Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

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

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ СОЦІАЛЬНОГО ІНТЕРНЕТ-СЕРВІСУ ДЛЯ ФОТОГРАФІВ
SOFTWARE IMPLEMENTATION OF A SOCIAL INTERNET SERVICE FOR PHOTOGRAPHERS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-141
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

АДЄЛЄВ Д.І.
(ПРІЗВИЩЕ та ініціали)
Керівник ЛЕОЩЕНКО Д.С.
(ПРІЗВИЩЕ та ініціали)
Рецензент БАБЕНКО Н.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

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

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“____” _____2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

АДЄЛЄВА Дмитра Івановича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація соціального інтернет-сервісу для фотографів.
Software Implementation of a Social Internet Service for Photographers

керівник проєкту (роботи) д-р філософ., доцент, ЛЕОЩЕНКО Сергій Дмитрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209
2. Строк подання студентом проєкту (роботи) 02 червня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Аналіз програмних засобів. 3. Розробка та реалізація системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка) _____ Дмитро АДЄЛЄВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Сергій ЛЕОЩЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 87 с., 10 табл., 40 рис., 3 дод., 18 джерел.

VISUAL STUDIO CODE, JAVA SCRIPT, VUE 3, POSTGRESQL, NODE.JS, EXPRESS, РОЗРОБКА, ВЕБЗАСТОСУНОК.

Об’єкт дослідження – сучасні інформаційні технології, що використовуються для розробки вебзастосунків, зокрема фреймворки JavaScript та засоби керування базами даних.

Метою роботи є дослідження процесу створення вебзастосунку, починаючи від архітектури клієнт-сервер, структури бази даних, до написання програмного коду та забезпечення взаємодії між інтерфейсом користувача й серверною частиною.

У результаті роботи було створено повноцінний вебзастосунок, який дає змогу користувачам взаємодіяти з динамічними даними, здійснювати пошук, фільтрацію, авторизацію, збереження інформації та комунікацію з сервером у режимі реального часу. Було реалізовано авторизацію користувача, інтеграцію з базою даних, а також інтерфейс, адаптований для зручного користування.

Висновок – були поглиблені знання щодо структури сучасних вебзастосунків, принципів їх розробки та використання новітніх інструментів. Робота підтвердила ефективність використання сучасних фреймворків та СУБД для побудови гнучких і масштабованих систем.

Галузь використання – застосунок може бути використаний як платформа для творців фотографій і публікації, для перегляду робіт інших та створення улюблених колекцій, або як комерційна платформа.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 87 pages, 10 tables, 40 figures, 3 appendixes, 18 sources.

VISUAL STUDIO CODE, JAVA SCRIPT, VUE 3, POSTGRESQL, NODE.JS, EXPRESS, DEVELOPMENT.

The object of research is modern information technologies used to develop web applications, in particular JavaScript frameworks and database management tools.

The purpose of the work is to study the process of creating a web application, starting from the client-server architecture, database structure, to writing program code and ensuring interaction between the user interface and the server side.

As a result of the work, a full-fledged web application was created that allows users to interact with dynamic data, search, filter, authorize, save information, and communicate with the server in real time. We implemented user authorization, database integration, and a user-friendly interface.

Conclusion - the knowledge of the structure of modern web applications, the principles of their development and the use of the latest tools was deepened. The work confirmed the effectiveness of using modern frameworks and DBMSs to build flexible and scalable systems.

Field of application - the application can be used as a platform for photo creators and publishers, to view the work of others and create favorite collections, or as a commercial platform.

ЗМІСТ

С.

