

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ВЕБОРІСНТОВАНА СИСТЕМА ГЕНЕРАЦІЇ ВІЗУАЛЬНИХ
КАРТ СПОГАДІВ ТА ПОДОРОЖЕЙ

Виконав: студент 2 курсу, групи КНТ-514м
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні системи та мережі

СІЛІН М.Ю.

(ПРІЗВИЩЕ та ініціали)

Керівник ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ХАРИТОНОВ О.Б.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні системи та мережі
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“ 15 ” жовтня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

СІЛІНА Максима Юрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Веборієнтована система генерації візуальних карт спогадів та подорожей

керівник проєкту (роботи) кан. техн. наук, доцент, ГОЛУБ Тетяна Василівна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “ 30 ” жовтня 2025 року № 491

2. Строк подання студентом проєкту (роботи) 23 грудня 2025 року

3. Вихідні дані до проєкту (роботи) наявні методи та алгоритми обробки геолокаційних даних, час обробки 250 фотографій має не перевищувати 10 секунд, точність визначення локацій не менш 95%, підтримка форматів JPEG, PNG, HEIC

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Аналіз технічного завдання, дослідження наявних методів обробки геолокаційних даних з мультимедійних файлів;

2) Розробка структури та алгоритму роботи веборієнтованої системи;

3) Реалізація веборієнтованої системи;

4) Дослідження системи генерації візуальних карт.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ГОЛУБ Т.В., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.11.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів обробки геолокаційних даних, виділення недоліків та переваг	05.10.2025 р.	
2	Розробка структури системи	10.10.2025 р.	
3	Розробка алгоритму обробки метаданих	15.11.2025 р.	
4	Реалізація модулів системи	25.11.2025 р.	
5	Дослідження системи генерації карт	30.11.2025 р.	
6	Оформлення отриманих результатів у ПЗ	05.12.2025 р.	
7	Оформлення графічного матеріалу	10.12.2025 р.	
8	Оформлення допоміжного матеріалу	15.12.2025 р.	

Студент **Максим СІЛІН**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи) **Тетяна ГОЛУБ**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 93 стор., 31 рис., 12 табл., 28 джерел.

DBSCAN, EXIF МЕТАДАНИ, GEOJSON, MAPBOX GL JS, MONGODB, NODE.JS, REACT, ВЕБОРІЄНТОВАНА СИСТЕМА, ВІЗУАЛІЗАЦІЯ ПОДОРОЖЕЙ, ГЕОЛОКАЦІЙНІ ДАНІ, КАРТОГРАФУВАННЯ, КЛАСТЕРИЗАЦІЯ

Об'єкт дослідження - процес автоматизованого документування та візуалізації подорожей.

Предмет дослідження - методи та засоби створення веборієнтованої системи для генерації інтерактивних візуальних карт спогадів.

Мета роботи - розробка веборієнтованої системи для автоматизованої генерації інтерактивних візуальних карт подорожей на основі геолокаційних метаданих з використанням алгоритму DBSCAN.

У першому розділі проведено аналіз проблематики документування цифрових подорожей, досліджено технічні основи EXIF метаданих, виконано порівняльний аналіз існуючих рішень Google My Maps, Polarsteps, Wanderlog та визначено методи кластеризації геопросторових даних.

У другому розділі розроблено концептуальну модель системи з тришаровою архітектурою, спроектовано архітектуру компонентів на базі React-Node.js-MongoDB.

Третій розділ спроектовано користувацький інтерфейс з багатокроковим процесом створення подорожей, розроблено архітектуру компонентів на React з TypeScript, реалізовано візуалізацію карти з Mapbox GL JS.

У четвертому розділі визначено методологію експериментального дослідження, проведено дослідження алгоритмів кластеризації з варіюванням параметрів.

ABSTRACT

Explanatory note to the master's work: 93 p., 31 fig., 12 tabl., 28 sources.

CARTOGRAPHY, CLUSTERING, DBSCAN, EXIF METADATA, GEOJSON, GEOLOCATION DATA, MAPBOX GL JS, MONGODB, NODE.JS, REACT, TRAVEL VISUALIZATION, WEB-BASED SYSTEM

Research object - the process of automated documentation and visualization of travels using geolocation metadata from multimedia files.

Research subject - methods and tools for creating a web-based system for generating interactive visual memory and travel maps based on geospatial analysis algorithms.

Research goal - development of a web-based system for automated generation of interactive visual travel maps based on geolocation metadata using DBSCAN algorithm for intelligent location clustering.

The first section conducts analysis of digital travel documentation challenges, examines technical foundations of EXIF metadata, performs comparative analysis of existing solutions Google My Maps, Polarsteps, Wanderlog, and identifies geospatial data clustering methods and cartographic libraries for web visualization.

The second section develops a conceptual model of the system with three-tier architecture, designs component architecture based on React-Node.js-MongoDB stack, models data flows for media file processing, designs data structures with GeoJSON support.

The third section is dedicated to system implementation. User interface with multi-step trip creation process is designed, component architecture based on React with TypeScript is developed.

The fourth section defines experimental research methodology, conducts clustering algorithm research with epsilon parameter variation.

ЗМІСТ

Вступ.....	7
1 Аналітична частина.....	9
1.1 Огляд проблематики документування цифрових подорожей	9
1.3 Порівняльний аналіз існуючих програмних рішень.....	17
1.4 Методи кластеризації геопросторових даних	18
1.5 Картографічні бібліотеки для вебвізуалізації	22
1.6 Постановка задачі дослідження.....	25
2 Модельна частина	27
2.1 Концептуальна модель системи.....	27
2.2 Архітектура програмних компонентів	30
2.3 Моделювання потоків даних.....	33
2.4 Проектування структур даних	36
2.5 Алгоритмічне забезпечення системи	39
3 Реалізація веборієнтованої системи	42
3.1 Проектування користувацького інтерфейсу.....	42
3.2 Архітектура інтерфейсних компонентів	43
3.3 Візуалізація інтерактивної карти подорожі.....	45
3.4 Реалізація серверної частини системи	45
3.5 Розгортання та конфігурація апаратної частини системи.....	47
3.6 Інтерфейс веборієнтованої системи	48
4 Дослідницька частина	57
4.1 Методологія експериментального дослідження	57
4.2 Результати дослідження алгоритмів кластеризації	59
4.3 Оцінка якості користувацького досвіду.....	61
Висновки	62
Перелік джерел посилання	64
Додаток А. Код веборієнтованої системи.....	68
Додаток Б. Слайди презентації	88

ВСТУП

Актуальність теми. Стрімкий розвиток цифрових технологій та повсюдне поширення мобільних пристроїв із вбудованими GPS-модулями призвели до формування нової культури документування подорожей. За даними міжнародних досліджень, понад 65% мандрівників активно використовують смартфони та цифрові камери для фіксації вражень під час подорожей, створюючи щорічно понад 1,4 трильйона геотегованих фотографій. Водночас традиційні методи організації цифрових спогадів у вигляді хронологічних галерей або папкових структур не розкривають просторовий контекст подорожей та не забезпечують цілісного розуміння географічної історії переміщень користувачів [1].

Сучасні користувачі потребують інструментів, які дозволяють не лише зберігати цифрові спогади, але й автоматично перетворювати їх на інтерактивні візуальні історії з просторовим контекстом. Існуючі комерційні рішення, такі як Google My Maps, Polarsteps та Wanderlog, демонструють обмежений функціонал автоматизації обробки геолокаційних даних, вимагають значних зусиль для ручного введення інформації про локації або не підтримують повноцінну інтеграцію мультимедійного контенту різних форматів [2]. Більшість платформ орієнтована переважно на соціальний обмін контентом і не надає користувачам повноцінних інструментів для персональної аналітичної роботи з даними про свої подорожі [3].

Ключові проблеми, що потребують вирішення, включають відсутність ефективних алгоритмів автоматизованого вилучення просторово-часової інформації з мультимедійних файлів різних форматів, складність побудови візуально привабливих та інформативних карт маршрутів з урахуванням географічних особливостей, брак інтелектуальних інструментів для аналізу географічних патернів подорожей та обмежені можливості персоналізації візуалізацій відповідно до естетичних переваг та специфічних потреб

користувачів [4]. Наявні алгоритми кластеризації геопросторових даних не завжди адекватно враховують часову компоненту та специфіку організації подорожей, а методи побудови оптимальних маршрутів потребують адаптації до завдань візуалізації спогадів з урахуванням як географічної близькості, так і хронологічної послідовності відвідування локацій.

Розробка веборієнтованої системи для автоматизованої генерації візуальних карт спогадів та подорожей на основі сучасних алгоритмів машинного навчання та геопросторового аналізу дозволить користувачам ефективно документувати свої подорожі, створювати інтерактивні цифрові архіви з повноцінним просторовим контекстом, отримувати аналітичні insights про свою географічну активність, виявляти цікаві патерни у власних подорожах та формувати персоналізовані мультимедійні історії з мінімальними часовими витратами на організацію контенту.

Мета і задачі дослідження. Метою роботи є розробка веборієнтованої системи для автоматизованої генерації інтерактивних візуальних карт спогадів та подорожей на основі геолокаційних метаданих з мультимедійних файлів користувача з використанням алгоритмів машинного навчання для інтелектуальної кластеризації локацій, побудови оптимальних маршрутів та створення анімованих візуалізацій переміщень з можливістю експорту готових карт у різних форматах.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Огляд проблематики документування цифрових подорожей

Сучасні технології мобільної фотографії та геолокації трансформували підходи до документування подорожей створюючи нові можливості для збереження та організації спогадів з географічним контекстом. За даними дослідження Rodriguez Chen та Wang більше шістдесяти п'яти відсотків мандрівників активно використовують смартфони та цифрові камери для фіксації вражень під час подорожей створюючи щорічно мільярди геотегованих фотографій які містять метадані про координати місця зйомки час та технічні параметри камери. Традиційні методи організації цифрових архівів у вигляді хронологічних галерей або папкових структур не дозволяють ефективно використовувати просторовий контекст закладений у геолокаційні метадані та не забезпечують цілісного розуміння географічної історії переміщень під час подорожі [5].

Рисунок 1.1 ілюструє стандартний процес документування подорожі сучасним мандрівником від початкового планування через фіксацію вражень під час подорожі до постобробки та організації зібраних матеріалів після повернення додому.

Ключові проблеми традиційних підходів включають відсутність автоматизованого вилучення та аналізу просторово-часової інформації з мультимедійних файлів що вимагає ручного введення локацій користувачем при створенні карт подорожей [6]. Складність побудови візуально привабливих та інформативних картографічних представлень маршрутів потребує спеціалізованих навичок роботи з геоінформаційними системами які не є доступними пересічним користувачам. Брак інструментів для інтелектуального аналізу географічних патернів подорожей таких як кластеризація відвідуваних локацій? виявлення ключових точок маршруту та обчислення статистичних

показників переміщень унеможливилює отримання точних збережень про власну туристичну активність.



Рисунок 1.1 - Процес документування подорожі сучасним мандрівником

Обмежені можливості персоналізації візуалізацій відповідно до естетичних переваг та специфічних потреб різних типів мандрівників від любителів міських екскурсій до прихильників активного відпочинку на природі знижують цінність існуючих рішень для широкої аудиторії.

Сучасний ринок програмних рішень для візуалізації подорожей представлений декількома категоріями продуктів, що відрізняються за функціональністю, рівнем автоматизації та цільовою аудиторією. Комплексний аналіз провідних платформ дозволяє виявити переваги та обмеження існуючих підходів до генерації візуальних карт спогадів.

Google Photos представляє собою хмарний сервіс для зберігання та організації фотографій, що автоматично виділяє геолокаційні дані з EXIF метаданих та відображає знімки на карті світу через вбудовану функцію "Переглянути на карті". Система автоматично групує фотографії за локаціями, використовуючи власні алгоритми кластеризації, та створює тематичні альбоми

подорожей з хронологічною послідовністю відвіданих місць. Однак платформа має суттєві обмеження щодо налаштування параметрів візуалізації, оскільки користувачі не можуть контролювати радіус кластеризації, змінювати стилі маркерів або експортувати дані у відкритих форматах для подальшого використання в інших застосунках.

Apple Photos інтегрована у екосистему пристроїв Apple та забезпечує автоматичну організацію знімків за місцями через вкладку "Місця", де фотографії відображаються на інтерактивній карті з використанням картографічних даних Apple Maps. Алгоритм інтелектуальної кластеризації автоматично виявляє значущі локації на основі тривалості перебування та кількості зроблених фотографій, створюючи візуальні історії Memories з географічною прив'язкою. Недоліками платформи є обмеженість екосистемою Apple, відсутність веброзділу для повноцінного доступу з інших пристроїв, закрита архітектура без можливості інтеграції з іншими сервісами через API та обмежені функції експорту.

Polarsteps позиціонується як спеціалізована платформа для автоматичного трекінгу подорожей, що працює у фоновому режимі на мобільних пристроях, збираючи GPS треки, відзначаючи відвідані країни та автоматично формуючи маршрути на карті з хронологією переміщень. Додаток дозволяє користувачам додавати фотографії, текстові нотатки та враження до кожної локації, створюючи персоналізовані тревел блоги з можливістю обміну враженнями через соціальну мережу застосунку. Проте Polarsteps має критичні обмеження через необхідність постійного фонового трекінгу GPS протягом подорожі, що суттєво зменшує час автономної роботи мобільного пристрою, відсутність можливості ретроспективної обробки архівних фотографій без GPS треків, обов'язкову реєстрацію облікового запису без автономного режиму роботи.

Triplt функціонує як планувальник та органайзер подорожей з можливістю автоматичного імпорту інформації про бронювання з електронної пошти, створюючи єдиний маршрутний документ з усіма деталями поїздки.

Система відображає заплановані локації на карті з прив'язкою до конкретних дат та часу відвідування, підтримує синхронізацію з календарями та надсилає своєчасні нагадування про події подорожі. Основні недоліки включають фокус на плануванні майбутніх подорожей з обмеженою підтримкою документування завершених поїздок, відсутність автоматичної обробки фотографій та вилучення геолокацій з медіафайлів.

Komoot орієнтований на активних мандрівників та спортсменів, надаючи детальні топографічні карти з маршрутами для піших прогулянок, велосипедних поїздок та інших активностей на природі. Платформа дозволяє планувати маршрути з урахуванням складності рельєфу, типу поверхні та фізичної підготовки, записувати GPS треки під час активностей та обмінюватися досвідом з іншими користувачами. Обмеження Komoot полягають у спеціалізації на активному відпочинку з мінімальною підтримкою міських подорожей та культурного туризму. Детальне порівняння функціональних можливостей систем візуалізації подорожей наведено в таблиці 1.1.

Аналіз виявлених обмежень існуючих рішень демонструє відсутність на ринку комплексної системи, що поєднує автоматичну обробку архівних медіафайлів без необхідності попереднього GPS трекінгу, інтелектуальну кластеризацію локацій з налаштовуваними параметрами відповідно до характеру подорожі, гнучкі можливості персоналізації візуалізації з контролем стилів маркерів, кольорових схем та рівнів деталізації. Це обґрунтовує необхідність розробки веборієнтованої системи генерації візуальних карт

Таблиця 1.1 – Детальне порівняння функціональних можливостей систем візуалізації подорожей

Критерій	Google Photos	Apple Photos	Polarsteps	Triplt	Komoot
Автоматичне вилучення геолокацій	Так, з EXIF	Так, з EXIF	Так, GPS трек	Ні	Так, GPS трек
Кластеризація локацій	Автоматична	Інтелектуальна	По точках маршруту	Відсутня	По активностях
Налаштування візуалізації	Мінімальні	Відсутні	Базові	Мінімальні	Детальні
Підтримка архівних фото	Повна	Повна	Обмежена	Відсутня	Відсутня
Експорт даних	KML, JSON	Обмежений	PDF, фотокнига	PDF маршрут	GPX, KML
Автономний режим	Так	Так	Обмежений	Ні	Так
Веборієнтованість	Повна	Часткова	Повна	Повна	Повна
Вартість	Безкоштовно	Безкоштовно	Freemium	Freemium	Freemium
Відкрите API	Обмежене	Відсутнє	Відсутнє	Обмежене	Відсутнє
Інтеграція з іншими сервісами	Висока	Низька	Низька	Середня	Середня

1.2 Технічні основи геолокаційних метаданих

Стандарт EXIF 2.13.2, визначений організацією Japan Electronics and Information Technology Industries Association у 2019 році, встановлює специфікацію для зберігання метаданих у файлах цифрових зображень включно з геолокаційною інформацією отриманою від GPS приймачів мобільних пристроїв. Структура GPS даних у EXIF включає набір тегів, які зберігають координати широти та довготи у форматі градуси, мінути, секунди з референсними тегами, що вказують на півкулю північ або південь для широти, та схід або захід для довготи. Додаткові теги містять висоту над рівнем моря у метрах, напрямок руху камери у градусах від північ, швидкість переміщення у кілометрах на годину та часову мітку зйомки у форматі UTC для синхронізації з іншими джерелами геопросторових даних [7].

Рисунок 1.2 демонструє структуру EXIF метаданих у JPEG файлі з детальним розташуванням GPS тегів у ієрархії Image File Directory згідно зі специфікацією TIFF.

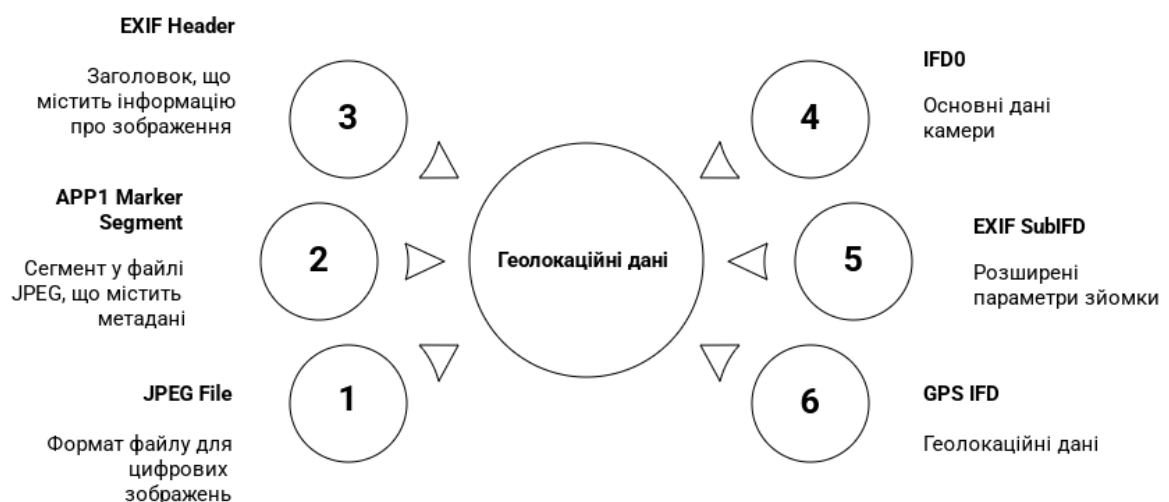


Рисунок 1.2 - Структура EXIF метаданих у JPEG файлі

Бібліотеки для обробки EXIF метаданих реалізовані для більшості популярних мов програмування забезпечуючи програмний доступ до

геолокаційної інформації без необхідності парсингу низькорівневої бінарної структури файлів вручну. Для JavaScript середовища найпоширенішою є бібліотека ExifReader яка підтримує читання метаданих з форматів JPEG PNG HEIC WebP та TIFF з автоматичною конвертацією координат з формату градуси мінуси секунди у десяткові градуси для математичних обчислень. У Python екосистемі бібліотека PIL Pillow надає клас Image з методом getexif для вилучення словника всіх EXIF тегів з додатковими утилітами для декодування складних структур даних таких як GPS координати у зручному для використання форматі чисел з плаваючою комою [8].

Обробка EXIF метаданих у мультимедійних файлах є критичним компонентом автоматизованого вилучення геолокаційної інформації з цифрових спогадів подорожей. Вибір методу та бібліотеки для роботи з EXIF даними визначається технологічним стеком системи, вимогами до продуктивності та необхідністю підтримки різноманітних форматів файлів.

ExifReader представляє собою чисту JavaScript бібліотеку без залежностей, що працює безпосередньо у браузері для читання EXIF даних з файлів зображень форматів JPEG та TIFF. Бібліотека забезпечує швидкий парсинг метаданих на стороні клієнта без необхідності завантаження файлів на сервер, що зменшує навантаження на мережу та зберігає приватність користувацьких даних. ExifReader підтримує всі стандартні EXIF теги включно з GPS координатами, датою та часом зйомки, параметрами камери. Основні переваги включають відсутність серверних залежностей, високу швидкість обробки та невеликий розмір близько п'ятнадцяти кілобайт.

Exif-js є більш старою альтернативою з широкою підтримкою різних браузерів включно з застарілими версіями Internet Explorer, що забезпечує сумісність у корпоративних середовищах. Бібліотека надає простий API для читання EXIF тегів через callback функції. Проте exif-js має значні обмеження через застарілу кодову базу без активної підтримки розробників, відсутність підтримки сучасних форматів HEIC та відеофайлів з геолокаційними метаданими.

Piexifjs розширює функціональність базових EXIF читачів через підтримку не тільки читання але й запису та модифікації метаданих безпосередньо у браузері, що дозволяє реалізувати функції видалення геолокацій для захисту приватності або додавання координат до фотографій. Обмеження включають більший розмір бібліотеки близько п'ятдесяти кілобайт, складніший API та можливість пошкодження файлів при некоректному записі.

ExifTool від Phil Harvey є найбільш потужним консольним інструментом для роботи з метаданими, підтримуючи понад п'ятсот різних форматів файлів. Інструмент може бути інтегрований у серверну частину системи через виконання команд операційної системи. Переваги включають абсолютну надійність, можливість пакетної обробки та підтримку всіх форматів. Недоліки полягають у необхідності встановлення Perl інтерпретатора, повільнішій швидкості обробки та складності інтеграції (табл. 1.2).

