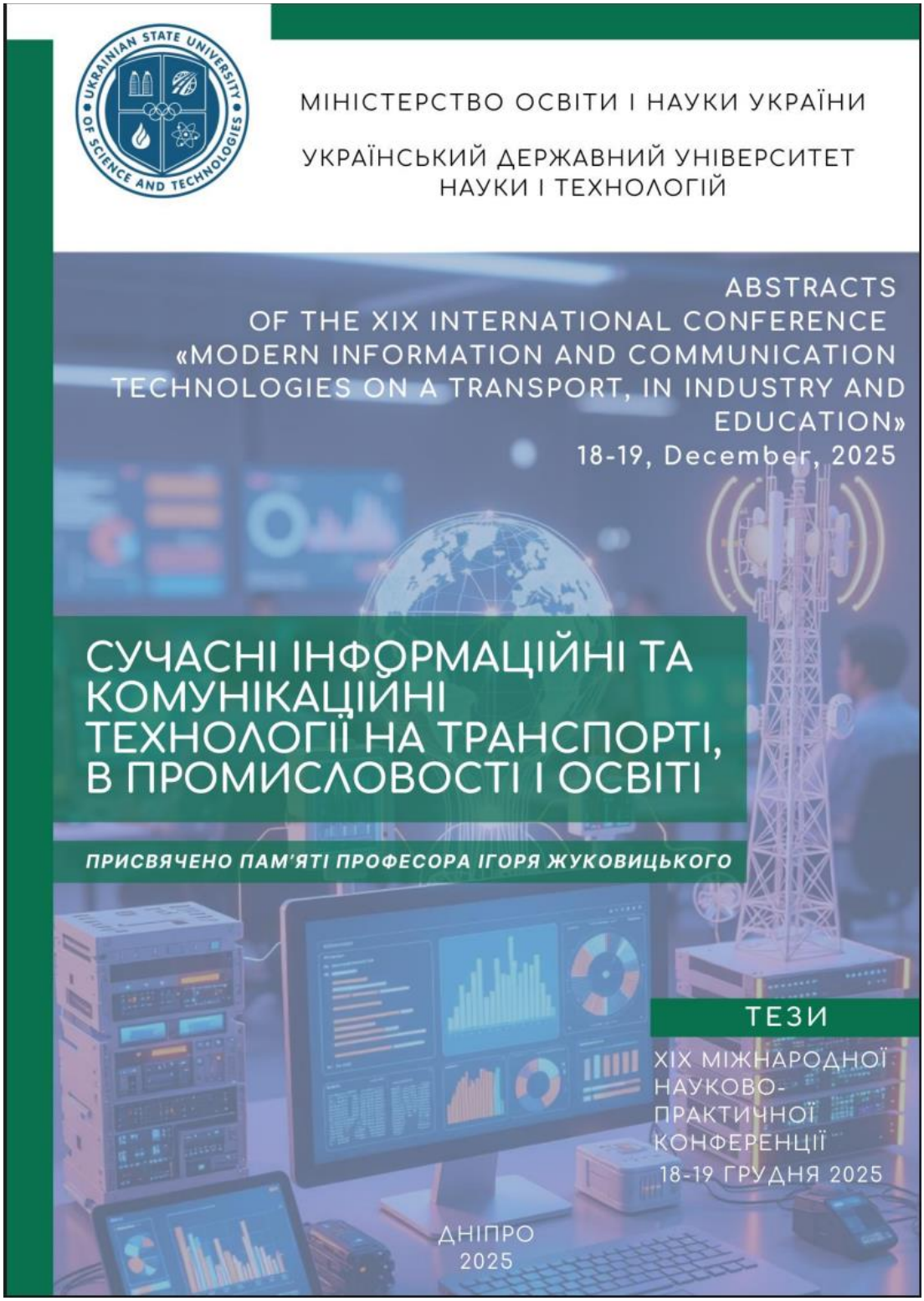
Приблизні показники для пейзажного зображення 1920×1080 (вага 2-3 МБ):

- PNG: тривалість завантаження 200-300 мілісекунд, час декодування 100-150 мс, показник LCP 300-400 мс, фізичний розмір 4-6 мегабайт;
- JPG 90%: завантаження 100-150 мс, декодування 50-80 мс, LCP 150-250 мс, розмір 1-2 МБ;
- JPG 75%: завантаження 50-100 мс, декодування 40-60 мс, LCP 100-200 мс, розмір 0.5-1 МБ;

- WebP Lossy 90%: завантаження 80-120 мс, декодування 60-90 мс, LCP 140-220 мс, розмір 0.8-1.5 МБ;
- WebP Lossy 75%: завантаження 40-80 мс, декодування 50-70 мс, LCP 90-170 мс, розмір 0.4-0.8 МБ; WebP Lossless: завантаження 150-250 мс, декодування 80-120 мс, LCP 250-350 мс, розмір 3-5 МБ.