Перелік скорочень та умовних познак 8

Вступ 9

1 Аналіз предметної області 11

1.1 Аналіз предметної області 11

1.2 Сучасний стан предметної області 13

1.3 Огляд та порівняння існуючих аналогів 14

1.3.1 Передмова 14

1.3.2 Сервіс 500px 15

1.3.3 Сервіс Flickr 16

1.3.4 Сервіс Behance 16

1.3.5 Таблиця порівняння існуючих онлайн фото-сервісів для фотографів 17

1.3.6 Висновки 19

1.4 Постановка завдання 19

1.4.1 Вимоги до інтерфейсу 19

1.4.2 Вимоги до безпеки 20

1.4.3 Основні вимоги до системи 20

1.5 Висновок розділу 21

2 Аналіз програмних засобів 22

2.1 Вибір мови програмування 22

2.2 Огляд особливостей обраних платформ і бібліотек 23

2.3 Вибір середовища розробки 24

2.4 Схема розробки 25

3 Розробка та реалізація системи 26
3.1 Загальна архітектура системи 26
3.2 Розробка бази даних 27
3.2.1 Схема бази даних 28
3.2.2 Опис таблиць 28
3.3 Реалізація реєстрації та авторизації користувачів 32
3.4 Публікація та збереження фотоматеріалів 33
3.5 Валідація та обробка помилок 34
3.6 Висновок до розділу 34
4 Експлуатація, тестування та експериментальне дослідження програми 36
4.1 Призначення й умови застосування програми 36
4.2 Вимоги до інтерфейсу програми 37
4.3 Умови застосування програми 38
4.4 Тестування програми 39
4.5 Висновок до розділу 51
Висновки 52
Перелік джерел посилання 54
Додаток А Технічне завдання 56
Додаток Б Текст програми 60
Додаток В Слайди презентації 80

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface, інтерфейс прикладного програмування;

CRUD – Create, Read, Update, Delete, базові операції з бекендом;

draw.io – онлайн-інструмент для створення діаграм та схем;

FormData – об'єкт для надсилання даних форми, включно з файлами, через HTTP-запит;

HTTP – протокол передачі даних між клієнтом і сервером в інтернеті;

JSON – JavaScript Object Notation, стандартний текстовий формат для представлення структурованих даних на основі об'єктного синтаксису JavaScript;

JWT-токен – JSON Web Token, використовуються для ідентифікації автентифікованого користувача;

ORM – Object-Relational Mapping, об'єктно-реляційне відображення;

Postman – інструмент для тестування API-запитів;

REST API – стиль архітектури API, де використовується HTTP і чіткі URL для роботи з ресурсами (GET, POST, PUT, DELETE);

Sequelize – ORM-бібліотека для Node.js;

SPA – односторінковий застосунок, де переходи відбуваються без перезавантаження сторінки;

UI – User Interface, користувацький інтерфейс;

UX – User Experience, досвід користувача;

БД – база даних.

ВСТУП

У сучасну цифрову епоху фотографія вийшла за межі своїх традиційних кордонів і стала потужним засобом художнього вираження, розповіді особистих історій та професійної комунікації. Зі стрімким розвитком інтернету та широким розповсюдженням смартфонів, оснащених високоякісними камерами, мільйони людей по всьому світу активно займаються фотографією - як хобі, так і в якості кар'єри.

Як наслідок, значно зріс попит на спеціалізовані онлайн-платформи, створені спеціально для фотографів. Соціальним мережам загального призначення часто бракує інструментів і функцій, орієнтованих на спільноту, яких прагнуть професійні фотографи та фотографи-початківці. Їм потрібно більше, ніж просто місце для завантаження зображень - їм потрібне середовище, яке підтримує творчий розвиток, конструктивний зворотній зв'язок, захист авторських прав і кураторство портфоліо.

Цей дипломний проєкт спрямований на розробку соціального інтернет-сервісу для фотографів - платформи, де користувачі можуть реєструватися, ділитися своїми фотографіями, досліджувати творчість інших та брати участь у значущій взаємодії в рамках сфокусованої спільноти. Сервіс покликаний сприяти популяризації мистецтва, заохочувати співпрацю та захищати інтелектуальну власність у цифровому середовищі, яке часто є вразливим до неправомірного використання.

Інтегруючи такі функції, як автентифікація користувачів, управління публікаціями, захист авторських прав, рейтингові системи та адміністративні інструменти, пропоноване рішення прагне заповнити прогалину між творчим самовираженням і технологічною функціональністю. Це не лише інструмент, а й цифровий простір, де візуальні оповідачі можуть процвітати, спілкуватися і розвиватися.