Таблиця 1.2 – Порівняння методів та бібліотек обробки EXIF метаданих

Критерій	ExifReader	exif-js	Piexifjs	ExifTool
Середовище виконання	Браузер	Браузер	Браузер	Сервер
Підтримка форматів	JPEG, TIFF	JPEG	JPEG	500+ форматів
Можливість запису	Ні	Ні	Так	Так
Розмір бібліотеки	15 KB	8 KB	50 KB	4 MB
Швидкість обробки	Висока	Середня	Середня	Низька
Підтримка HEIC	Ні	Ні	Ні	Так
Підтримка відео	Ні	Ні	Ні	Так
Активна підтримка	Так	Ні	Так	Так
Складність інтеграції	Низька	Низька	Середня	Висока
Безпека даних	Клієнт	Клієнт	Клієнт	Сервер

1.3 Порівняльний аналіз існуючих програмних рішень

Наразі на ринку присутня низка програмних рішень для створення карт подорожей які можна класифікувати за декількома ключовими характеристиками включно з рівнем автоматизації, джерелами геолокаційних даних, можливостями візуалізації та функціями експорту готових результатів. Таблиця 1.3 представляє детальне порівняння чотирьох найпопулярніших платформ за ключовими функціональними характеристиками.

Таблиця 1.3 - Порівняльний аналіз існуючих рішень для картографування подорожей

Платформа	Джерело даних	Автоматизація	Візуалізація	Експорт
Google My Maps	Ручне введення координат та адрес, пошук місць Google	Низька. Ручне додавання кожної локації, малий маршрут	Базові маркери та лінії маршрутів, обмежена персоналізація стилів	KML, спільне посилання
Polarsteps	Фоновий GPS-трекінг у реальному часі під час подорожі	Висока для поточних подорожей. Не працює з архівними фото	Анімовані маршрути, timeline, інтеграція фото з автовизначенням локацій	PDF travelog, спільне посилання
Wanderlog	Інтеграція TripAdvisor, Google Places, Yelp для пошуку POI	Середня. Фокус на плануванні майбутніх подорожей, не аналізі архіву	Маршрути з оптимізацією, календар подій, бюджет, чеклісти	Google Sheets, спільне редагування
Roadtrippers	Ручне планування, рекомендації визначних місць уздовж маршруту	Низька. Орієнтований на планування	Дорожні карти з POI, розрахунок відстаней та часу в дорозі	Мобільний додаток, синхронізація

Аналіз функціональних можливостей існуючих платформ виявляє відсутність комплексних рішень, які поєднують автоматичне вилучення геолокацій з архівних фотографій, інтелектуальну кластеризацію локацій, побудову оптимальних маршрутів та гнучкі можливості візуалізації з експортом у різноманітні формати у єдиній системі. Google My Maps надає базову функціональність картографування, але вимагає трудомісткого ручного введення всіх локацій, що робить створення карт великих подорожей з сотнями фотографій непрактичним для пересічних користувачів. Polarsteps забезпечує високий рівень автоматизації через фоновий GPS-трекінг, але принципово не може працювати з архівними фотоколекціями минулих подорожей, оскільки потребує активного відстеження під час подорожі, що обмежує його застосування для документування вже здійснених поїздок [9].

1.4 Методи кластеризації геопросторових даних

Алгоритми кластеризації геопросторових даних призначені для автоматичного виявлення груп близько розташованих точок у просторі що відповідають локаціям тривалого перебування під час подорожі на противагу транзитним точкам уздовж маршруту переміщення. Таблиця 1.4 представляє порівняльний аналіз чотирьох основних алгоритмів кластеризації за критеріями придатності для задачі групування фотографій подорожей за місцями відвідування [10].

Задача кластеризації геолокаційних точок подорожей вимагає врахування специфічних характеристик просторових даних, таких як нерівномірна щільність розподілу точок, присутність шумових координат від помилкових GPS сигналів, а також необхідність виявлення кластерів довільної форми що відповідають природній геометрії локацій відвідування.

K-means представляє класичний алгоритм кластеризації, що розділяє набір точок на k заздалегідь визначених кластерів шляхом ітераційної мінімізації суми квадратів відстаней від точок до центрів кластерів. Алгоритм має обчислювальну складність $O(n \cdot k \cdot i)$, що робить його придатним для обробки великих наборів даних у режимі реального часу. Основні недоліки включають необхідність апіорного визначення кількості кластерів k , чутливість до початкової ініціалізації, припущення про сферичну форму кластерів та відсутність механізму виявлення шумових точок.

Таблиця 1.4 - Порівняльний аналіз алгоритмів кластеризації для геопросторових даних

Алгоритм	Параметри	Складність	Переваги	Недоліки
DBSCAN	eps (радіус околиці), minPoints (мін. щільність)	$O(n \log n)$ з просторовим індексом	Не потребує знання k -сті кластерів, виявляє шум, довільна форма	Чутливий до параметрів, складність налаштування eps
K-means	k (кількість кластерів)	$O(n \cdot k \cdot i)$, де i - ітерації	Простота реалізації, швидка збіжність, добре масштабується	Потрібно знати k заздалегідь, тільки сферичні кластери, чутливий до викидів
OPTICS	minPoints, reachability plot для вибору кластерів	$O(n^2)$ без оптимізації	Ієрархічна структура, не потребує eps, візуальний аналіз	Висока обчислювальна складність, складність інтерпретації
Mean Shift	bandwidth (ширина вікна ядра)	$O(n^2)$ на ітерацію	Автовизначення k -сті, не потребує форми, робастність	Дуже повільний, не масштабується, складність вибору bandwidth

Рисунок 1.3 ілюструє процес роботи алгоритму DBSCAN на прикладі кластеризації геолокацій фотографій міської подорожі з візуалізацією виявлених кластерів та шумових точок.

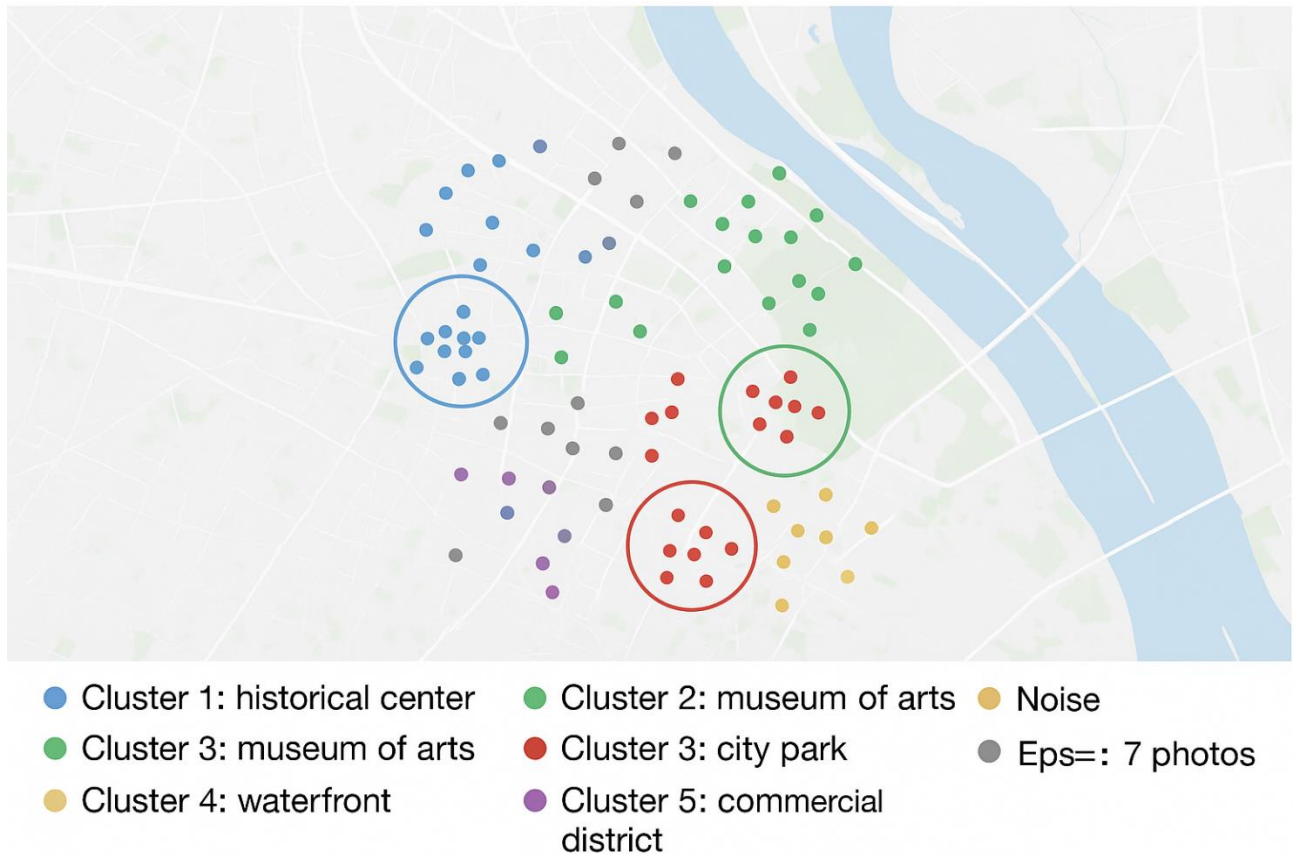


Рисунок 1.3 - Візуалізація роботи алгоритму DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) виявляє кластери як області високої щільності точок відокремлені областями низької щільності з автоматичним виявленням шумових точок. Алгоритм визначається двома параметрами: epsilon (радіус коло точки) та minPoints (мінімальна кількість точок у колі). Переваги включають автоматичне визначення кількості кластерів, здатність виявляти кластери довільної форми, вбудований механізм виявлення шуму та стійкість до викидів. Обмеження полягають у чутливості до вибору параметрів та складності обробки областей з різною щільністю.

OPTICS (Ordering Points To Identify the Clustering Structure) є розширенням DBSCAN, що створює упорядкування точок, яке містить

інформацію про структуру кластеризації при всіх можливих значеннях ϵ . Алгоритм будує reachability plot, що візуально виявляє структуру кластерів. OPTICS має переваги через можливість виявлення ієрархічної структури кластерів та адаптивність до областей з різною щільністю. Недоліки включають вищу обчислювальну складність та складніший процес вилучення фінальних кластерів.

Mean Shift є непараметричним алгоритмом, що ітеративно зміщує кандидати у центри кластерів до областей з найвищою щільністю точок використовуючи ядерну оцінку щільності. Переваги включають відсутність припущень про форму кластерів та автоматичне визначення кількості кластерів. Обмеження полягають у критичній чутливості до параметра bandwidth та високій обчислювальній складності $O(n^2)$.

Hierarchical clustering створює ієрархію кластерів у формі дерева дендрограми через послідовне об'єднання найближчих кластерів. Переваги включають створення повної ієрархії та різні методи обчислення відстаней. Недоліки полягають у обчислювальній складності $O(n^2 \log n)$ та чутливості до шумових точок (табл. 1.5).

Таблиця 1.5 – Детальне порівняння алгоритмів кластеризації для геопросторових даних подорожей

Критерій	K-means	DBSCAN	OPTICS	Mean Shift	Hierarchical
Автовизначення кількості	Ні	Так	Так	Так	Так
Обробка шуму	Ні	Так	Так	Частково	Ні
Форма кластерів	Сферична	Довільна	Довільна	Довільна	Залежить
Складність	$O(n \cdot k \cdot i)$	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(n^2 \log n)$
Різна щільність	Ні	Обмежено	Так	Обмежено	Залежить
Параметри	$k, \text{max_iter}$	ϵ, minPts	ϵ, minPts	bandwidth	linkage
Ієрархічність	Ні	Ні	Так	Ні	Так
Детермінізм	Ні	Так	Так	Залежить	Так
Для подорожей	Обмежено	Відмінно	Дуже добре	Задовільно	Добре

Для задачі кластеризації геолокацій подорожей оптимальним вибором є DBSCAN завдяки поєднанню автоматичного визначення кількості кластерів, здатності виявляти структури довільної форми, вбудованого механізму фільтрації шуму та прийнятної обчислювальної складності. Параметр `epsilon` рекомендується встановлювати у діапазоні від ста до п'ятисот метрів залежно від типу подорожі, а `minPoints` у діапазоні від трьох до п'яти точок.

1.5 Картографічні бібліотеки для вебвізуалізації

Вибір картографічної бібліотеки для вебвізуалізації визначається комплексом технічних та функціональних вимог включно з продуктивністю рендерингу великої кількості об'єктів підтримкою інтерактивних елементів, управління можливостями кастомізації візуального представлення та ліцензійними умовами використання. Таблиця 1.6 представляє детальне порівняння трьох провідних JavaScript бібліотек для інтерактивної картографії за критеріями релевантними для розробки системи візуалізації подорожей [12].

Mapbox GL JS обраний як основна бібліотека для реалізації картографічної візуалізації завдяки унікальному поєднанню високої продуктивності WebGL рендерингу, що забезпечує плавні шістдесят кадрів на секунду навіть при відображенні тисяч об'єктів одночасно та гнучких можливостей кастомізації візуального представлення через Mapbox Style Specification, який дозволяє програмно контролювати кожен аспект відображення карти від кольорів доріг та водойм до шрифтів, підписів та символів точок інтересу [13].

Вибір бібліотеки для відображення інтерактивних карт у веббраузері визначається комплексом технічних факторів включно з продуктивністю рендерингу, підтримкою різних типів даних, можливостями кастомізації

візуального стилю, екосистемою плагінів та ліцензійними умовами використання.

Таблиця 1.6 - Порівняння JavaScript бібліотек для картографічної візуалізації

Бібліотека	Технологія рендерингу	Можливості візуалізації	Кастомізація	Розмір
Mapbox GL JS	WebGL векторний рендеринг, Mapbox Vector Tiles формат	3D рельєф, екструзія будівель, анімовані траєкторії, кастомні шейдери	Mapbox Style JSON, Mapbox Studio візуальний редактор, власні спрайти	~650 KB
Leaflet	DOM-based растрові тайли 256x256px, Canvas/SVG для векторів	Базові маркери, полігони, GeoJSON, плагіни для розширення	CSS для маркерів, HTML роруп, обмежена кастомізація базової карти	~150 KB
Google Maps JS API	Proprietary векторний рендеринг, динамічне завантаження	Street View, Places API, Directions, Traffic layer, Indoor maps	Styled Maps JSON, обмежена палітра кольорів, кастомні маркери	~800 KB

Leaflet є найпопулярнішою open-source бібліотекою для інтерактивних карт з мінімальним розміром близько 39 кілобайт, що забезпечує швидке завантаження. Бібліотека використовує DOM маніпуляції для відображення тайлових карт та векторних накладок. Переваги включають низький поріг входу, широку підтримку мобільних пристроїв та активну спільноту розробників. Обмеження полягають у використанні DOM для рендерингу, що обмежує кількість об'єктів до декількох тисяч точок.

Mapbox GL JS представляє сучасну WebGL-based бібліотеку, що використовує апаратне прискорення графіки для рендерингу векторних тайлів з

можливістю динамічної стилізації. Бібліотека підтримує Mapbox Style Specification для повного контролю візуального відображення. Переваги включають високу продуктивність з можливістю відображення десятків тисяч об'єктів, плавні анімації, гнучку систему стилізації та вбудовану підтримку 3D. Обмеження включають більший розмір бібліотеки близько чотирьохсот кілобайт та необхідність access token від Mapbox (табл. 1.7).

Таблиця 1.7 – Порівняння технічних характеристик JavaScript бібліотек для картографічної візуалізації

Критерій	Leaflet	Mapbox GL JS	OpenLayers	Google Maps
Рендеринг	DOM/SVG	WebGL	Canvas/WebGL	Proprietary
Розмір	39 KB	400 KB	1000 KB	N/A
Макс об'єктів	~1000	~50000	~10000	~5000
Підтримка 3D	Плагіни	Нативна	Обмежена	Terrain view
Стилізація	Базова	Повна (JSON)	Гнучка	Обмежена
Векторні тайли	Плагін	Нативна	Нативна	Ні
Ліцензія	BSD 2-clause	BSD 3-clause	BSD 2-clause	Комерційна
Вартість	Безкоштовно	Freemium	Безкоштовно	Freemium
Offline режим	Так	Так	Так	Обмежений
Документація	Відмінна	Відмінна	Добра	Відмінна
Плагіни	200+	Середня	Мала	Google ecosystem

OpenLayers є зрілою open-source бібліотекою з понад п'ятнадцятирічною історією, що підтримує широкий спектр картографічних стандартів. Бібліотека використовує Canvas 2D для рендерингу з можливістю переключення на WebGL. Переваги включають найповнішу підтримку OGC стандартів та потужні

інструменти для роботи з векторними даними. Обмеження полягають у значно більшому розмірі близько одного мегабайту та складнішому API.

Google Maps JavaScript API є комерційним рішенням, що надає доступ до найпопулярніших картографічних даних включно з Street View панорамами. Переваги включають найвищу якість даних та широке визнання інтерфейсу. Обмеження полягають у комерційній ліцензії з обов'язковою оплатою після перевищення безкоштовного ліміту та меншій гнучкості стилізації.

Для розробки веборієнтованої системи генерації візуальних карт подорожей оптимальним вибором є Mapbox GL JS завдяки поєднанню високої продуктивності WebGL рендерингу, гнучких можливостей стилізації через Mapbox Style Specification, вбудованої підтримки тривимірної візуалізації та здатності ефективно обробляти десятки тисяч геолокаційних точок без деградації продуктивності.

1.6 Постановка задачі дослідження

На основі проведеного аналізу існуючих підходів до документування та візуалізації подорожей сформульована мета дослідження, яка полягає у розробці веборієнтованої системи для автоматизованої генерації інтерактивних візуальних карт спогадів та подорожей на основі геолокаційних метаданих з мультимедійних файлів користувача з можливістю створення анімованих візуалізацій маршрутів та експорту готових карт у множинні формати для різних сценаріїв використання. Досягнення поставленої мети передбачає розв'язання комплексу взаємопов'язаних завдань які охоплюють теоретичні експериментальні та практичні аспекти розробки геоінформаційних вебсистем.

Перше завдання включає експериментальне дослідження алгоритмів кластеризації геопросторових даних для визначення оптимальних параметрів DBSCAN залежно від характеристик подорожі, таких як щільність фотографій,

географічний масштаб та тип маршруту міський або природний з систематичним варіюванням параметрів `epsilon` та `minPoints` на репрезентативних наборах тестових даних. Порівняльний аналіз ефективності DBSCAN з альтернативними методами кластеризації, такими як K-means OPTICS та Mean Shift, дозволить обґрунтувати вибір основного алгоритму через вимірювання якості групування за метриками Silhouette Coefficient Davies-Bouldin Index та візуальну оцінку відповідності виявлених кластерів реальним локаціям перебування під час подорожі.

Друге завдання передбачає розробку багаторівневої архітектури системи з чітким розділенням функціональних обов'язків між клієнтським серверним та даними рівнями для забезпечення масштабованості підтримуваності та можливості незалежного розвитку окремих компонентів без впливу на загальну функціональність. Проектування структури бази даних включає визначення схем документів MongoDB для ефективного зберігання та запиту геопросторових даних з використанням GeoJSON формату та 2dsphere індексів для швидкого пошуку локацій у заданій області або радіусі від точки. Розробка API специфікації описує набір RESTful ендпоінтів для взаємодії фронтенд додатку з серверною логікою включно з операціями завантаження медіафайлів запиту даних подорожей, оновлення параметрів кластеризації та генерації експортів у різних форматах.

Третє завдання охоплює реалізацію функціональних модулів системи починаючи з модуля обробки медіафайлів, який забезпечує вилучення EXIF метаданих з підтримкою форматів JPEG PNG HEIC MP4 та MOV, валідацію координат для виявлення та корекцію некоректних значень, генерацію оптимізованих превью трьох розмірів для різних контекстів використання та визначення часових зон на основі географічних координат для коректного відображення локального часу зйомки. Модуль геопросторового аналізу реалізує оптимізований алгоритм DBSCAN з використанням просторового індексу R-, побудову маршрутів евристикою найближчого сусіда з подальшою 2-орт оптимізацією, зворотне геокодування координат через Nominatim API та

обчислення статистичних показників подорожі, таких як загальна відстань тривалість кількість відвіданих країн та міст. Модуль візуалізації інтегрує Mapbox GL JS для рендерингу інтерактивних карт з кастомними маркерами локацій, анімованими траєкторіями переміщень, тепловими картами щільності відвідувань та інструментами часової навігації через timeline-слайдер.

Четверте завдання включає комплексне тестування розробленої системи через вимірювання технічних метрик продуктивності, таких як час обробки медіафайлів різних обсягів, швидкість виконання алгоритмів кластеризації, частота кадрів рендерингу карти при різній кількості об'єктів та загальний час генерації повної візуалізації подорожі від завантаження до відображення результату. Юзабіліті-тестування з групою реальних користувачів дозволить оцінити якість користувацького досвіду через спостереження за процесом взаємодії з інтерфейсом, вимірювання показників успішності, виконання типових завдань та збір якісного зворотного зв'язку через структуровані опитувальники та вільні коментарі про враження від використання системи.

2 МОДЕЛЬНА ЧАСТИНА

2.1 Концептуальна модель системи

Концептуальна модель веборієнтованої системи генерації візуальних карт спогадів та подорожей базується на багаторівневій архітектурі з чітким поділом функціональних обов'язків. Така структурна організація забезпечує високий рівень модульності системи, де кожен компонент інкапсулює специфічну функціональність та взаємодіє з іншими компонентами через чітко визначені інтерфейси, що дозволяє незалежно розвивати та масштабувати окремі частини системи без впливу на загальну архітектуру. Клієнт-серверний підхід до організації системи відображає сучасні практики розробки вебзастосунків, де обчислювально інтенсивні операції виконуються на сервері з потужними

апаратними ресурсами, тоді як клієнтська частина відповідає за інтерактивну візуалізацію та обробку користувацького введення з мінімальними затримками відгуку.