Інтерпретація отриманих даних: Формати WebP із втратами (Lossy) демонструють найкраще співвідношення між візуальною якістю та обсягом файлу. JPG з компресією 75% забезпечує мінімальний розмір при збереженні прийнятної якості. PNG характеризується максимальним розміром, але гарантує абсолютну точність відтворення оригіналу без будь-яких втрат. Експортуйте зібрані дані у форматі CSV для поглибленого статистичного аналізу

ДОДАТОК Г


UKRAINIAN STATE UNIVERSITY OF SCIENCE AND TECHNOLOGIES
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ
ABSTRACTS
OF THE XIX INTERNATIONAL CONFERENCE
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025
СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ
ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО
ТЕЗИ
XIX МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025
ДНІПРО
2025

Дослідження продуктивності роботи веб-додатків з різними форматами зображень

Посмітюха М.О., Гришечкіна Т.С, Український державний університет науки і технологій, Україна

Зображення займають до 41% загального розміру переданих даних на сучасних веб-сайтах, це робить їх ключовим фактором, який впливає на продуктивність веб-додатків. У контексті мого дослідження продуктивність (або ефективність, швидкість) – це оцінка за кількома ключовими показниками, які демонструють, як швидко та якісно контент завантажується і відображається для користувача. Ключові метрики користувацького досвіду представлені компанією Google як Web Vitals. Існування різних графічних форматів (JPEG, PNG, WebP, тощо) з різними алгоритмами стиснення вимагає кількісного дослідження їхнього впливу на швидкість завантаження, декодування та рендерингу для обґрунтування вибору формату.

Метою дослідження є порівняння показників продуктивності веб-додатків при використанні зображень із різними форматами (PNG, JPG, WebP для статичних та gif, swf та webp для покадрової анімації) та встановлення статистичної відмінності.

Умови реалізації дослідження:

- автоматизований збір Web Vitals та інших показників у контрольованому, ізольованому середовищі різних браузерів
- автоматична конвертація зображень у тестовий набір даних (png, jpeg 90/75%, webp Lossy 90/75%, webp Lossless, gif, swf та WebP animated).
- використання ізольованого середовище для тестування (Load Time, Decode Time, LCP, FCP, PLT, File Size) на окремій test-page.html, що унеможливлює вплив сторонніх ресурсів.
- пакетне тестування: тестування з можливістю налаштування кількості ітерацій для отримання статистично вірних (медіанних та найбільших/найменших) значень.
- фіксація результатів. Аналіз статистики, що включає описову статистику та критерій Вілкоксона для оцінки значущості різниці показників продуктивності.

У ході експерименту порівнянні продуктивності сторінок із зображеннями різних форматів. Пакетний режим роботи веб-додатка автоматично розраховує показники для вибірки та формує загальну статистику.

Аналіз на тестових даних показує систематичні відмінності у показниках між різними форматами.

Наприклад, порівняння формату WebP Lossy та PNG із аналогічними відсотком втрати при стисканні показало такі медіанні значення (на прикладі статичної графіки):

- Load Time: 245 мс для PNG проти 98 мс для WebP при ідеальному показнику < 100 мс.
- LCP (час завантаження основного контенту сторінки, його готовність для взаємодії з користувачем. PNG 356 мс проти WebP 187 мс.
- Розмір файлу: PNG 5.2 MB проти 1.1 MB для WebP, при тому що конвертували однакове зображення (SSIM – показник структурної схожості>90).

Як результат, можна стверджувати, що формат зображення є важливим фактором, який впливає на показники продуктивності веб-додатків, такі, як час завантаження ресурсу, рендеринг (LCP) та обсяг переданого трафіку. Відповідне програмне забезпечення є ефективним інструментом для кількісної оцінки та порівняння цих показників.