Не менш важливим є адміністративний рівень системи, який дозволяє модерувати контент, створений користувачами, і гарантує, що платформа залишатиметься безпечною, поважною і зосередженою на своїй основній меті.

Адміністративна панель надаватиме інструменти для управління користувачами, перегляду коментарів та виокремлення найбільш рейтингових фотографій на основі показників популярності та залученості. Таким чином, проєкт задовольняє як функціональні потреби окремих фотографів, так і ширші вимоги до управління спільнотою, пропонуючи масштабоване і змістовне рішення для цифрової епохи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі було розглянуто існуючі фото-сервіси, їхній функціонал та недоліки, визначено мету й завдання майбутнього вебдодатку, обґрунтовано вибір мікросервісної архітектур, сформульовано ключові функціональні й нефункціональні вимоги до системи, а також зроблено висновки щодо шляхів покращення існуючих рішень.

1.1 Аналіз предметної області

Фотографія стала життєво важливою складовою сучасної культури та суспільства. Це вже не просто засіб фіксації миттєвостей, а потужна форма самовираження, візуальної комунікації і навіть активізму. Через фотографію люди передають емоції, ідеї та наративи, які долають мовні бар'єри. У сучасному візуальному світі платформи, які підтримують і відзначають цей засіб, відіграють вирішальну роль у формуванні мистецьких і культурних ландшафтів.

Розробка спеціалізованого онлайн-порталу для фотографів спрямована на задоволення специфічних потреб зростаючої творчої спільноти. На відміну від універсальних соціальних медіа-платформ, для яких пріоритетом є обсяг і вірусність, сервіс, орієнтований на фотографію, фокусується на якості, презентації та творчих намірах. Його основна мета - створити простір, де фотографи - як професійні, так і аматори - можуть ділитися своїми роботами, створювати портфоліо та спілкуватися з однодумцями у значущий спосіб.

Потреба в такому додатку стає все більш очевидною в сучасній цифровій екосистемі. Хоча тисячі фотографій завантажуються в мережу щосекунди, дуже мало платформ пропонують механізми захисту творчої власності, структурований зворотній зв'язок або можливість кураторських мистецьких відкриттів. Спеціальний сервіс, розроблений для фотографів,

пропонує користувачам можливість брати участь у творчому обміні, зберігаючи при цьому контроль над своєю інтелектуальною власністю.

Більше того, такий тип платформи виконує подвійну функцію. З одного боку, вона діє як публічна галерея, доступна для всіх, де користувачі можуть переглядати, оцінювати та взаємодіяти з фотографічним контентом. З іншого боку, вона слугує особистим робочим простором для фотографів, де вони організовують свої колекції, черпають натхнення та керують видимістю своїх робіт. Така подвійна структура заохочує як особисте зростання, так і соціальну активність, що робить його комплексним рішенням для авторів.

Портал також інтегрує соціальні функції, такі як коментарі, що дозволить користувачам створювати аудиторію та отримувати зворотній зв'язок. Можливість створювати колекції з робіт інших користувачів сприяє взаємодії та співпраці спільноти, допомагаючи користувачам відкривати для себе нові перспективи та техніки. Крім того, такі функції, як адміністративна модерація коментарів та рекомендації контенту на основі популярності, забезпечують безпечну та цікаву роботу користувачів.

У інтернет-сервісу для фотографів є наступні основні функції:

- реєстрація та авторизація користувачів;
- завантаження та публікація особистого фотоконтенту;
- захист від несанкціонованого копіювання та порушення авторських прав;
- перегляд та дослідження фотографій інших користувачів;
- створення особистих колекцій з чужих робіт;
- оцінювання та коментування публікацій;
- адміністративна модерація коментарів та контенту;
- відображення та рекомендація фотографій з найвищим рейтингом на основі популярності;
- надання повнофункціональної адміністративної панелі для управління системою.