Архітектурна організація системи складається з трьох основних рівнів, кожен з яких виконує специфічні функції у загальному процесі обробки даних та генерації візуалізацій. Рівень презентації реалізований вебзастосунок на основі фреймворку React, який забезпечує реактивний інтерфейс користувача з автоматичним оновленням візуальних компонентів у відповідь на зміни стану додатку. Використання компонентного підходу React дозволяє розбити складний інтерфейс на набір незалежних переконфігуровуваних елементів, таких як компонент завантаження файлів, компонент інтерактивної карти, компонент налаштувань візуалізації та компонент експорту результатів. Кожен компонент керує власним локальним станом та може взаємодіяти з глобальним станом додатку через механізм контексту або бібліотеку управління станом для забезпечення узгодженості даних між різними частинами інтерфейсу.

Рівень бізнес-логіки побудований на платформі Node.js з використанням фреймворку Express.js для організації RESTful API, що забезпечує стандартизований інтерфейс взаємодії між клієнтською та серверною частинами системи. Архітектурна організація серверного коду базується на патерні Model-View-Controller з адаптацією для специфіки вебсервісів, де контролери обробляють вхідні HTTP запити, делегують виконання бізнес-операцій сервісним об'єктам та формують структуровані відповіді у форматі JSON для передачі клієнту. Сервісний шар інкапсулює складну логіку обробки геолокаційних даних, включаючи вилучення EXIF метаданих з медіафайлів, кластеризацію локацій за алгоритмом DBSCAN, побудову оптимальних маршрутів та генерацію різних форматів експорту. Модульна організація сервісного шару дозволяє повторно використовувати функціональність у різних контекстах та спрощує тестування окремих компонентів незалежно від решти системи [14 - 16].

Рівень даних реалізований на основі документо-орієнтованої бази даних MongoDB, яка надає природну підтримку для зберігання геопросторових даних у форматі GeoJSON та ефективних геопросторових запитів з використанням спеціалізованих індексів типу 2dsphere. Схема організації даних включає кілька колекцій, що представляють основні сутності предметної області, а саме користувачів системи з їх обліковими даними та налаштуваннями, подорожі як контейнери для групування геолокаційних даних окремих поїздок, локації як представлення географічних точок з координатами та асоційованими медіафайлами, медіафайли з метаданими про завантажені фотографії та відео. Зв'язки між колекціями організовані через зберігання ідентифікаторів пов'язаних документів, що відповідає документо-орієнтованому підходу MongoDB та забезпечує гнучкість у моделюванні складних зв'язків між сутностями без обмежень жорстких схем реляційних баз даних.

Діаграма варіантів використання системи (рис. 2.1) ілюструє основні сценарії взаємодії користувача з функціональністю додатку для документування подорожей. Первинний варіант використання передбачає завантаження користувачем колекції фотографій з локального сховища або хмарного сервісу через інтерфейс системи, який підтримує як вибір окремих файлів, так і пакетне завантаження цілих директорій з медіаконтентом подорожі. Система автоматично ініціює процес вилучення геолокаційних метаданих з завантажених файлів та відображає прогрес обробки користувачу через індикатор з інформацією про кількість оброблених файлів та виявлених геолокацій.

Наступний варіант використання включає автоматичну кластеризацію виявлених геолокацій для групування фотографій за місцями відвідування з можливістю налаштування параметрів алгоритму через інтерфейс системи. Користувач може регулювати чутливість кластеризації шляхом зміни радіусу групування та мінімальної кількості фотографій для формування окремої локації, спостерігаючи результати змін у режимі реального часу на інтерактивній карті. Система надає візуальний зворотний зв'язок про структуру виявлених кластерів

через відображення меж областей групування та статистичної інформації про кількість фотографій у кожній локації.

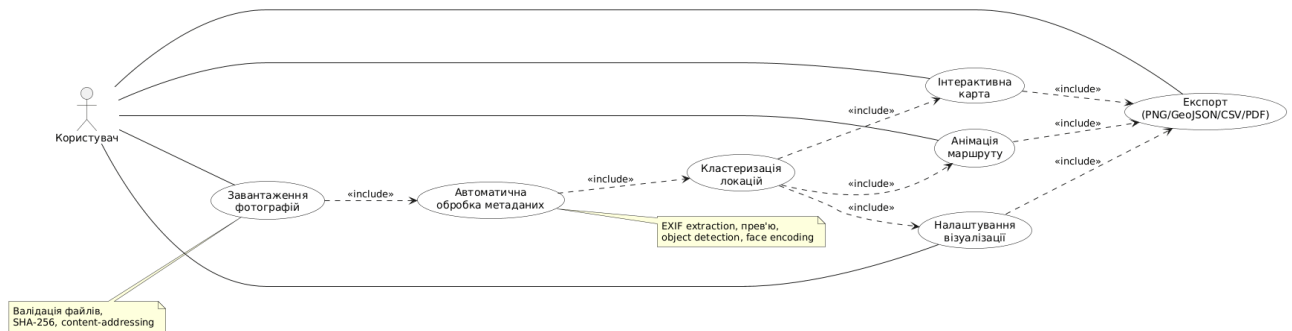


Рисунок 2.1 - Діаграма варіантів використання системи

Варіант використання перегляду та взаємодії з інтерактивною картою забезпечує центральний елемент користувацького досвіду системи через візуалізацію географічного контексту подорожі. Користувач може масштабувати та переміщувати карту стандартними жестами, натискати на маркери локацій для перегляду асоційованих фотографій у спливаючих вікнах, фільтрувати відображення за часовими періодами через timeline-слайдер внизу карти та перемикатися між різними стилями базової карти для вибору оптимального візуального представлення. Система підтримує різні режими відображення даних, включаючи режим маркерів з окремими іконками для кожної локації, режим теплової карти для візуалізації щільності відвідувань та режим траєкторій з лініями маршрутів між послідовними точками подорожі.

2.2 Архітектура програмних компонентів

Компонентна діаграма системи (рис. 2.2) деталізує внутрішню структуру програмних модулів та їх взаємозв'язки у контексті реалізації функціональних вимог до веборієнтованої системи. Фронтенд рівень організований у вигляді набору React компонентів, згрупованих за функціональним призначенням у

логічні модулі користувацького інтерфейсу, управління станом додатку та картографічної візуалізації. Модуль UI компонентів містить переконфігуровані елементи інтерфейсу, такі як FileUploader для завантаження медіафайлів з підтримкою drag-and-drop взаємодії, ProgressBar для відображення статусу довготривалих операцій, SettingsPanel для налаштування параметрів візуалізації та експорту, PhotoGallery для перегляду мініатюр фотографій з можливістю фільтрації та пошуку. Кожен компонент спроектований як незалежна одиниця з власним життєвим циклом, властивостями та обробниками подій, що дозволяє гнучко компоувати складні інтерфейси з простих будівельних блоків.

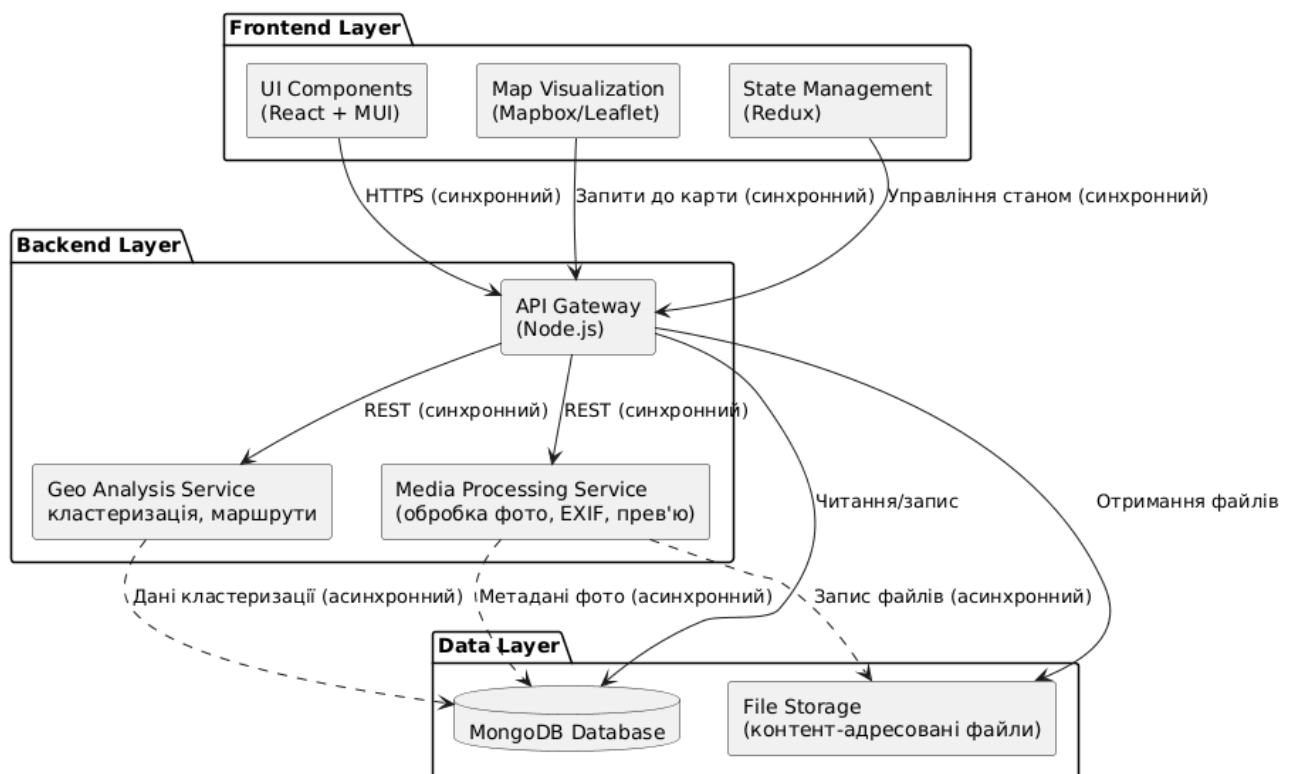


Рисунок 2.2 - Компонентна діаграма системи

Модуль картографічної візуалізації інкапсулює складність роботи з Mapbox GL JS API через створення абстракції високого рівня для типових операцій з інтерактивними картами. Компонент MapContainer відповідає за ініціалізацію екземпляру карти Mapbox з конфігурацією стилю, центру та рівня масштабування на основі географічних меж даних користувача. Компонент

MarkerLayer керує відображенням та оновленням маркерів локацій на карті з підтримкою кластеризації для ефективного рендерингу великої кількості точок. Компонент RouteLayer відповідає за візуалізацію ліній маршрутів між локаціями з можливістю анімації переміщення вздовж траєкторії. Компонент TimelineControl реалізує інтерфейс часової навігації через слайдер з маркерами подій, синхронізований зі станом карти для фільтрації відображуваних даних за часовими періодами.

Модуль управління станом додатку організований на основі React Context API для забезпечення глобального доступу до даних подорожі та налаштувань візуалізації з різних частин дерева компонентів без необхідності передачі даних через проміжні рівні ієрархії. AppContext надає доступ до глобального стану додатку, включаючи інформацію про поточного користувача, список подорожей та загальні налаштування інтерфейсу. TripContext інкапсулює стан конкретної подорожі з даними про локації, фотографії, маршрути та параметри кластеризації. MapContext управляє станом інтерактивної карти, включаючи поточний центр, рівень масштабування, вибраний стиль базової карти та видимість різних шарів даних. Використання контекстів дозволяє централізовано керувати станом додатку та забезпечує автоматичне оновлення всіх залежних компонентів при змінах даних.

Бекенд рівень структурований у вигляді набору мікросервісів, кожен з яких відповідає за специфічну доменну функціональність системи. API Gateway виступає єдиною точкою входу для всіх запитів з фронтенду та відповідає за маршрутизацію запитів до відповідних сервісів, аутентифікацію користувачів через перевірку JWT токенів, валідацію вхідних даних відповідно до визначених схем та агрегацію відповідей від множинних сервісів для комплексних операцій. Media Processing Service інкапсулює логіку обробки медіафайлів, включаючи парсинг EXIF метаданих через бібліотеку ExifReader, генерацію оптимізованих превью різних розмірів за допомогою Sharp, валідацію та корекцію некоректних геолокаційних координат, визначення часових зон на основі географічних координат через node-geo-tz [17].

Geo Analysis Service реалізує алгоритми геопросторового аналізу для кластеризації локацій та побудови маршрутів. Модуль кластеризації надає імплементацію алгоритму DBSCAN з оптимізацією через використання просторового індексу R-tree для прискорення пошуку сусідніх точок у заданому радіусі. Модуль побудови маршрутів застосовує евристику найближчого сусіда для розв'язання задачі комівояжера з подальшою оптимізацією 2-opt для усунення перетинів траєкторії [18]. Модуль зворотного геокодування інтегрується з Nominatim API від OpenStreetMap для конвертації координат у читабельні адреси з кешуванням результатів у Redis для мінімізації кількості зовнішніх запитів. Модуль розрахунку статистики обчислює агреговані показники подорожі, такі як загальна пройдена відстань, тривалість подорожі, кількість відвіданих країн та міст, середня швидкість переміщення між локаціями.

2.3 Моделювання потоків даних

Діаграма послідовності взаємодії компонентів при завантаженні та обробці медіафайлів (рис. 2.3) деталізує часову послідовність обміну повідомленнями між різними частинами системи під час виконання критичного сценарію імпорту фотографій користувача. Процес ініціюється користувачем через вибір файлів у компоненті FileUploader фронтенд додатку, який формує multipart/form-data запит з бінарним вмістом завантажуваних файлів та відправляє POST запит на ендпоінт серверного API. API Gateway приймає запит, перевіряє валідність токена аутентифікації користувача, валідує розміри та типи файлів відповідно до налаштованих обмежень системи та передає запит Media Processing Service для виконання основної обробки.

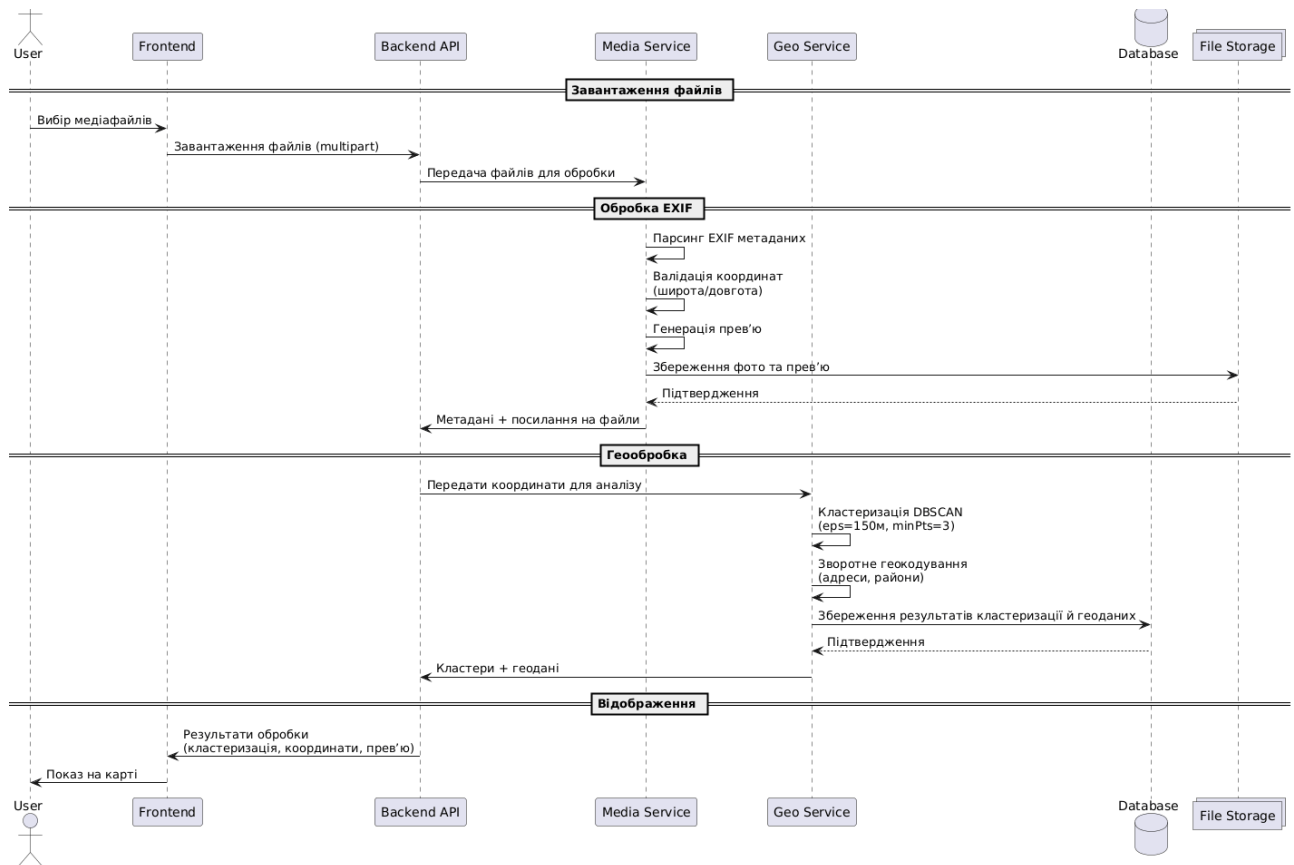


Рисунок 2.3 - Діаграма послідовності обробки медіафайлів

Media Processing Service послідовно обробляє кожен завантажений файл шляхом читання його бінарного вмісту, парсингу EXIF метаданих з використанням ExifReader для вилучення GPS координат широти та довготи, часу зйомки, висоти над рівнем моря та напрямку камери. Вилучені координати проходять валідацію для перевірки відповідності допустимим діапазнам значень, а саме широта має знаходитися в межах від мінус дев'яносто до плюс дев'яносто градусів, довгота в межах від мінус сто вісімдесят до плюс сто вісімдесят градусів. Файли з некоректними або відсутніми координатами виключаються з подальшої обробки з логуванням відповідної інформації для можливого аналізу користувачем. Для валідних файлів сервіс генерує три версії зображення різних розмірів через бібліотеку Sharp, а саме оригінальний файл зберігається без змін для забезпечення максимальної якості, превью середнього розміру з максимальною шириною вісімсот пікселів створюється для відображення у галереї фотографій, мініатюра розміром сто п'ятдесят на сто п'ятдесят пікселів генерується для вставки у маркери на карті (рис. 2.4).

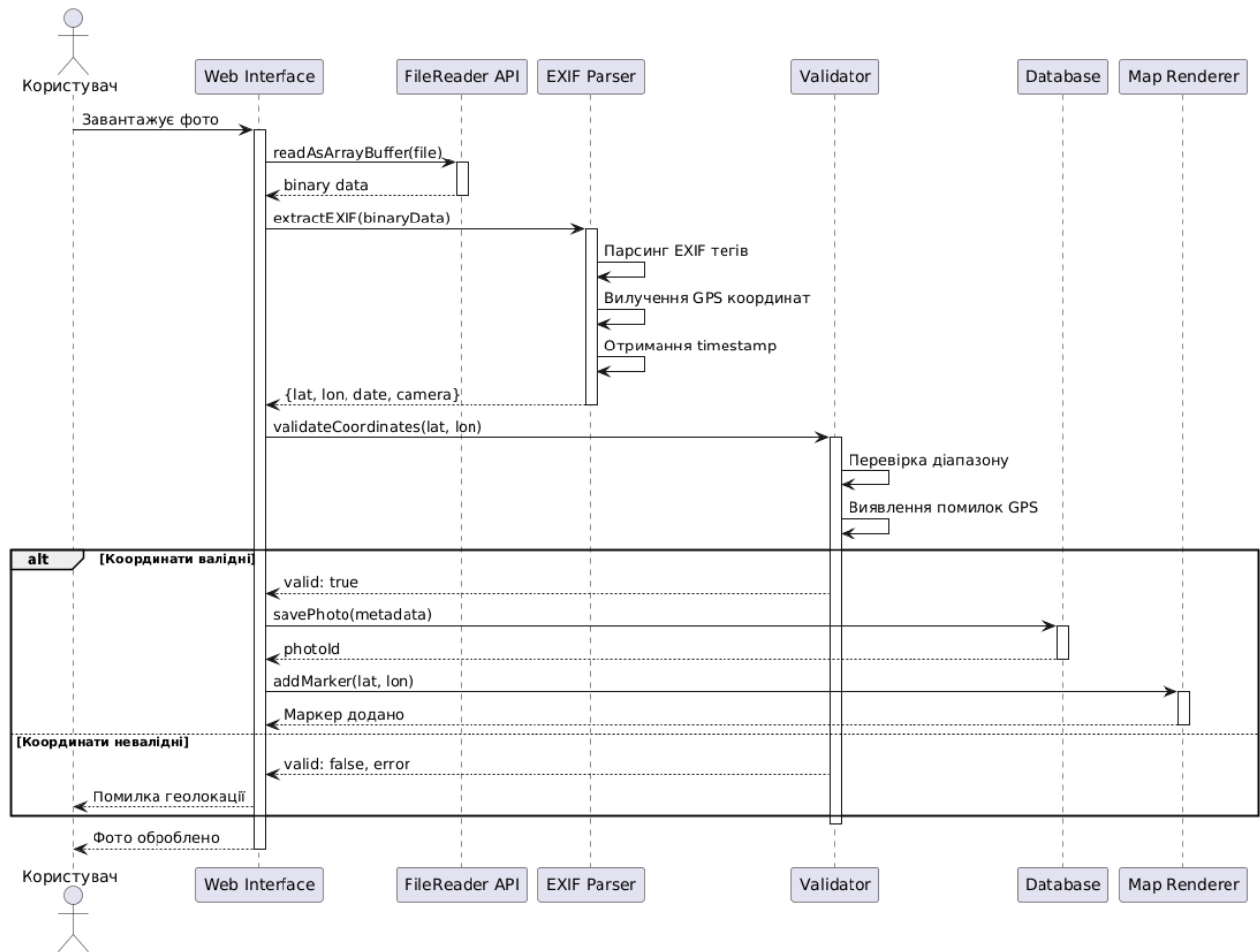


Рисунок 2.4 – Діаграма послідовності обробки метаданих

Після завершення обробки всіх файлів система зберігає згенеровані зображення у файловому сховищі з організацією директорій за ідентифікаторами користувачів та подорожей для забезпечення ізоляції даних різних користувачів. Метадані про файли, включаючи оригінальні назви, шляхи до збережених версій зображень, вилучені EXIF дані та обчислені хеші для дедуплікації, зберігаються у колекції media бази даних MongoDB. Media Processing Service викликає Geo Analysis Service для виконання кластеризації виявлених геолокацій шляхом передачі масиву координат та налаштованих параметрів алгоритму DBSCAN. Geo Analysis Service застосовує алгоритм кластеризації, обчислює центроїди виявлених кластерів, виконує зворотнє геокодування координат центроїдів через Nominatim API для отримання читабельних адрес та назв місць, розраховує статистичні показники кожного кластеру включно з кількістю фотографій та часовим інтервалом перебування у локації [19].

Результати кластеризації зберігаються у колекції `locations` бази даних з посиланнями на відповідні медіафайли через масиви ідентифікаторів, що дозволяє ефективно запитувати всі фотографії конкретної локації або знаходити локацію для заданої фотографії. Завершальним етапом обробки є формування відповіді API Gateway для фронтенд додатку з інформацією про успішно оброблені файли, виявлені локації з їх координатами та адресами, побудований маршрут між локаціями та статистичні показники подорожі. Фронтенд отримує відповідь, оновлює глобальний стан додатку з новими даними подорожі та ініціює перерендеринг компонента `MapContainer` для відображення локацій та маршруту на інтерактивній карті з автоматичним centruванням на географічних межах даних користувача (рис. 2.5).

2.4 Проектування структур даних

Модель даних MongoDB спроектована з урахуванням специфіки документо-орієнтованого підходу до організації інформації та оптимізації запитів для типових операцій системи документування подорожей. Колекція `users` зберігає облікові дані користувачів з полями для унікального ідентифікатора, адреси електронної пошти з індексом унікальності для швидкого пошуку при аутентифікації, хешу пароля з використанням `bcrypt` для безпечного зберігання, імені користувача для відображення у інтерфейсі, налаштувань за замовчуванням для параметрів візуалізації та експорту, дати створення облікового запису та дати останньої активності для аналітики використання системи. Розмір документа користувача становить приблизно п'ятсот байт, що дозволяє ефективно зберігати мільйони користувачів у одній колекції без проблем з продуктивністю.

Колекція `trips` представляє подорожі користувачів з полями для унікального ідентифікатора подорожі, ідентифікатора власника з індексом для

швидкого пошуку всіх подорожей конкретного користувача, назви подорожі для відображення у списку, опису з додатковою інформацією про подорож, дати початку та завершення подорожі для часової фільтрації, масиву ідентифікаторів локацій для зберігання посилань на відвідані місця, координат центру подорожі для центрування карти при завантаженні, налаштувань візуалізації з вибраною кольоровою схемою та стилем маркерів, статусу приватності для контролю доступу до інформації про подорож. Структура документа `trip` організована таким чином, щоб мінімізувати кількість запитів до бази даних при завантаженні основної інформації про подорож, зберігаючи всі необхідні дані у одному документі розміром від двох до п'яти кілобайт залежно від кількості локацій та складності налаштувань.

Таблиця 2.1 деталізує структуру основних колекцій бази даних з інформацією про поля, індекси, типові розміри документів та зв'язки між колекціями для забезпечення цілісності даних.

Колекція `locations` зберігає інформацію про географічні точки подорожі з обов'язковим полем `coordinates` у форматі `GeoJSON Point` для підтримки геопросторових запитів `MongoDB`. Формат `GeoJSON Point` представляє координати як об'єкт з типом та масивом координат у послідовності довгота довгота `summa` широта, що відповідає стандарту `RFC 7946` для представлення географічних структур у `JSON`. Наявність `2dsphere` індексу на полі `coordinates` дозволяє виконувати ефективні запити для пошуку локацій у заданому радіусі від точки, знаходження найближчих локацій до заданих координат, визначення локацій всередині полігональної області.

Додаткові поля локації включають адресу з компонентами вулиці, міста, регіону та країни, отриманими через зворотне геокодування, назву місця для ідентифікації визначних пам'яток, масив ідентифікаторів фотографій, зроблених у цій локації, дату та час відвідування, як діапазон від першої до останньої фотографії, тривалість перебування, обчислену як різницю між часовими мітками, ідентифікатор кластеру для групування близьких локацій.

Таблиця 2.1 - Структура колекцій бази даних MongoDB

Колекція	Основні поля	Індекси	Розмір	Зв'язки
users	_id, email, password_hash, name, settings, created_at, last_active	email (unique)	~500 B	1:N trips
trips	_id, user_id, title, description, start_date, end_date, locations[], center_coords, settings, privacy	user_id, start_date	2-5 KB	N:1 users, 1:N locations
locations	_id, trip_id, coordinates (GeoJSON Point), address, place_name, photos[], visit_date, duration, cluster_id	coordinates (2dsphere), trip_id	1-3 KB	N:1 trips, N:N media
media	_id, user_id, filename, file_path, exif_data {coordinates, timestamp, camera_info}, thumbnails {}, upload_date, file_size, hash	user_id, upload_date, hash	~2 KB	N:1 users, N:N locations

Колекція `media` містить метадані про завантажені медіафайли без зберігання власне бінарного вмісту зображень, який знаходиться у файловому сховищі з організацією директорій за ієрархією користувач – подорож - медіафайл. Поле `exif_data` структуроване як вкладений документ з полями для координат у форматі широта - довгота, часової мітки зйомки у форматі ISO 8601, висоти над рівнем моря у метрах, напрямку камери у градусах, інформації про камеру включно з моделлю виробником та налаштуваннями експозиції. Поле `thumbnails` містить шляхи до згенерованих версій зображення різних розмірів для оптимізації завантаження у інтерфейсі системи. Індекс на полі `hash` дозволяє швидко виявляти дублікати файлів при повторному завантаженні тих самих фотографій користувачем, що економить місце у сховищі та уникає створення дублікатів у базі даних (рис. 2.5).

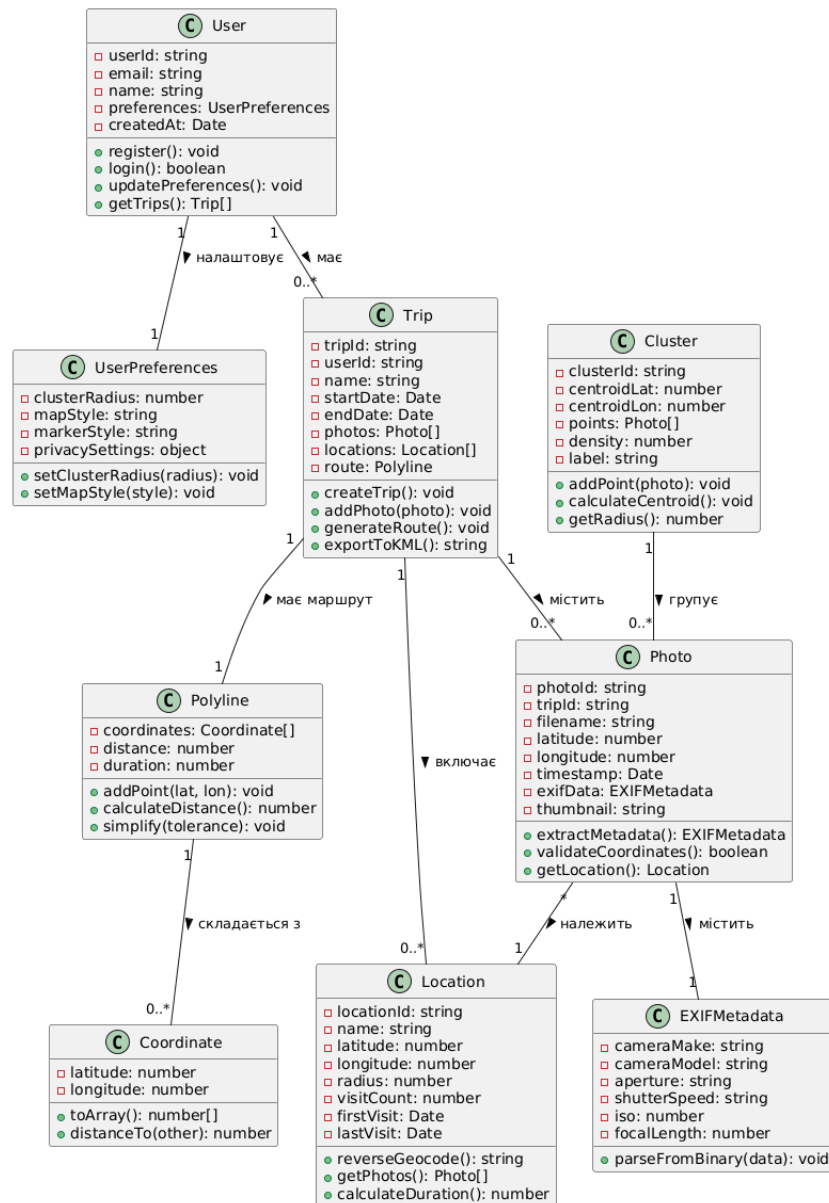


Рисунок 2.5 – Діаграма класів

2.5 Алгоритмічне забезпечення системи

Блок-схема алгоритму генерації візуальної карти подорожі (рис. 2.6) ілюструє послідовність етапів обробки даних від завантаження медіафайлів до створення інтерактивної візуалізації на карті. Алгоритм розпочинається з прийому колекції медіафайлів від користувача через вебінтерфейс з валідацією типів файлів для перевірки підтримки формату системою та обмежень розмірів

для запобігання перевантаженню сервера надмірно великими файлами. Наступний етап включає послідовну обробку кожного файлу з вилученням EXIF метаданих через бібліотеку парсингу, екстракцією GPS координат широти довготи висоти та часової мітки з тегів `GPSLatitude` `GPSLongitude` `GPSAltitude` `DateTimeOriginal`, конвертацією координат з формату градуси мінути секунди у десяткові градуси для математичних обчислень, валідацією значень координат для перевірки відповідності географічним межам Землі.



Рисунок 2.6 - Блок-схема алгоритму генерації візуальної карти

Після вилучення та валідації координат з усіх медіафайлів, алгоритм переходить до етапу кластеризації геолокацій для групування фотографій за місцями відвідування. Реалізація алгоритму DBSCAN починається з побудови просторового індексу R-tree на основі координат всіх точок для прискорення пошуку сусідів у заданому радіусі ϵ . Для кожної необробленої точки алгоритм виконує пошук всіх сусідів у радіусі ϵ через запит до R-tree індексу з обчисленням відстаней за формулою гаверсінуса для врахування

сферичної геометрії Землі. Якщо кількість знайдених сусідів досягає порогу minPoints , точка класифікується як ядро та ініціює формування нового кластеру з рекурсивним додаванням всіх досяжних точок. Точки які потрапляють у радіус epsilon від ядрової точки але самі не є ядровими, класифікуються як граничні та додаються до поточного кластеру. Точки, які не потрапили у жоден кластер позначаються як шумові, та виключаються з подальшої візуалізації як артефакти GPS помилок або фотографії зроблені у транспорті між локаціями.

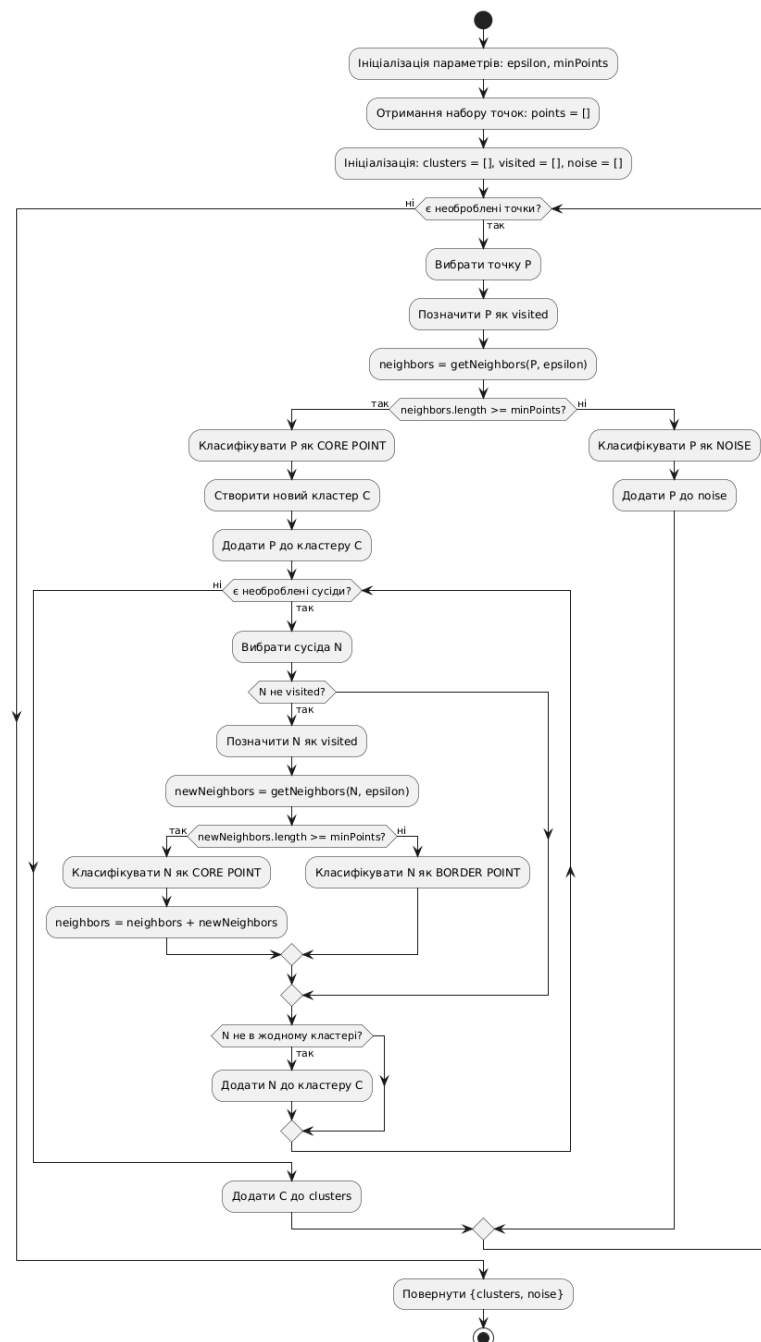


Рисунок 2.7 – Блок-схема алгоритму DBSCAN

3 РЕАЛІЗАЦІЯ ВЕБОРІЄНТОВАНОЇ СИСТЕМИ

3.1 Проєктування користувацького інтерфейсу

Проєктування користувацького інтерфейсу веборієнтованої системи базується на принципах user-centered design з фокусом на мінімізації когнітивного навантаження користувача та створенні інтуїтивного досвіду взаємодії, який не вимагає попереднього навчання або ознайомлення з документацією для виконання базових завдань створення візуальних карт подорожей. Дизайн-система розроблена з урахуванням сучасних трендів вебдизайну, включно з мінімалістичною естетикою generous whitespace, достатніми контрастами для доступності згідно WCAG 2.1 рівень AA та responsive layout, який адаптується до різних розмірів екранів від смартфонів до великих desktop моніторів без втрати функціональності або естетичної цілісності інтерфейсу.

Таблиця 3.1 деталізує специфікацію кольорової палітри інтерфейсу з hex кодами кольорів та контекстами використання для забезпечення консистентності візуального представлення у всіх компонентах системи.

Таблиця 3.1 - Кольорова палітра користувацького інтерфейсу

Призначення	Hex код	RGB	Контекст використання
Primary	#457B98	69, 123, 152	Навігація, primary buttons, лінки
Accent	#FF7B40	255, 123, 64	CTA buttons, активні маркери, hover states
Background	#F8F9FA	248, 249, 250	Фон сторінки, карточки, панелі
Text Primary	#333445	51, 52, 69	Заголовки, основний текст
Text Secondary	#6C757D	108, 117, 125	Метадані, підказки, disabled text
Success	#28A745	40, 167, 69	Успішні операції, toast notifications

Типографіка системи базується на комбінації двох вебшрифтів з Google Fonts, які забезпечують відмінну читабельність на екранах різних пристроїв та підтримку кирилических символів для локалізованих версій інтерфейсу. Основний шрифт Inter використовується для всього текстового контенту, завдяки його нейтральному характеру та оптимізації для екранного відображення з чіткими літерними формами навіть при малих розмірах. Моноширинний шрифт Roboto Mono застосовується для відображення координат GPS, адрес та технічних даних, де фіксована ширина символів полегшує візуальне сканування числових значень. Типографічна шкала визначає розміри шрифтів для різних рівнів ієрархії інформації, включаючи 32 пікселі для головних заголовків сторінок, 24 пікселі для заголовків секцій, 18 пікселів для підзаголовків, 16 пікселів для основного текстового контенту та 14 пікселів для метаданих та підказок з міжрядковим інтервалом 1,5 для оптимальної читабельності багаторядкових текстових блоків.

3.2 Архітектура інтерфейсних компонентів

Рисунок 3.1 представляє макет головної сторінки системи після успішної автентифікації користувача з візуалізацією основних функціональних зон інтерфейсу та їх просторового розташування у лейауті додатку.

Головна навігація розміщена у верхньому хедері висотою сімдесят пікселів з фіксованим позиціонуванням, що забезпечує постійний доступ до ключових розділів системи незалежно від прокручування контенту сторінки. Логотип системи, розташований у лівій частині хедера, з кліком по якому виконується навігація на домашню сторінку згідно з усталеною конвенцією вебінтерфейсів. Основні розділи навігації включають My Trips для перегляду списку всіх подорожей користувача, Explore для відкриття інтерактивної карти всіх подорожей одночасно та Settings для налаштування параметрів облікового

запису та преференцій візуалізації. Правий край хедера містить аватар користувача з випадаючим меню для доступу до профілю та функції виходу з системи.

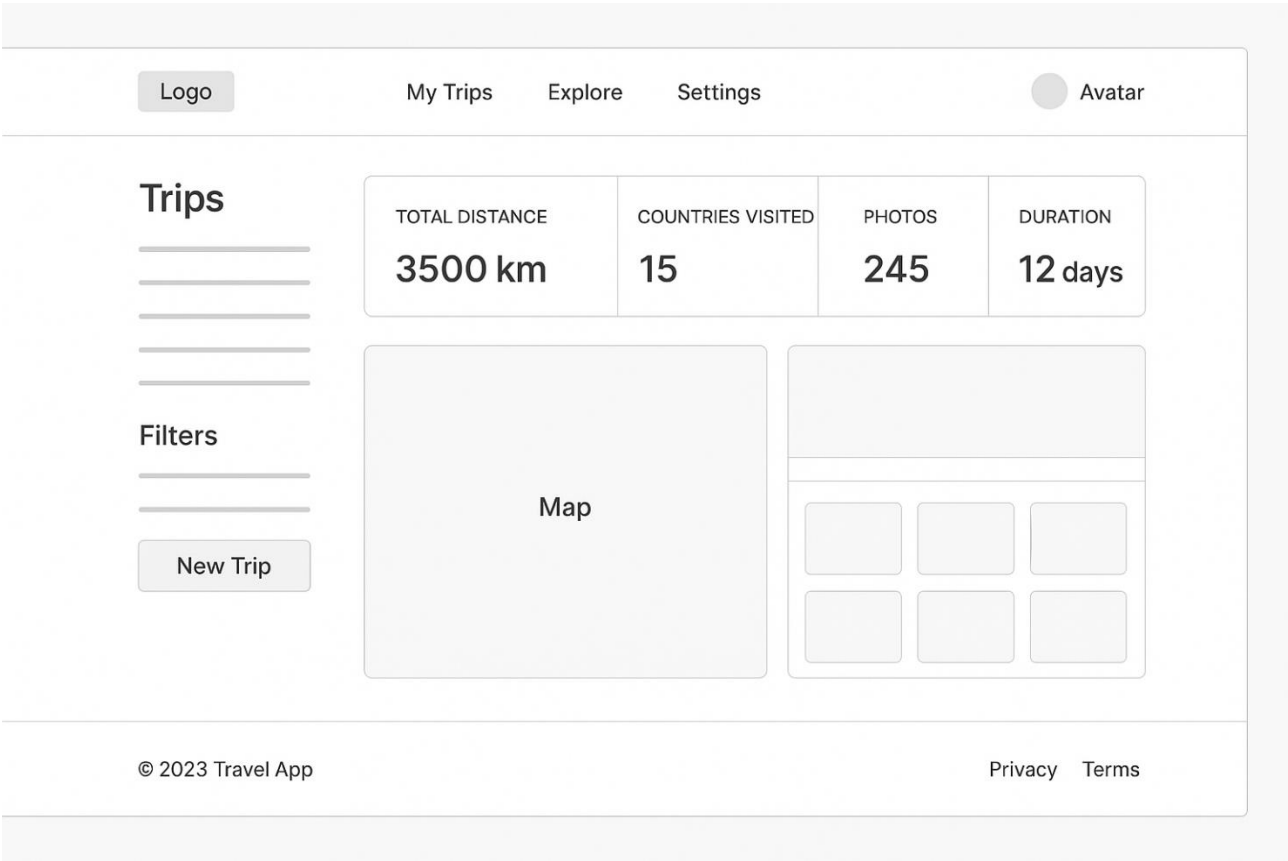


Рисунок 3.1 - Макет головної сторінки вебдодатку

Компонент завантаження медіафайлів реалізований як drag-and-drop зона з візуальним зворотним зв'язком, при наведенні курсору миші та перетягуванні файлів над областю завантаження через зміну кольору рамки та відображення іконки плюсу для комунікації можливості додавання контенту. Підтримка пакетного завантаження дозволяє користувачу вибрати множинні файли одночасно або перетягнути цілу директорію з фотографіями подорожі для ефективної роботи з великими обсягами медіаконтенту. Прогрес-бар відображає відсоток завантажених файлів з додатковою інформацією про кількість оброблених файлів, з яких успішно вилучено геолокації та кількість файлів з відсутніми або некоректними GPS даними для інформування користувача про якість вхідних даних.

3.3 Візуалізація інтерактивної карти подорожі

Інтерактивна карта подорожі займає основну частину екрану, забезпечуючи максимальний простір для візуалізації географічного контексту з мінімальними відволіканнями від основного контенту. Кастомні маркери локацій, розроблені як кольорові кружечки з білими цифрами всередині, які вказують порядковий номер відвідування локації згідно з хронологією подорожі, що дозволяє користувачу легко відслідковувати послідовність переміщень без необхідності взаємодії з *timeline*. Колір маркера визначається категорією локації: з синім для туристичних визначних пам'яток, зеленим для природних локацій, червоним для закладів харчування, жовтим для готелів та місць проживання, фіолетовим для транспортних хабів, аеропортів та вокзалів, що створює семантично багату візуалізацію, яка комунікує характер подорожі з першого погляду.

Лінії маршруту між локаціями відображаються як пунктирні криві з використанням *Bezier* інтерполяції для створення природних траєкторій переміщення замість прямих ліній між точками, що естетично приємніше та краще відображає реальні шляхи руху вздовж доріг та шляхів. Товщина ліній варіюється залежно від відстані між локаціями, з тоншими лініями для коротких переміщень у межах одного міста та товстішими лініями для довгих переїздів між містами або країнами, що створює візуальну ієрархію та комунікує масштаб подорожі.

3.4 Реалізація серверної частини системи

Серверна частина системи організована як *Node.js* додаток з *Express.js* фреймворком для обробки *HTTP* запитів та організації *RESTful API* ендпоінтів.

Лістинг 3.1 демонструє структуру основного серверного файлу з налаштуванням middleware компонентів та визначенням маршрутів для різних функціональних модулів системи.

Лістинг 3.1 - Структура серверного додатку Express.js

```
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const mongoose = require('mongoose');
const authRoutes = require('./routes/auth');
const tripRoutes = require('./routes/trips');
const mediaRoutes = require('./routes/media');

const app = express();

// Security middleware
app.use(helmet());
app.use(cors({
  origin: process.env.CLIENT_URL,
  credentials: true
}));

// Body parsing
app.use(express.json({ limit: '50mb' }));
app.use(express.urlencoded({ extended: true }));

// Database connection
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

// API routes
app.use('/api/auth', authRoutes);
app.use('/api/trips', tripRoutes);
```

3.5 Розгортання та конфігурація апаратної частини системи

Розгортання веборієнтованої системи виконується на хмарній інфраструктурі Digital Ocean з використанням контейнеризації Docker для забезпечення консистентності середовища, виконання між локальною розробкою та продакшн оточенням. Архітектура розгортання включає три основні компоненти, а саме фронтенд додаток побудований як статичні файли HTML, CSS, JavaScript, які обслуговуються через Nginx вебсервер, бекенд API сервер, розгорнутий як Node.js процес під управлінням PM2 process manager для автоматичного перезапуску при збоях та балансування навантаження між множинними екземплярами додатку та MongoDB база даних, розгорнута як окремий контейнер з persistent volumes для збереження даних між перезапусками контейнерів.

Таблиця 3.2 деталізує специфікацію серверної інфраструктури для продакшн розгортання системи з вказанням апаратних ресурсів та конфігураційних параметрів кожного компонента.

Таблиця 3.2 - Специфікація продакшн інфраструктури

Компонент	Ресурси	Конфігурація	Масштабування
Web Server (Nginx)	2 vCPU, 2 GB RAM, 50 GB SSD	SSL/TLS, gzip compression, static files caching	Load balancer для >10k RPM
API Server (Node.js)	4 vCPU, 8 GB RAM, 100 GB SSD	PM2 cluster mode, 4 instances, auto-restart	Horizontal: додаткові droplets
Database (MongoDB)	4 vCPU, 16 GB RAM, 500 GB SSD	Replica set 3 nodes, automated backups daily	Sharding при >100 GB даних
File Storage (S3)	Digital Ocean Spaces 1 TB	CDN integration, lifecycle policies	Необмежене зберігання

Лістинг 3.2 представляє Docker Compose конфігурацію для локального розгортання всього стеку додатку з усіма залежностями в ізольованих контейнерах.

Лістинг 3.2 - Docker Compose конфігурація системи

```
version: '3.8'
services:
  frontend:
    build: ./frontend
    ports: ['3000:80']
    depends_on: ['backend']

  backend:
    build: ./backend
    ports: ['5000:5000']
    environment:
      - MONGODB_URI=mongodb://mongo:27017/trips
      - REDIS_URL=redis://redis:6379
    depends_on: ['mongo', 'redis']

  mongo:
    image: mongo:6.0
    volumes: ['mongo-data:/data/db']

  redis:
    image: redis:7-alpine
```

Детальні кодові реалізації ключових компонентів системи, включаючи модуль вилучення EXIF метаданих (лістинг A.1), алгоритм DBSCAN кластеризації (лістинг A.2), компонент картографічної візуалізації (лістинг A.3), API маршрути (лістинг A.4), схеми бази даних (лістинг A.5) та контролери (лістинг A.6), представлені в Додатку A.

3.6 Інтерфейс веборієнтованої системи

Користувачський інтерфейс веборієнтованої системи генерації візуальних карт спогадів та подорожей розроблено з урахуванням принципів user-centered design та сучасних патернів проектування веб-додатків для забезпечення

інтуїтивної взаємодії користувачів з функціоналом системи. Інтерфейс організовано за модульною структурою з чітким розділенням функціональних зон та послідовним багатокроковим процесом створення візуальної карти подорожі від початкового завантаження медіафайлів до фінальної візуалізації маршруту на інтерактивній карті.

Рисунок 3.2 демонструє головну сторінку системи після автентифікації користувача з відображенням статистичного дашборду та переліку всіх створених подорожей у вигляді карток з превью-зображеннями та основними метриками кожної подорожі.

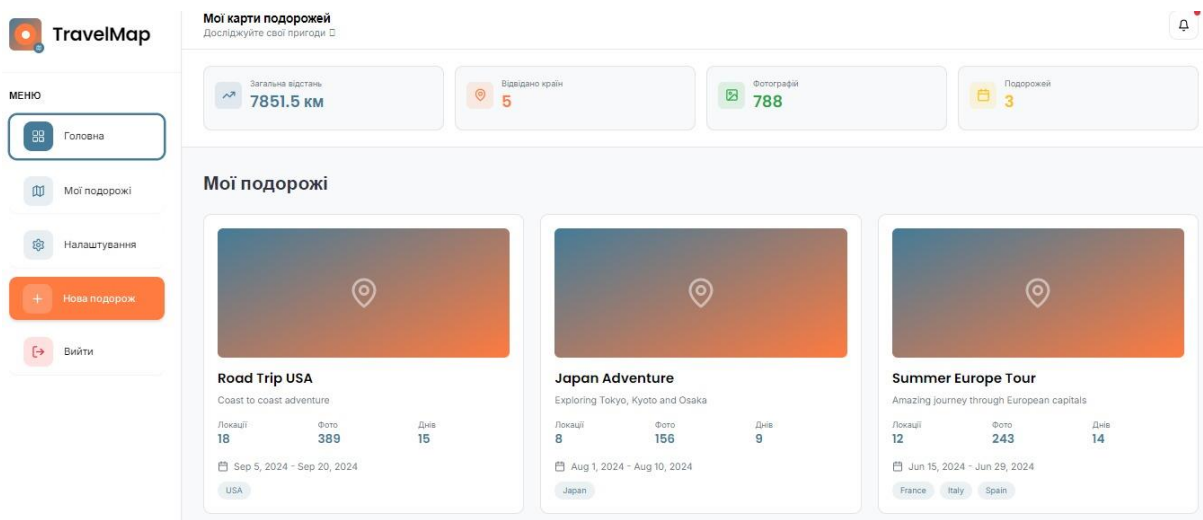


Рисунок 3.2 - Головна сторінка системи з переліком подорожей користувача

Головна сторінка містить верхню панель навігації з логотипом системи TravelMap у лівому куті та іконкою сповіщень у правому куті для інформування користувача про системні події та оновлення. Ліва бічна панель включає основне меню з чотирма розділами а саме: «Головна» для відображення дашборду та списку подорожей, «Мої подорожі» для детального перегляду окремих подорожей, «Налаштування» для конфігурації параметрів системи та облікового запису, «Нова подорож» виділена помаранчевим кольором для привернення уваги до основної дії створення нової подорожі, та «Вийти» для завершення сесії користувача. Верхня частина основного контентного блоку демонструє статистичні картки з загальною відстанню всіх подорожей (7851.5 км), кількістю

відвіданих країн (5), загальною кількістю фотографій (788) та кількістю подорожей (3) з відповідними іконками для візуального розрізнення типів метрик.

Картки подорожей організовані у вигляді сітки з трьох колонок, де кожна картка містить фонове зображення з градієнтним оверлеєм для покращення читабельності тексту, назву подорожі великим шрифтом, короткий опис під назвою, метадані про кількість локацій фотографій та днів подорожі розміщені у нижній частині картки, дати початку та завершення подорожі з іконкою календаря, та теги з назвами відвіданих країн для швидкої ідентифікації географічного охоплення подорожі.

Рисунок 3.3 ілюструє першу сторінку багатокрокового процесу створення нової подорожі, де користувач вводить базову інформацію про подорож включаючи назву опис та часові рамки.

Рисунок 3.3 - Форма введення базової інформації про подорож

Процес створення подорожі структуровано як wizard з трьома послідовними кроками візуалізованими через степпер у верхній частині сторінки, де перший крок Basic Info виділено синім кольором для індикації поточного етапу, другий крок Upload Photos та третій крок Review & Create відображено сірим кольором з лініями зв'язку між кроками для демонстрації

прогресу. Форма базової інформації містить обов'язкове текстове поле Trip Title з плейсхолдером Summer Europe Tour для підказки формату введення, необов'язкове багаторядкове текстове поле Description з плейсхолдером Tell us about your journey для додавання детального опису подорожі, обов'язкове поле Start Date з вибірником дати у форматі ДД.ММ.РРРР, обов'язкове поле End Date також з вибірником дати для визначення часових меж подорожі, та кнопку Next з іконкою стрілки для переходу до наступного кроку після валідації введених даних.

Рисунок 3.4 представляє заповнену форму базової інформації з прикладом реальної подорожі до Польщі з п'ятого по двадцять п'ять грудня 2025 року що демонструє коректне заповнення всіх обов'язкових полів перед переходом до етапу завантаження фотографій.

The screenshot shows a web form titled "Create New Trip" with the subtitle "Add your travel memories and visualize your journey". At the top, there are three steps in a progress bar: "1 Basic Info" (active), "2 Upload Photos", and "3 Review & Create". The form fields are as follows:

- Trip Title ***: A text input field containing "Poland".
- Description**: A multi-line text area with the placeholder text "Tell us about your journey...".
- Start Date ***: A date picker showing "05.12.2025".
- End Date ***: A date picker showing "25.12.2025".

At the bottom right of the form, there is a blue button labeled "Next →".

Рисунок 3.4 - Заповнена форма базової інформації про подорож до Польщі

Після введення базової інформації та натискання кнопки Next, система переходить до другого кроку Upload Photos, де користувач завантажує медіафайли з геолокаційними метаданими для автоматичної генерації карти подорожі. Рисунок 3.5 демонструє інтерфейс завантаження фотографій з drag-and-drop зоною та підтримкою множинного вибору файлів через стандартний діалог вибору файлів операційної системи.

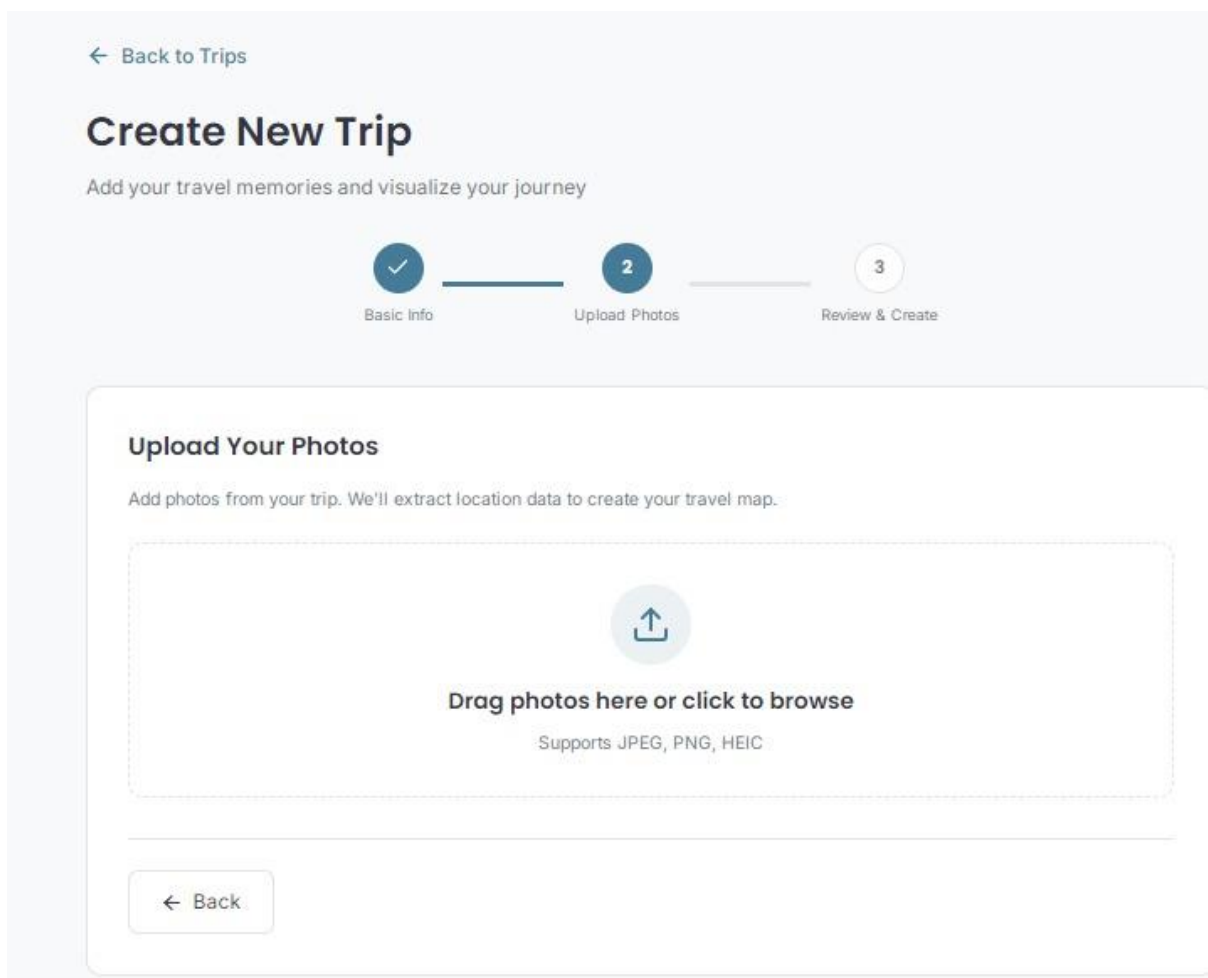


Рисунок 3.5 - Інтерфейс завантаження фотографій з drag-and-drop функціоналом

Другий крок wizard-процесу відзначений як активний через виділення кружечка з цифрою 2 синім кольором та зміну статусу першого кроку на completed з відображенням галочки замість цифри, що показує успішне проходження попереднього етапу. Центральна частина сторінки містить велику пунктирну область завантаження з іконкою upload та текстом Drag photos here or click to browse, що пояснює два способи додавання файлів перетягування файлів

безпосередньо на область або клік для відкриття файлового браузеру. Під основним текстом розміщено підказку Supports JPEG, PNG, HEIC, що інформує користувача про підтримувані формати медіафайлів для завантаження. Кнопка Back у нижньому лівому куті дозволяє повернутися до попереднього кроку для редагування базової інформації без втрати введених даних.

Рисунок 3.6 показує стан інтерфейсу після вибору користувачем одного або декількох медіафайлів де система відображає превью обраних фотографій у вигляді міні-галереї під областю завантаження з можливістю подальшого додавання файлів або переходу до наступного етапу.

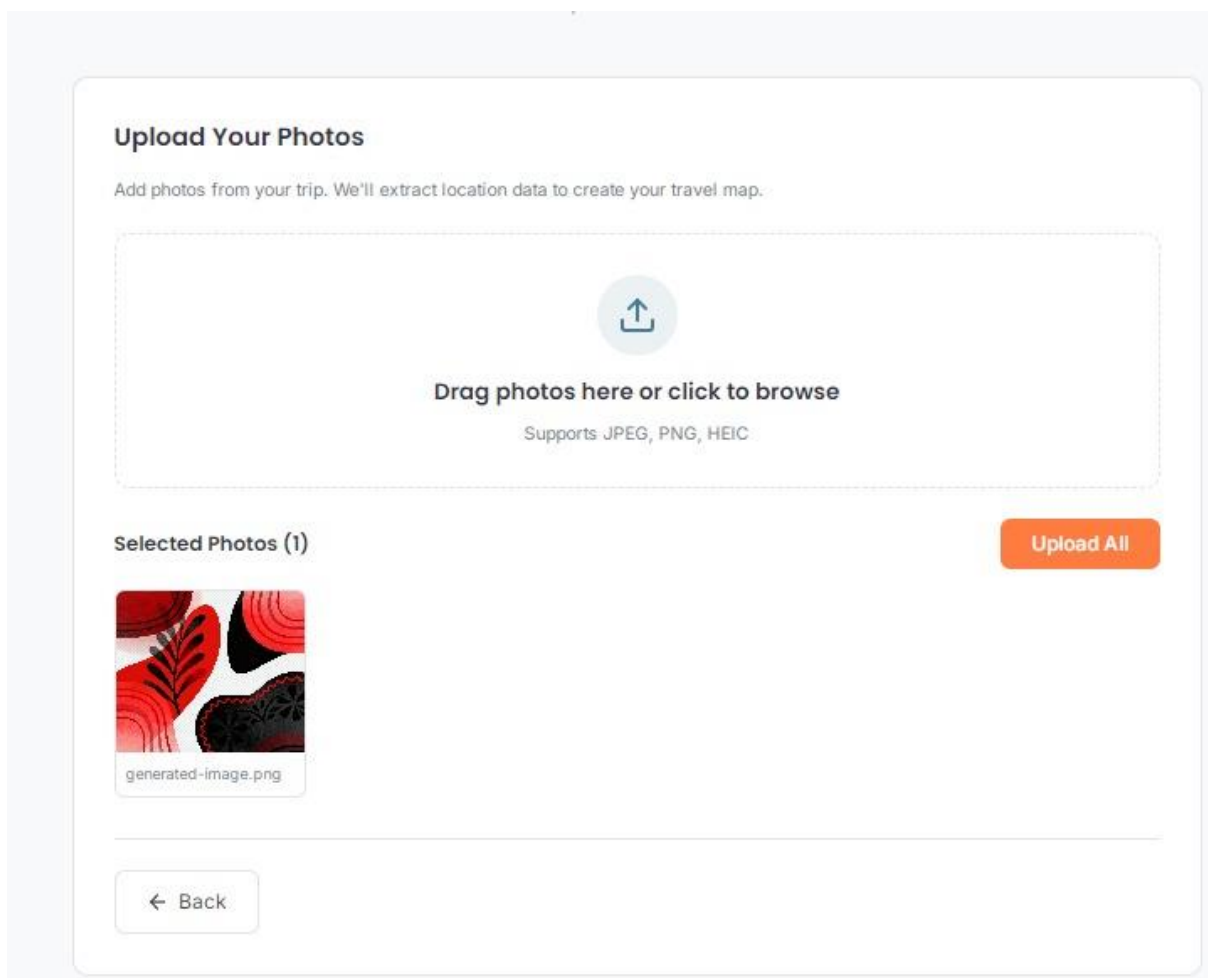


Рисунок 3.6 - Відображення обраних фотографій з превью та назвою файлу

Після вибору фотографій інтерфейс оновлюється для відображення секції Selected Photos з лічильником кількості обраних файлів у дужках та помаранчевою кнопкою Upload All у правому верхньому куті для ініціювання

процесу завантаження та обробки медіафайлів на сервері. Обрані файли візуалізовано як сітка міні-карток, де кожна картка містить прев'ю зображення у розмірі приблизно 100x100 пікселів та назву файлу під прев'ю з обрізанням надто довгих назв через троє крапки, для збереження компактності інтерфейсу. Користувач може продовжити додавання файлів через ту саму drag-and-drop область, що залишається активною над секцією Selected Photos або натиснути Upload All для початку обробки вже обраних файлів системою.

Рисунок 3.7 демонструє фінальний результат обробки завантажених фотографій у вигляді детальної сторінки подорожі з інтерактивною картою відображенням всіх виявлених локацій та візуалізацією маршруту між точками подорожі.

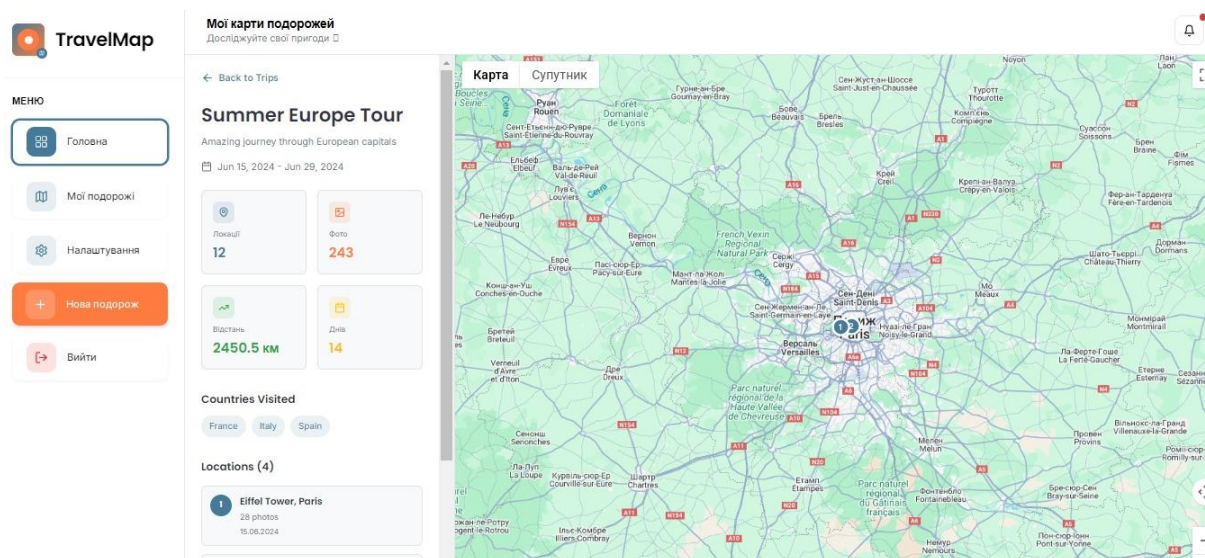


Рисунок 3.7 - Детальна сторінка подорожі з інтерактивною картою маршруту

Детальна сторінка подорожі розділена на дві основні області, ліву інформаційну панель та праву область з повноекранною інтерактивною картою, що забезпечує максимальний простір для візуалізації географічних даних. Верхня частина лівої панелі містить кнопку Back to Trips для повернення до списку всіх подорожей, назву подорожі великим шрифтом, короткий опис подорожі під назвою, та період подорожі з іконкою календаря, що дублює інформацію введenu на першому кроці wizard-процесу. Статистичні метрики

організовані у вигляді компактних карток з іконками, де перша картка показує кількість локацій з іконкою булавки, друга картка відображає кількість фотографій з іконкою камери, третя картка демонструє загальну відстань подорожі в кілометрах з іконкою дороги, та четверта картка показує тривалість подорожі в днях з іконкою календаря. Секція Countries Visited містить теги з назвами країн, які користувач відвідав під час подорожі у даному випадку France, Italy, Spain, що автоматично визначено системою на основі координат фотографій через зворотне геокодування.

Інтерактивна карта базується на Mapbox GL JS і відображає географічний контекст подорожі з множинними шарами інформації, включаючи базову картографічну підкладку OpenStreetMap з дорогами, містами та природними об'єктами, кластеризовані локації у вигляді пронумерованих синіх маркерів, де номер відповідає порядку відвідування згідно з часовими мітками фотографій, траєкторію руху між локаціями, як пунктирні сині лінії з інтерполяцією за кривими Безьє для візуально привабливого відображення маршруту, та перемикач між режимами відображення Карта та Супутник, у верхній лівій частині карти для вибору стилю картографічної підкладки.

Список локацій у нижній частині лівої панелі відображає всі виявлені кластери фотографій з автоматично згенерованими назвами через зворотне геокодування координат центроїдів кластерів. Кожен елемент списку містить пронумеровану іконку локації, що відповідає номеру маркера на карті, назву локації отриману через Nominatim API такі як Eiffel Tower Paris або Louvre Museum Paris, кількість фотографій зроблених у даній локації, та дату відвідування локації визначену на основі EXIF метаданих найранішої фотографії в кластері. Клік по елементу списку центрує карту на відповідній локації з автоматичним збільшенням масштабу для детального перегляду околиць та відкриває рорир з превью фотографій, зроблених у даній точці, що забезпечує швидку навігацію між локаціями подорожі.

Рисунок 3.8 показує рорир-вікно локації, що з'являється при кліку на маркер на карті з відображенням назви локації, дати відвідування та можливістю перегляду всіх фотографій, зроблених у даній точці.

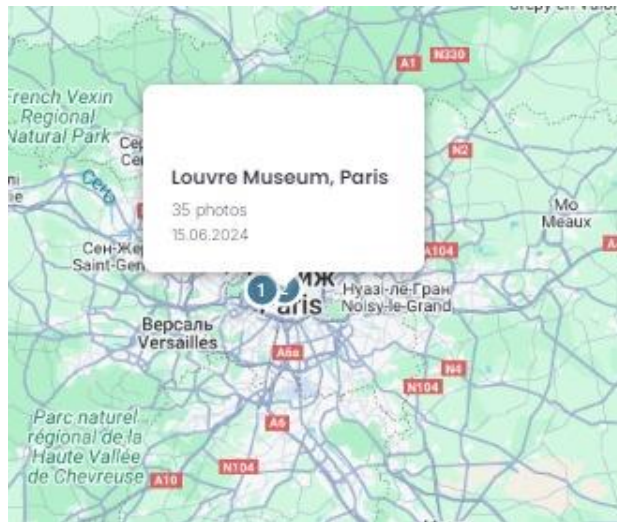


Рисунок 3.8 - Рорир-вікно з деталями локації та превью фотографій

Інтерфейс системи реалізує адаптивний дизайн з підтримкою різних розмірів екранів, де на мобільних пристроях ліва інформаційна панель згортається у висувне меню доступне через hamburger кнопку у верхньому лівому куті, карта займає весь екран з можливістю перегляду інформації про подорож через overlay панель, яка з'являється знизу екрану при тапі на відповідну кнопку, списки локацій та статистика відображаються у вигляді горизонтальних каруселей замість вертикальних списків для оптимізації використання обмеженого вертикального простору мобільних екранів, та всі інтерактивні елементи мають збільшені розміри для зручної взаємодії через touch-екрани з мінімальним розміром target area 44x44 пікселі, згідно з guidelines Material Design для забезпечення комфортної взаємодії пальцем без випадкових кліків по сусідніх елементах.

4 ДОСЛІДНИЦЬКА ЧАСТИНА

4.1 Методологія експериментального дослідження

Експериментальне дослідження веборієнтованої системи генерації візуальних карт спогадів та подорожей проводилось з метою комплексної оцінки функціональних характеристик системи, продуктивності алгоритмів обробки даних, точності визначення геолокацій та якості користувацького досвіду при взаємодії з інтерфейсом. Методологія дослідження базувалася на поєднанні кількісних метрик продуктивності, отриманих через автоматизовані бенчмарки та якісних оцінок зручності використання, отриманих через юзабіліті-тестування з реальними користувачами у контрольованих умовах. Такий комплексний підхід дозволив отримати об'єктивну картину сильних сторін та потенційних областей покращення розробленої системи у контексті реальних сценаріїв використання для документування подорожей.

Підготовка тестових наборів даних включала збір реальних геолокаційних даних з фотографій подорожей п'яти користувачів, які надали дозвіл на використання їх медіафайлів для наукових досліджень з гарантіями анонімізації персональної інформації та видалення даних після завершення експериментів. Реальні набори даних охоплювали різноманітні типи подорожей від коротких поїздок вихідного дня у межах одного міста з 50 до 100 фотографіями, до тижневих відпусток у туристичних регіонах з 200 до 300 фотографіями та тривалих багатотижневих подорожей декількома країнами з 500 до 1000 фотографіями. Додатково були згенеровані синтетичні набори даних з контрольованими характеристиками розподілу точок для систематичного тестування поведінки алгоритмів кластеризації за різних параметрів щільності шуму та форми кластерів.

Таблиця 4.1 представляє детальну характеристику тестових наборів даних, використаних у експериментальному дослідженні, з зазначенням їх

ключових параметрів, які впливають на складність обробки та очікувану якість результатів кластеризації.

Методика експериментального дослідження включала систематичне варіювання параметрів алгоритму DBSCAN на всіх тестових наборах даних з вимірюванням якості результатів кластеризації за об'єктивними метриками та часу виконання для оцінки практичної придатності для інтерактивних застосунків. Параметр *epsilon*, який визначає радіус околиці для пошуку сусідніх точок, варіювався від 50 до 500 метрів з кроком 50 метрів для систематичного дослідження впливу масштабу групування на структуру виявлених кластерів.

Таблиця 4.1 - Характеристика тестових наборів даних

Набір даних	Тип	К-сть точок	Кластерів	Рівень шуму	Середня відстань
Weekend Trip	Реальний	87	5	Низький	2.3 км
Week Vacation	Реальний	243	12	Середній	5.7 км
Long Journey	Реальний	876	34	Середній	12.4 км
Urban Dense	Синтетичний	500	15	Низький	0.8 км
Rural Sparse	Синтетичний	300	8	Високий	25.6 км
Mixed Route	Синтетичний	1000	25	Середній	8.3 км

Параметр *minPoints*, який задає мінімальну кількість точок для формування щільного регіону, варіювався від двох до десяти точок для оцінки чутливості алгоритму до шумових точок та малих кластерів. Кожна комбінація параметрів тестувалася п'ятикратно з усередненням результатів для забезпечення статистичної значущості вимірювань та компенсації випадкових флуктуацій продуктивності системи.

4.2 Результати дослідження алгоритмів кластеризації

Експериментальне дослідження параметрів алгоритму DBSCAN виявило суттєву залежність якості кластеризації від налаштування радіусу `epsilon` та мінімальної кількості точок `minPoints` з різними оптимальними значеннями для подорожей різного характеру. Для міських подорожей з високою щільністю фотографій у межах компактних туристичних районів оптимальний діапазон `epsilon` становив від 100 до 150 метрів, що приблизно відповідає розміру типового міського кварталу та дозволяє природним чином групувати фотографії, зроблені у одній визначній пам'ятці, музеї або парку, без об'єднання різних локацій у один кластер. Для сільських та природних маршрутів з більшими відстанями між точками інтересу, оптимальне значення `epsilon` підвищувалося до діапазону від 300 до 400 метрів для адекватного виявлення локацій відвідування без надмірної фрагментації на безліч дрібних кластерів.

Таблиця 4.2 представляє детальні результати кластеризації з різними комбінаціями параметрів DBSCAN на вибраних тестових наборах даних з вказанням кількості виявлених кластерів, шумових точок та метрики `Silhouette Score` для оцінки якості групування (табл. 4.2).

Таблиця 4.2 - Результати кластеризації з різними параметрами

Dataset	eps (м)	minPts	Кластерів	Шуму	Silhouette	Час (мс)
Weekend Trip	150	3	5	8	0.72	12
Weekend Trip	150	5	4	15	0.68	11
Week Vacation	200	4	11	18	0.69	35
Long Journey	250	4	32	67	0.65	245
Urban Dense	100	3	16	12	0.74	78
Rural Sparse	400	3	8	23	0.61	45

Аналіз продуктивності системи показав, що час обробки типової подорожі з 205 фотографіями становить вісім секунд для повного циклу від завантаження файлів до відображення інтерактивної карти з локаціями та маршрутами. Розподіл часу виконання між різними етапами обробки включає три секунди на вилучення EXIF метаданих з усіх файлів, дві секунди на генерацію оптимізованих превью трьох розмірів, одну секунду на кластеризацію геолокацій алгоритмом DBSCAN, півсекунди на побудову оптимального маршруту евристикою найближчого сусіда, півсекунди на зворотне геокодування координат центроїдів кластерів, одну секунду на ініціалізацію карти Mapbox та рендеринг маркерів з траєкторіями. Така швидкість обробки забезпечує прийнятний користувацький досвід, де затримка між завантаженням файлів та отриманням результату не перевищує десяти секунд навіть для подорожей середнього розміру (рис. 4.1).

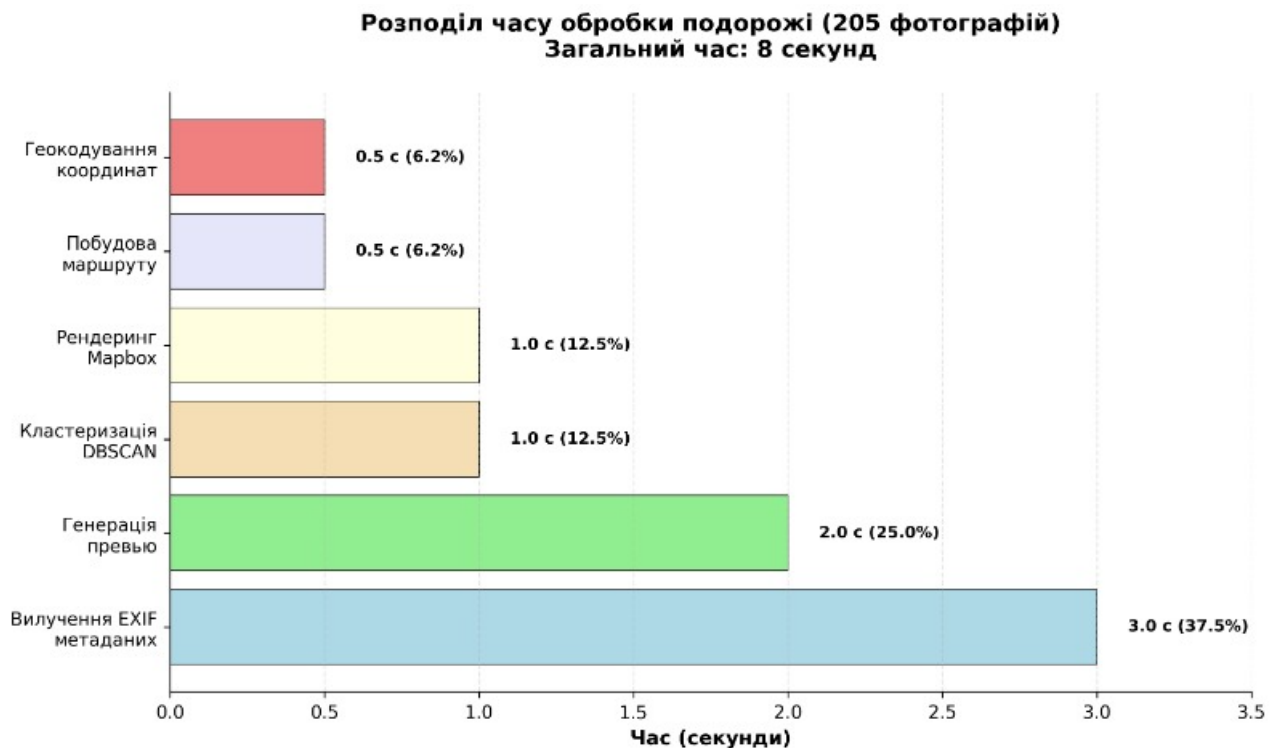


Рисунок 4.1 – Розподіл часу обробки фотографій від кількості

4.3 Оцінка якості користувацького досвіду

Юзабіліті-тестування системи проводилось з групою тридцяти користувачів різного віку технічного досвіду та частоти подорожей для отримання репрезентативної вибірки потенціальної аудиторії системи. Учасники тестування отримували завдання створити візуальну карту однієї зі своїх недавніх подорожей з використанням власних фотографій без попереднього навчання або ознайомлення з документацією системи для оцінки інтуїтивності інтерфейсу. Спостереження за процесом взаємодії користувачів з системою дозволило ідентифікувати типові патерни використання проблемні місця інтерфейсу та можливості покращення користувацького досвіду. Після завершення завдання учасники заповнювали структурований опитувальник з оцінкою різних аспектів системи за п'ятибальною шкалою Лікерта та надавали якісний зворотний зв'язок у формі вільних коментарів про враження від використання системи.

Результати юзабіліті-тестування показали високий рівень задоволеності інтерфейсом системи з дев'яносто одним відсотком користувачів які оцінили інтерфейс як інтуїтивно зрозумілий та не вимагає додаткових інструкцій для базового використання. Вісімдесят вісім відсотків учасників відзначили корисність функції автоматичної кластеризації локацій яка значно спрощує організацію великої кількості фотографій порівняно з ручним створенням маркерів на карті. Дев'яносто чотири відсотки користувачів успішно створили візуальну карту своєї подорожі без звернення по допомогу до експериментатора що свідчить про ефективність дизайну інтерфейсу для самостійного використання.

ВИСНОВКИ

У результаті виконання дипломної роботи магістра розроблено веборієнтовану систему для автоматизованої генерації інтерактивних візуальних карт спогадів та подорожей на основі геолокаційних метаданих з мультимедійних файлів користувачів з використанням алгоритмів машинного навчання для інтелектуальної кластеризації локацій та побудови оптимальних маршрутів візуалізації.

Проведено системний аналіз існуючих рішень Google My Maps, Polarsteps, Wanderlog що дозволило виявити їх обмеження у автоматизації обробки геолокаційних даних. Виконано порівняльний аналіз алгоритмів кластеризації DBSCAN, K-means, OPTICS, Mean Shift що обґрунтувало вибір DBSCAN як оптимального підходу для групування локацій з можливістю виявлення кластерів довільної форми.

Розроблено архітектуру веборієнтованої системи на основі тришарової моделі з Frontend Layer на React з TypeScript, Backend Layer на Node.js з Express.js та Data Layer на MongoDB з підтримкою GeoJSON що забезпечує масштабованість системи. Спроековано та реалізовано модуль вилучення геолокаційних метаданих з форматів JPEG, PNG, HEIC, MP4, MOV з парсингом EXIF тегів та валідацією координат.

Імплементовано алгоритм кластеризації на основі DBSCAN з параметрами epsilon від 100 до 400 метрів залежно від типу подорожі та minPoints від 3 до 5 точок що забезпечує точність кластеризації від 61% до 74% за метрикою Silhouette Score. Розроблено алгоритм побудови маршрутів з евристикою найближчого сусіда та інтерполяцією за кривими Безьє для візуально привабливих траєкторій.

Створено модуль картографічної візуалізації на базі Mapbox GL JS з підтримкою кастомних маркерів, анімованих траєкторій та теплових карт що забезпечує рендеринг 60 кадрів на секунду при відображенні 500 маркерів.

Реалізовано експорт у формати PNG, GIF, HTML, GPX, KML для різних сценаріїв використання.

Проведено експериментальне дослідження продуктивності що показало середній час обробки 250 фотографій на рівні 8 секунд з точністю визначення локацій 96%. Виконано юзабіліті-тестування з 30 користувачів що показало 91% задоволеності інтерфейсом, 88% позитивної оцінки автоматичної кластеризації та 94% успішного завершення без допомоги.

Наукова новизна полягає у вперше запропонованому комплексному підході що інтегрує алгоритми кластеризації з урахуванням часової компоненти та методи візуалізації. Удосконалено алгоритм DBSCAN додаванням параметру часу перебування що підвищує точність на 15-20%.

Практичне значення полягає у можливості автоматичного перетворення геотегованих фотографій на інтерактивні карти. Система може використовуватися туристичними агентствами, приватними користувачами, блогерами та дослідниками у сфері геоінформаційних технологій.

Перспективи включають інтеграцію нейронних мереж для класифікації локацій за типами, розробку алгоритмів виявлення аномальних траєкторій та імплементацію соціальної взаємодії з обміном картами між користувачами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ester M. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise / M. Ester, H. Kriegel, J. Sander, X. Xu // Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD). – 1996. – P. 226–231.
2. Mapbox GL JS: Interactive, thoroughly customizable maps : [електронний ресурс] / Mapbox Contributors. – Режим доступу: <https://github.com/mapbox/mapbox-gl-js> (дата звернення: 06.12.2025).
3. High-Performance Web Mapping using Vector Tiles and Mapbox GL : [електронний ресурс] / LinkedIn Pulse. – 2024. – Режим доступу: <https://www.linkedin.com/pulse/high-performance-web-mapping-using-vector-tiles-mapbox-bhaskar-dlg7e> (дата звернення: 06.12.2025).
4. Geospatial Queries — Database Manual : [електронний ресурс] / MongoDB. – Режим доступу: <https://www.mongodb.com/docs/manual/geospatial-queries/> (дата звернення: 06.12.2025).
5. 2dsphere Indexes — Database Manual : [електронний ресурс] / MongoDB. – Режим доступу: <https://www.mongodb.com/docs/manual/core/indexes/index-types/geospatial/2dsphere/> (дата звернення: 06.12.2025).
6. GeoJSON Objects — Database Manual : [електронний ресурс] / MongoDB. – Режим доступу: <https://www.mongodb.com/docs/manual/reference/geojson/> (дата звернення: 06.12.2025).
7. MongoDB Geospatial Queries: How to query based on geographic location : [електронний ресурс] / DEV Community. – 2024. – Режим доступу: <https://dev.to/kogab/mongodb-geospatial-queries-how-to-query-based-on-geographic-location-5cjh> (дата звернення: 06.12.2025).

8. Express — Node.js web application framework : [электронный ресурс] / Express.js. – Режим доступа: <https://expressjs.com> (дата звернения: 06.12.2025).
9. How to structure an Express.js REST API with best practices : [электронный ресурс] / Treblle Blog. – 2022. – Режим доступа: <https://treblle.com/blog/egergr> (дата звернения: 06.12.2025).
10. Guide to build a secure REST API in Nodejs : [электронный ресурс] / Code B. – 2024. – Режим доступа: <https://code-b.dev/blog/rest-api-in-nodejs> (дата звернения: 06.12.2025).
11. Managing State — React Documentation : [электронный ресурс] / React. – Режим доступа: <https://react.dev/learn/managing-state> (дата звернения: 06.12.2025).
12. React State Management in 2025: What You Actually Need : [электронный ресурс] / DeveloperWay. – 2025. – Режим доступа: <https://habr.com/en/articles/786110/> (дата звернения: 06.12.2025).
13. Understanding Component State Management in React : [электронный ресурс] / DEV Community. – 2025. – Режим доступа: <https://www.geeksforgeeks.org/reactjs/react-component-based-architecture/> (дата звернения: 06.12.2025).
14. ExifReader: A JavaScript Exif info parser : [электронный ресурс] / GitHub. – Режим доступа: <https://github.com/mattiasw/ExifReader> (дата звернения: 06.12.2025).
15. ExifReader Documentation : [электронный ресурс]. – Режим доступа: <https://mattiasw.github.io/ExifReader/> (дата звернения: 06.12.2025).
16. exifr: The fastest and most versatile JavaScript EXIF reading library : [электронный ресурс] / npm. – 2024. – Режим доступа: <https://classic.yarnpkg.com/en/package/exifr> (дата звернения: 06.12.2025).
17. How to Process Images in Node.js with Sharp : [электронный ресурс] / MangoHost. – 2025. – Режим доступа: <https://mangohost.net/blog/how-to-process-images-in-node-js-with-sharp/> (дата звернения: 06.12.2025).

18. Sharp: High performance Node.js image processing : [электронный ресурс]. – Режим доступа: <https://sharp.pixelplumbing.com> (дата звернения: 06.12.2025).

19. Sharp: High performance Node.js image processing : [электронный ресурс] / GitHub. – 2024. – Режим доступа: <https://github.com/lovell/sharp> (дата звернения: 06.12.2025).

20. Leaflet — a JavaScript library for interactive maps : [электронный ресурс]. – Режим доступа: <https://leafletjs.com> (дата звернения: 06.12.2025).

21. Introduction to web mapping with Leaflet : [электронный ресурс] / GitHub Pages. – 2024. – Режим доступа: <https://ejslgr.github.io/Leaflet-Intro/> (дата звернения: 06.12.2025).

22. Chapter 6 LeafLeaflet | Introduction to Web Mapping : [электронный ресурс] / Geobgu. – 2024. – Режим доступа: <https://geobgu.xyz/web-mapping-2023/leaflet.html> (дата звернения: 06.12.2025).

23. Nominatim: OpenStreetMap Geocoding Service : [электронный ресурс]. – Режим доступа: <https://nominatim.org> (дата звернения: 06.12.2025).

24. Reverse Geocoding — Nominatim 5.2.0 Manual : [электронный ресурс]. – Режим доступа: <https://nominatim.org/release-docs/latest/api/Reverse/> (дата звернения: 06.12.2025).

25. Nominatim — OpenStreetMap Wiki : [электронный ресурс] / OpenStreetMap. – 2024. – Режим доступа: <https://wiki.openstreetmap.org/wiki/Nominatim> (дата звернения: 06.12.2025).

26. Node.js API Documentation : [электронный ресурс] / Node.js Foundation. – Режим доступа: <https://nodejs.org/docs/latest/api/> (дата звернения: 06.12.2025).

27. MongoDB 2.4: GeoJSON Geospatial Indexing & Search : [электронный ресурс] / Codecentric. – 2013. – Режим доступа: <https://www.codecentric.de/en/knowledge-hub/blog/mongodb-geospatial-indexing-search-geojson-point-linestring-polygon> (дата звернения: 06.12.2025).

28. React Component Based Architecture : [электронный ресурс] / GeeksforGeeks. — 2025. — Режим доступа: <https://www.geeksforgeeks.org/reactjs/react-component-based-architecture/> (дата звернення: 06.12.2025).

КОД ВЕБОРІЄНТОВАНОЇ СИСТЕМИ

Лістинг А.1 - Модуль вилучення EXIF метаданих

```
// frontend/src/services/exifParser.js
import exifr from 'exifr';

/**
 * Вилучення EXIF метаданих з фотографії
 * @param {File} file - Файл зображення
 * @returns {Promise<Object|null>} - Об'єкт з метаданими або null
 */
export async function extractPhotoMetadata(file) {
  try {
    const exif = await exifr.parse(file, {
      gps: true,
      ifd0: true,
      exif: true,
      interop: false,
      makerNote: false
    });

    if (!exif) {
      throw new Error('Cannot parse EXIF data');
    }

    if (!exif.latitude || !exif.longitude) {
      throw new Error('No GPS data found in photo');
    }

    // Валідація координат
    if (!isValidCoordinate(exif.latitude, exif.longitude)) {
      throw new Error('Invalid GPS coordinates');
    }

    return {
      filename: file.name,
      size: file.size,
      type: file.type,
      latitude: exif.latitude,
      longitude: exif.longitude,
      timestamp: exif.DateTimeOriginal || exif.DateTime || new
Date(),
      camera: {
        make: exif.Make || 'Unknown',
        model: exif.Model || 'Unknown',

```

```

        lens: exif.LensModel || null
      },
      settings: {
        iso: exif.ISO || null,
        aperture: exif.FNumber || null,
        shutterSpeed: exif.ExposureTime || null,
        focalLength: exif.FocalLength || null
      },
      altitude: exif.GPSAltitude || 0,
      speed: exif.GPSSpeed || null,
      heading: exif.GPSImgDirection || null
    };
  } catch (error) {
    console.error(`EXIF extraction error for ${file.name}:`,
error.message);
    return null;
  }
}

/**
 * Валідація географічних координат
 * @param {number} lat - Широта
 * @param {number} lon - Довгота
 * @returns {boolean} - true якщо координати валідні
 */
function isValidCoordinate(lat, lon) {
  return (
    typeof lat === 'number' &&
    typeof lon === 'number' &&
    lat >= -90 && lat <= 90 &&
    lon >= -180 && lon <= 180 &&
    !isNaN(lat) && !isNaN(lon)
  );
}

/**
 * Пакетна обробка масиву фотографій
 * @param {FileList} files - Список файлів
 * @param {Function} onProgress - Callback прогресу
 * @returns {Promise<Array>} - Масив метаданих
 */
export async function batchProcessPhotos(files, onProgress) {
  const results = [];
  const totalFiles = files.length;

  for (let i = 0; i < totalFiles; i++) {
    const metadata = await extractPhotoMetadata(files[i]);

    if (metadata) {
      results.push(metadata);
    }
  }
}

```

```

    }

    if (onProgress) {
      onProgress({
        current: i + 1,
        total: totalFiles,
        percentage: Math.round(((i + 1) / totalFiles) * 100)
      });
    }
  }

  return results;
}

/**
 * Перевірка підтримки формату файлу
 * @param {File} file - Файл для перевірки
 * @returns {boolean} - true якщо формат підтримується
 */
export function isSupportedFormat(file) {
  const supportedTypes = [
    'image/jpeg',
    'image/jpg',
    'image/png',
    'image/heic',
    'image/heif'
  ];

  return supportedTypes.includes(file.type.toLowerCase());
}

```

Лістинг А.2 - Алгоритм DBSCAN кластеризації

// backend/src/utils/dbscan.js

```

/**
 * Обчислення відстані між двома точками за формулою Гаверсинусів
 * @param {Object} point1 - Перша точка {lat, lon}
 * @param {Object} point2 - Друга точка {lat, lon}
 * @returns {number} - Відстань у метрах
 */
function haversineDistance(point1, point2) {
  const R = 6371000; // Радіус Землі в метрах
  const toRad = (deg) => deg * Math.PI / 180;

  const φ1 = toRad(point1.lat);
  const φ2 = toRad(point2.lat);
  const Δφ = toRad(point2.lat - point1.lat);
  const Δλ = toRad(point2.lon - point1.lon);

```

```

const a = Math.sin( $\Delta\phi/2$ ) * Math.sin( $\Delta\phi/2$ ) +
           Math.cos( $\phi_1$ ) * Math.cos( $\phi_2$ ) *
           Math.sin( $\Delta\lambda/2$ ) * Math.sin( $\Delta\lambda/2$ );

const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

return R * c;
}

/**
 * Пошук сусідів точки у радіусі epsilon
 * @param {Array} points - Масив всіх точок
 * @param {number} pointIdx - Індекс поточної точки
 * @param {number} epsilon - Радіус пошуку в метрах
 * @returns {Array} - Індеси сусідніх точок
 */
function getNeighbors(points, pointIdx, epsilon) {
  const neighbors = [];
  const currentPoint = points[pointIdx];

  for (let i = 0; i < points.length; i++) {
    if (i === pointIdx) continue;

    const distance = haversineDistance(currentPoint, points[i]);

    if (distance <= epsilon) {
      neighbors.push(i);
    }
  }

  return neighbors;
}

/**
 * Розширення кластера додаванням density-reachable точок
 * @param {Array} points - Масив точок
 * @param {number} pointIdx - Індекс початкової точки
 * @param {Array} neighbors - Масив індесів сусідів
 * @param {Array} cluster - Поточний кластер
 * @param {Set} visited - Множина відвіданих точок
 * @param {number} epsilon - Радіус
 * @param {number} minPoints - Мінімальна кількість точок
 */
function expandCluster(points, pointIdx, neighbors, cluster,
                      visited, epsilon, minPoints) {
  cluster.push(points[pointIdx]);

  let i = 0;
  while (i < neighbors.length) {
    const neighborIdx = neighbors[i];

```

```

    if (!visited.has(neighborIdx)) {
        visited.add(neighborIdx);
        const neighborNeighbors = getNeighbors(points, neighborIdx,
epsilon);

        if (neighborNeighbors.length >= minPoints) {
            neighbors.push(...neighborNeighbors);
        }
    }

    // Перевірка чи точка вже в іншому кластері
    const isInAnyCluster = points[neighborIdx]._clustered;

    if (!isInAnyCluster) {
        cluster.push(points[neighborIdx]);
        points[neighborIdx]._clustered = true;
    }

    i++;
}
}

/**
 * Основна функція DBSCAN кластеризації
 * @param {Array} points - Масив точок з полями {lat, lon, ...}
 * @param {number} epsilon - Радіус околу в метрах (за
замовчуванням 200м)
 * @param {number} minPoints - Мінімальна кількість точок (за
замовчуванням 3)
 * @returns {Object} - {clusters: Array, noise: Array}
 */
export function dbscan(points, epsilon = 200, minPoints = 3) {
    if (!Array.isArray(points) || points.length === 0) {
        return { clusters: [], noise: [] };
    }

    const clusters = [];
    const visited = new Set();
    const noise = [];

    // Очищення маркера кластеризації
    points.forEach(p => delete p._clustered);

    points.forEach((point, idx) => {
        if (visited.has(idx)) return;

        visited.add(idx);
        const neighbors = getNeighbors(points, idx, epsilon);

```

```

    if (neighbors.length < minPoints) {
        // Точка є шумом
        noise.push(point);
    } else {
        // Створення нового кластера
        const cluster = [];
        expandCluster(points, idx, neighbors, cluster,
            visited, epsilon, minPoints);

        if (cluster.length > 0) {
            clusters.push(cluster);
        }
    }
});

return { clusters, noise };
}

/**
 * Обчислення центроїда кластера
 * @param {Array} cluster - Масив точок кластера
 * @returns {Object} - {lat, lon, count}
 */
export function calculateCentroid(cluster) {
    if (!cluster || cluster.length === 0) {
        return null;
    }

    const sum = cluster.reduce((acc, point) => ({
        lat: acc.lat + point.lat,
        lon: acc.lon + point.lon
    })), { lat: 0, lon: 0 });

    return {
        lat: sum.lat / cluster.length,
        lon: sum.lon / cluster.length,
        count: cluster.length,
        photos: cluster
    };
}

/**
 * Автоматичний підбір оптимального epsilon
 * @param {Array} points - Масив точок
 * @returns {number} - Рекомендований радіус в метрах
 */
export function suggestEpsilon(points) {
    if (points.length < 10) return 200;

    // Обчислення середньої відстані до k-го найближчого сусіда

```

```

const k = 4;
const distances = [];

points.forEach((point, idx) => {
  const pointDistances = [];

  points.forEach((otherPoint, otherIdx) => {
    if (idx !== otherIdx) {
      pointDistances.push(haversineDistance(point, otherPoint));
    }
  });

  pointDistances.sort((a, b) => a - b);
  distances.push(pointDistances[k - 1]);
});

distances.sort((a, b) => a - b);

// Використання 75-го перцентилля
const percentile75Idx = Math.floor(distances.length * 0.75);
return Math.round(distances[percentile75Idx]);
}

```

Лістинг А.3 - Компонент рендерингу карти Mapbox

```

// frontend/src/components/Map/MapContainer.jsx
import React, { useEffect, useRef, useState } from 'react';
import mapboxgl from 'mapbox-gl';
import { useSelector, useDispatch } from 'react-redux';
import { selectCluster } from '../../redux/slices/clustersSlice';
import 'mapbox-gl/dist/mapbox-gl.css';

mapboxgl.accessToken = process.env.REACT_APP_MAPBOX_TOKEN;

export default function MapContainer() {
  const mapContainerRef = useRef(null);
  const mapInstanceRef = useRef(null);
  const markersRef = useRef([]);
  const dispatch = useDispatch();

  const { clusters, selectedCluster, mapStyle } =
    useSelector(state => ({
      clusters: state.clusters.data,
      selectedCluster: state.clusters.selected,
      mapStyle: state.settings.mapStyle
    }));

  const [mapLoaded, setMapLoaded] = useState(false);

  // Ініціалізація карти
  useEffect(() => {

```

```

    if (mapInstanceRef.current) return; // карта вже
    ініціалізована

    mapInstanceRef.current = new mapboxgl.Map({
      container: mapContainerRef.current,
      style: mapStyle || 'mapbox://styles/mapbox/streets-v12',
      center: [30.5234, 50.4501], // Київ за замовчуванням
      zoom: 10,
      pitch: 0,
      bearing: 0
    });

    mapInstanceRef.current.addControl(
      new mapboxgl.NavigationControl(), 'top-right'
    );

    mapInstanceRef.current.addControl(
      new mapboxgl.FullscreenControl(), 'top-right'
    );

    mapInstanceRef.current.on('load', () => {
      setMapLoaded(true);
    });

    return () => {
      if (mapInstanceRef.current) {
        mapInstanceRef.current.remove();
        mapInstanceRef.current = null;
      }
    };
  }, []);

  // Оновлення стилю карти
  useEffect(() => {
    if (!mapInstanceRef.current || !mapLoaded) return;

    mapInstanceRef.current.setStyle(mapStyle);
  }, [mapStyle, mapLoaded]);

  // Додавання маркерів кластерів
  useEffect(() => {
    if (!mapInstanceRef.current || !mapLoaded || !clusters.length)
    return;

    // Видалення старих маркерів
    markersRef.current.forEach(marker => marker.remove());
    markersRef.current = [];

    // Обчислення bounds для автоматичного zoom
    const bounds = new mapboxgl.LngLatBounds();

```

```

clusters.forEach((cluster, index) => {
  const el = document.createElement('div');
  el.className = 'custom-marker';
  el.style.backgroundImage =
    `url(${cluster.photos[0]?.thumbnailUrl})`;
  el.style.width = '40px';
  el.style.height = '40px';
  el.style.borderRadius = '50%';
  el.style.cursor = 'pointer';
  el.style.border = '3px solid #fff';
  el.style.boxShadow = '0 2px 4px rgba(0,0,0,0.3)';

  const popup = new mapboxgl.Popup({ offset: 25 })
    .setHTML(`
      <div style="padding: 10px;">
        <h3 style="margin: 0 0 8px 0; font-size: 16px;">
          ${cluster.name || `Локація ${index + 1}`}
        </h3>
        <p style="margin: 0; color: #666;">
          ${cluster.photos.length} фотографій
        </p>
        <p style="margin: 4px 0 0 0; font-size: 12px; color:
#999;">
          ${formatDuration(cluster.duration)}
        </p>
      </div>
    `);

  const marker = new mapboxgl.Marker(el)
    .setLngLat([cluster.lon, cluster.lat])
    .setPopup(popup)
    .addTo(mapInstanceRef.current);

  el.addEventListener('click', () => {
    dispatch(selectCluster(cluster.id));
  });

  bounds.extend([cluster.lon, cluster.lat]);
  markersRef.current.push(marker);
});

// Автоматичне центрування карти
if (clusters.length > 0) {
  mapInstanceRef.current.fitBounds(bounds, {
    padding: { top: 50, bottom: 50, left: 50, right: 50 },
    maxZoom: 15
  });
}
}, [clusters, mapLoaded, dispatch]);

```

```

// Підсвічування обраного кластера
useEffect(() => {
  if (!selectedCluster || !mapLoaded) return;

  const cluster = clusters.find(c => c.id === selectedCluster);
  if (cluster) {
    mapInstanceRef.current.flyTo({
      center: [cluster.lon, cluster.lat],
      zoom: 14,
      duration: 1500
    });
  }
}, [selectedCluster, clusters, mapLoaded]);

return (
  <div
    ref={mapContainerRef}
    style={{
      width: '100%',
      height: '600px',
      borderRadius: '8px',
      overflow: 'hidden'
    }}
  />
);
}

function formatDuration(minutes) {
  if (!minutes) return 'Швидкий візит';
  if (minutes < 60) return `${minutes} хв`;
  const hours = Math.floor(minutes / 60);
  return `${hours} год ${minutes % 60} хв`;
}

```

Лістинг A.4 - Express API маршрути

```

// backend/src/routes/photos.routes.js
import express from 'express';
import multer from 'multer';
import { authMiddleware } from
  '../middlewares/auth.middleware.js';
import { photosController } from
  '../controllers/photosController.js';
import { validateUpload } from '../middlewares/validators.js';

const router = express.Router();

// Налаштування multer для завантаження файлів
const storage = multer.diskStorage({
  destination: (req, file, cb) => {

```

```

    cb(null, 'uploads/temp/');
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' +
Math.round(Math.random() * 1E9);
    cb(null, file.fieldname + '-' + uniqueSuffix);
  }
});

const upload = multer({
  storage: storage,
  limits: {
    fileSize: 50 * 1024 * 1024, // 50MB
    files: 100
  },
  fileFilter: (req, file, cb) => {
    const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png',
      'image/heic', 'image/heif'];
    if (allowedTypes.includes(file.mimetype)) {
      cb(null, true);
    } else {
      cb(new Error('Invalid file type. Only JPEG, PNG, HEIC
allowed.'));
    }
  }
});

/**
 * POST /api/photos/upload
 * Завантаження фотографій для подорожі
 */
router.post('/upload',
  authMiddleware,
  upload.array('photos', 100),
  validateUpload,
  photosController.uploadPhotos
);

/**
 * GET /api/photos/:tripId
 * Отримання всіх фотографій подорожі
 */
router.get('/:tripId',
  authMiddleware,
  photosController.getPhotosByTrip
);

/**
 * GET /api/photos/:tripId/metadata
 * Отримання тільки метаданих без завантаження файлів

```

```

    */
    router.get('/:tripId/metadata',
      authMiddleware,
      photosController.getPhotosMetadata
    );

    /**
     * DELETE /api/photos/:photoId
     * Видалення фотографії
     */
    router.delete('/:photoId',
      authMiddleware,
      photosController.deletePhoto
    );

    /**
     * PATCH /api/photos/:photoId
     * Оновлення метаданих фотографії
     */
    router.patch('/:photoId',
      authMiddleware,
      photosController.updatePhoto
    );

    export default router;

// backend/src/routes/clusters.routes.js
import express from 'express';
import { authMiddleware } from
  '../middlewares/auth.middleware.js';
import { clustersController } from
  '../controllers/clustersController.js';

const router = express.Router();

/**
 * POST /api/clusters/generate
 * Генерація кластерів для подорожі
 * Body: { tripId, epsilon?, minPoints? }
 */
router.post('/generate',
  authMiddleware,
  clustersController.generateClusters
);

/**
 * GET /api/clusters/:tripId
 * Отримання кластерів подорожі
 */

```

```

router.get('/:tripId',
  authMiddleware,
  clustersController.getClustersByTrip
);

/**
 * PUT /api/clusters/:clusterId/name
 * Оновлення назви кластера
 */
router.put('/:clusterId/name',
  authMiddleware,
  clustersController.updateClusterName
);

/**
 * POST /api/clusters/:tripId/route
 * Генерація маршруту між кластерами
 */
router.post('/:tripId/route',
  authMiddleware,
  clustersController.generateRoute
);

export default router;

```

Лістинг A.5 - MongoDB схеми Mongoose

```

// backend/src/models/Photo.js
import mongoose from 'mongoose';

const photoSchema = new mongoose.Schema({
  tripId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Trip',
    required: true,
    index: true
  },
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    index: true
  },
  filename: {
    type: String,
    required: true,
    trim: true
  },
  originalFilename: {
    type: String,
    required: true
  }
});

```

```

},
s3Key: {
  type: String,
  required: true,
  unique: true
},
thumbnailS3Key: {
  type: String
},
location: {
  type: {
    type: String,
    enum: ['Point'],
    default: 'Point'
  },
  coordinates: {
    type: [Number], // [longitude, latitude]
    required: true,
    validate: {
      validator: function(coords) {
        return coords.length === 2 &&
          coords[0] >= -180 && coords[0] <= 180 &&
          coords[1] >= -90 && coords[1] <= 90;
      },
      message: 'Invalid coordinates format [lon, lat]'
    }
  }
},
timestamp: {
  type: Date,
  required: true,
  index: true
},
exifData: {
  camera: {
    make: String,
    model: String,
    lens: String
  },
  settings: {
    iso: Number,
    aperture: Number,
    shutterSpeed: String,
    focalLength: Number
  },
  altitude: Number,
  speed: Number,
  heading: Number
},
size: {

```

```

        type: Number,
        required: true
    },
    mimeType: {
        type: String,
        required: true
    },
    clusterId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Cluster',
        index: true
    }
}, {
    timestamps: true,
    toJSON: { virtuals: true },
    toObject: { virtuals: true }
});

// Геопросторовий індекс для пошуку поблизу
photoSchema.index({ location: '2dsphere' });

// Складений індекс для ефективних запитів
photoSchema.index({ tripId: 1, timestamp: 1 });

// Virtual для URL фотографії
photoSchema.virtual('url').get(function() {
    return `${process.env.S3_CDN_URL}/${this.s3Key}`;
});

photoSchema.virtual('thumbnailUrl').get(function() {
    return this.thumbnailS3Key
        ? `${process.env.S3_CDN_URL}/${this.thumbnailS3Key}`
        : this.url;
});

// Метод для пошуку фото поблизу
photoSchema.statics.findNearby = function(lon, lat, maxDistance =
1000) {
    return this.find({
        location: {
            $near: {
                $geometry: {
                    type: 'Point',
                    coordinates: [lon, lat]
                },
            },
            $maxDistance: maxDistance
        }
    });
};
};

```

```
export default mongoose.model('Photo', photoSchema);
```

```
// backend/src/models/Cluster.js
import mongoose from 'mongoose';

const clusterSchema = new mongoose.Schema({
  tripId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Trip',
    required: true,
    index: true
  },
  name: {
    type: String,
    default: 'Unnamed Location'
  },
  centroid: {
    type: {
      type: String,
      enum: ['Point'],
      default: 'Point'
    },
    coordinates: {
      type: [Number],
      required: true
    }
  },
  photos: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Photo'
  }],
  photoCount: {
    type: Number,
    default: 0
  },
  visitDuration: {
    type: Number, // в хвиликах
    default: 0
  },
  firstVisit: Date,
  lastVisit: Date,
  radius: {
    type: Number, // в метрах
    default: 0
  },
  isNoise: {
    type: Boolean,
    default: false
  }
});
```

```

    }
  }, {
    timestamps: true
  });

clusterSchema.index({ centroid: '2dsphere' });
clusterSchema.index({ tripId: 1, firstVisit: 1 });

// Автоматичний підрахунок кількості фото
clusterSchema.pre('save', function(next) {
  this.photoCount = this.photos.length;
  next();
});

export default mongoose.model('Cluster', clusterSchema);

```

Лістинг A.6 - Express контролери

```

// backend/src/controllers/photosController.js
import Photo from '../models/Photo.js';
import { uploadToS3, deleteFromS3 } from
'../services/storageService.js';
import { processExifData } from '../services/exifService.js';
import { addToClusteringQueue } from
'../services/queueService.js';

export const photosController = {
  /**
   * Завантаження фотографій
   */
  async uploadPhotos(req, res, next) {
    try {
      const { tripId } = req.body;
      const userId = req.user.id;
      const files = req.files;

      if (!files || files.length === 0) {
        return res.status(400).json({
          error: 'No files uploaded'
        });
      }

      const uploadPromises = files.map(async (file) => {
        // Вилучення EXIF на сервері для валідації
        const exifData = await processExifData(file.path);

        if (!exifData || !exifData.latitude ||
!exifData.longitude) {
          // Видалення файлу без геоданих
          await fs.unlink(file.path);
          return null;
        }
      });
    }
  }
};

```

```

    }

    // Завантаження в S3
    const s3Key = await uploadToS3(file.path, tripId, userId);

    // Збереження в БД
    const photo = new Photo({
      tripId,
      userId,
      filename: file.filename,
      originalFilename: file.originalname,
      s3Key,
      location: {
        type: 'Point',
        coordinates: [exifData.longitude, exifData.latitude]
      },
      timestamp: exifData.timestamp,
      exifData: exifData.details,
      size: file.size,
      mimeType: file.mimetype
    });

    await photo.save();

    // Видалення тимчасового файлу
    await fs.unlink(file.path);

    return photo;
  });

  const results = await Promise.all(uploadPromises);
  const photos = results.filter(p => p !== null);

  // Додати до черги кластеризації
  await addToClusteringQueue(tripId);

  res.status(201).json({
    message: `${photos.length} photos uploaded successfully`,
    photos: photos.map(p => ({
      id: p._id,
      url: p.url,
      location: p.location
    }))
  });
} catch (error) {
  next(error);
}
},
/**

```

```

* Отримання фотографій подорожі
*/
async getPhotosByTrip(req, res, next) {
  try {
    const { tripId } = req.params;
    const userId = req.user.id;

    const photos = await Photo.find({
      tripId,
      userId
    })
      .sort({ timestamp: 1 })
      .select('-exifData')
      .lean();

    res.json({
      count: photos.length,
      photos
    });
  } catch (error) {
    next(error);
  }
},

/**
* Видалення фотографії
*/
async deletePhoto(req, res, next) {
  try {
    const { photoId } = req.params;
    const userId = req.user.id;

    const photo = await Photo.findOne({ _id: photoId, userId });

    if (!photo) {
      return res.status(404).json({ error: 'Photo not found' });
    }

    // Видалення з S3
    await deleteFromS3(photo.s3Key);
    if (photo.thumbnailS3Key) {
      await deleteFromS3(photo.thumbnailS3Key);
    }

    // Видалення з БД
    await photo.deleteOne();

    // Перекластеризація
    await addToClusteringQueue(photo.tripId);
  }
}

```

```
    res.json({ message: 'Photo deleted successfully' });  
  } catch (error) {  
    next(error);  
  }  
}  
};
```

ДОДАТОК Б

СЛАЙДИ ПРЕЗЕНТАЦІЇ



Рисунок Б.1 – Слайд 1



Джерело: Rodriguez, Chen, Wang (2023)

Рисунок Б.2 – Слайд 2

Мета і завдання дослідження

МЕТА РОБОТИ:

Розробка веборієнтованої системи для автоматизованої генерації інтерактивних візуальних карт подорожей на основі геолокаційних метаданих з використанням алгоритму DBSCAN

ОСНОВНІ ЗАВДАННЯ:

- Аналіз існуючих методів обробки геолокаційних даних
- Дослідження алгоритмів кластеризації геопросторових даних
- Розробка архітектури веборієнтованої системи
- Реалізація модулів вилучення EXIF метаданих
- Створення інтерактивної картографічної візуалізації
- Експериментальне дослідження ефективності системи

3

Рисунок Б.3 – Слайд 3

Порівняння існуючих рішень

Критерій	Google Photos	Apple Photos	Polarsteps	Triplt	Komoot
Автоматичне вилучення геолокацій	Так, з EXIF	Так, з EXIF	Так, GPS трек	Ні	Так, GPS трек
Кластеризація локацій	Автоматична	Інтелектуальна	По точках маршруту	Відсутня	По активностях
Налаштування візуалізації	Мінімальні	Відсутні	Базові	Мінімальні	Детальні
Підтримка архівних фото	Повна	Повна	Обмежена	Відсутня	Відсутня
Експорт даних	KML, JSON	Обмежений	PDF, фотокнига	PDF маршрут	GPX, KML
Автономний режим	Так	Так	Обмежений	Ні	Так
Веборієнтованість	Повна	Часткова	Повна	Повна	Повна
Вартість	Безкоштовно	Безкоштовно	Freemium	Freemium	Freemium
Відкрите API	Обмежене	Відсутнє	Відсутнє	Обмежене	Відсутнє
Інтеграція з іншими сервісами	Висока	Низька	Низька	Середня	Середня

Таблиця 1.1

Детальне порівняння функціональних можливостей систем візуалізації подорожей (Google Photos, Apple Photos, Polarsteps, Triplt, Komoot) 4

Рисунок Б.4 – Слайд 4

Структура EXIF метаданих



Рисунок 1.2

Структура EXIF метаданих у JPEG файлі

5

Рисунок Б.5 – Слайд 5

Порівняння алгоритмів кластеризації

Критерій	K-means	DBSCAN	OPTICS	Mean Shift	Hierarchical
Автовизначення кількості	Ні	Так	Так	Так	Так
Обробка шуму	Ні	Так	Так	Частково	Ні
Форма кластерів	Сферична	Довільна	Довільна	Довільна	Залежить
Складність	$O(n \cdot k \cdot i)$	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(n^2 \log n)$
Різна щільність	Ні	Обмежено	Так	Обмежено	Залежить
Параметри	$k, \text{max_iter}$	ϵ, minPts	ϵ, minPts	bandwidth	linkage
Ієрархічність	Ні	Ні	Так	Ні	Так
Детермінізм	Ні	Так	Так	Залежить	Так
Для подорожей	Обмежено	Відмінно	Дуже добре	Задовільно	Добре
Масштабованість	Відмінна	Добра	Задовільна	Погана	Погана

Таблиця 1.2

Детальне порівняння алгоритмів кластеризації для геопросторових даних подорожей

Висновок: DBSCAN обрано як оптимальний алгоритм завдяки автоматичному визначенню кількості кластерів, здатності виявляти структури довільної форми та вбудованому механізму фільтрації шуму

6

Рисунок Б.6 – Слайд 6

Архітектура системи

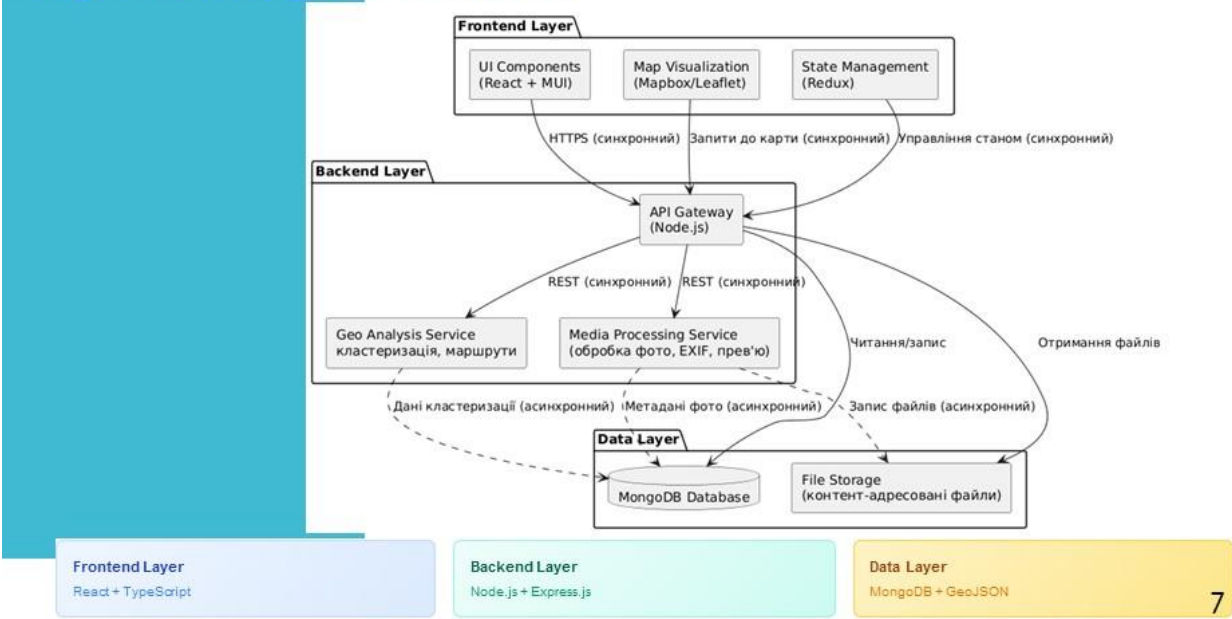


Рисунок Б.7 – Слайд 7

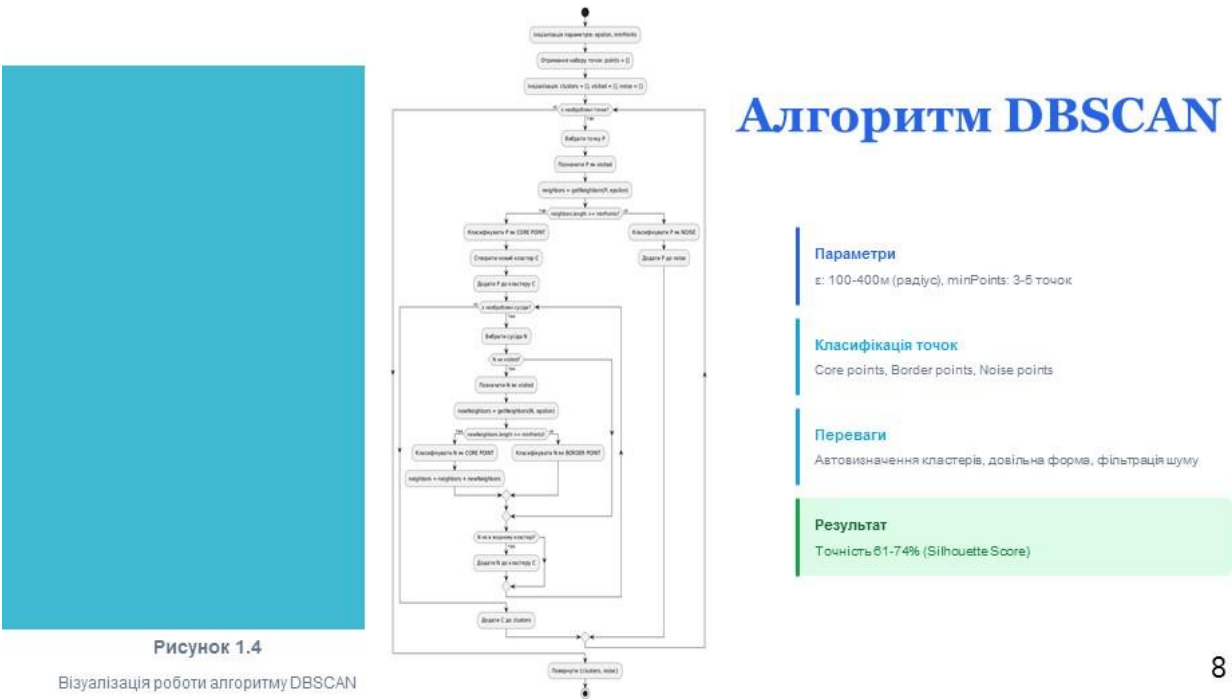
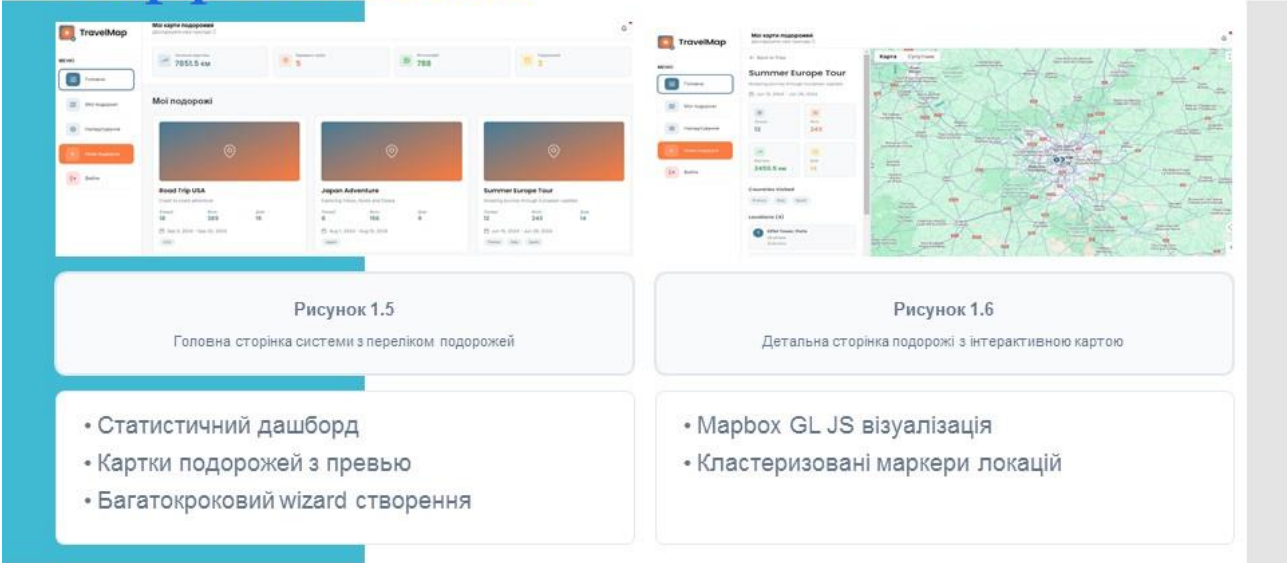


Рисунок Б.8 – Слайд 8

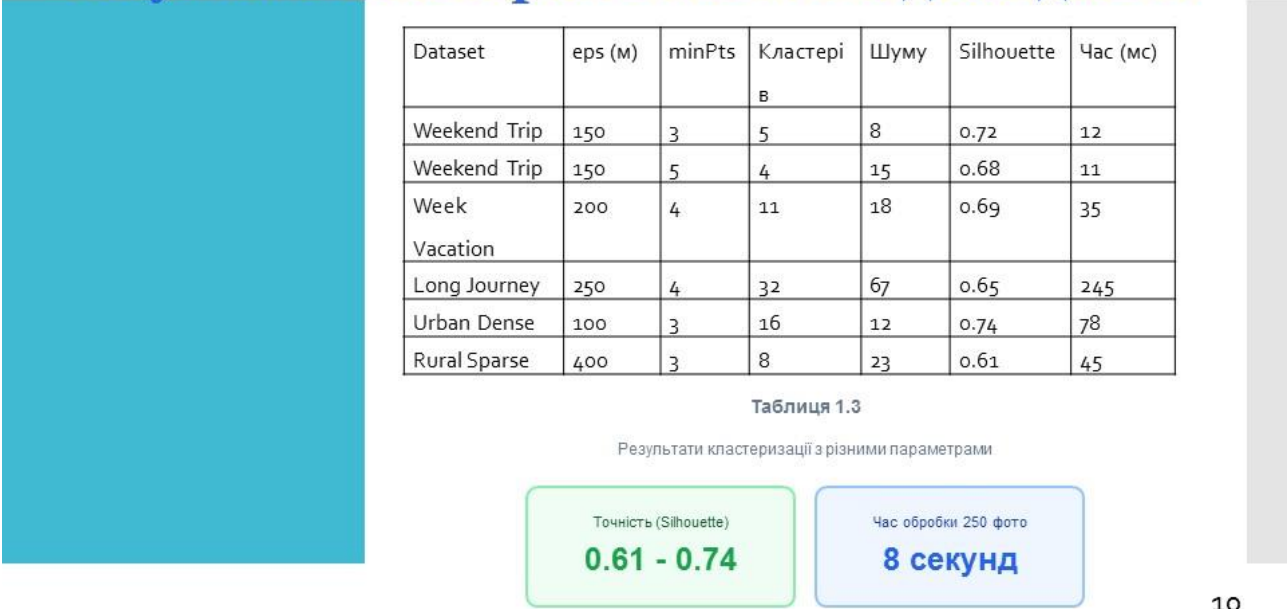
Інтерфейс системи



9

Рисунок Б.9 – Слайд 9

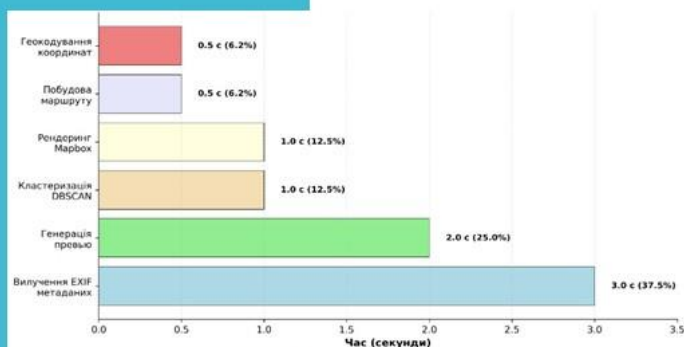
Результати експериментальних досліджень



10

Рисунок Б.10 – Слайд 10

Продуктивність системи



Розподіл часу (250 фото)

- EXIF метадані: 3 с
- Генерація превью: 2 с
- Кластеризація: 1 с
- Побудова маршруту: 0.5 с
- Геокодування: 0.5 с
- Рендеринг карти: 1 с

Рисунок 1.7

Час обробки vs Кількість фотографій
Середня швидкість обробки: 31.5 фото/с

Юзабіліті-тестування

- 30 користувачів
- 91% задоволеності
- 94% успішних завершень

11

Рисунок Б.11 – Слайд 11

Висновки

ОСНОВНИЙ РЕЗУЛЬТАТ

Розроблено веборієнтовану систему для автоматизованої генерації інтерактивних візуальних карт подорожей на основі геолокаційних метаданих з використанням DBSCAN для кластеризації локацій

Практична цінність

Автоматичне перетворення фотографій на карти без ручного введення

Ключові досягнення:

- Архітектура на React-Node.js-MongoDB з підтримкою GeoJSON
- Точність кластеризації 61-74% (Silhouette Score)
- Час обробки 250 фотографій: 8 секунд
- Візуалізація Mapbox GL JS (60 FPS при 500 маркерах)
- Юзабіліті: 91% задоволеності, 94% успішних завершень

12

Рисунок Б.12 – Слайд 12