1.2 Сучасний стан предметної області

Останніми роками популярність фотографії різко зросла завдяки доступності високоякісних камер у смартфонах, поширенню соціальних мереж і зростаючому інтересу до візуального контенту. Як наслідок, з'явився широкий спектр платформ для обміну фотографіями, презентації портфоліо та творчої співпраці. Однак більшість із цих платформ, як-от Instagram, Facebook чи Pinterest, є універсальними і не повністю відповідають специфічним потребам фотографів.

Ці популярні сервіси надають перевагу швидкому споживанню контенту та алгоритмізованій взаємодії, що часто призводить до знецінення мистецької роботи та ускладнює виокремлення серйозних творців. Зазвичай їм бракує таких важливих інструментів, як вдосконалений захист авторських прав, структурований зворотній зв'язок, управління колекціями та формати презентації з високою роздільною здатністю. Це обмежує можливості фотографів професійно демонструвати свої портфоліо або здобувати довгострокове визнання у вузькому колі.

На противагу цьому, менша кількість платформ, таких як 500px, Flickr та Behance, мають на меті обслуговувати креативних професіоналів. Хоча ці платформи надають покращені інструменти для презентацій та функції професійного нетворкінгу, їм все ще не вистачає повної екосистеми, яка поєднує інтерактивність користувачів, захист авторських прав, модерацію контенту та інтелектуальні рекомендації - і все це в єдиному уніфікованому досвіді. Деякі з них обмежені платним доступом, застарілими інтерфейсами або недостатньою кількістю соціальних функцій, що знижує їхню доступність і потенціал залучення.

Поточний стан предметної області демонструє чіткий попит на сучасний спеціалізований інтернет-сервіс, який не лише полегшує обмін фотографіями, але й підтримує розвиток спільноти, забезпечує змістовний зворотній зв'язок, захищає права авторів та сприяє створенню високоякісного

контенту. Проєкт, запропонований у цій дипломній роботі, спрямований на усунення цих недоліків шляхом розробки платформи, яка об'єднує творчі, соціальні та адміністративні потреби фотографів у єдине рішення.

1.3 Огляд та порівняння існуючих аналогів

1.3.1 Передмова

Щоб краще зрозуміти сильні та слабкі сторони існуючих платформ для фотографів, важливо провести порівняльний аналіз кількох відомих онлайн-сервісів. Цей аналіз дозволяє визначити спільні риси цих платформ, виклики, з якими все ще стикаються фотографи, а також сфери, в яких сучасні рішення є недостатніми. Мета полягає в тому, щоб навчитися на цих платформах і використати отримані знання для розробки більш функціонального, зручного та безпечного сервісу, пристосованого спеціально до потреб фотографів.

Наступні три платформи - 500px, Flickr та Behance - були обрані для цього порівняння через їхню популярність серед фотографів, творчих професіоналів та ентузіастів. Кожен з цих сервісів має унікальні функції та підходи до обміну фотографіями, нетворкінгу та презентації портфоліо.

Платформи будуть оцінюватися за такими критеріями:

- інтерфейс та досвід користувача: зручність навігації, якість дизайну та швидкість реагування;
- портфоліо та фотопрезентація: інструменти для завантаження, організації та демонстрації робіт;
- соціальна взаємодія: коментарі, вподобання, поширення та підписка;
- залучення спільноти: видимість, основний контент та активність спільноти;
- захист авторських прав: інструменти для захисту зображень від несанкціонованого використання;

- адміністративні інструменти: можливості модерації, звітування та управління контентом;
- доступність і вартість: безкоштовні та платні функції, обмеження та загальна доступність;
- система рекомендацій: наскільки добре платформа просуває популярний або високоякісний контент.

1.3.2 Сервіс 500px

500px - це фотоплатформа, орієнтована на демонстрацію високоякісних робіт професійних фотографів та фотографів-аматорів [1]. Платформа робить акцент на створенні портфоліо та зворотному зв'язку зі спільнотою.

Інтерфейс вебзастосунку 500px показано на рис. 1.1 [1].

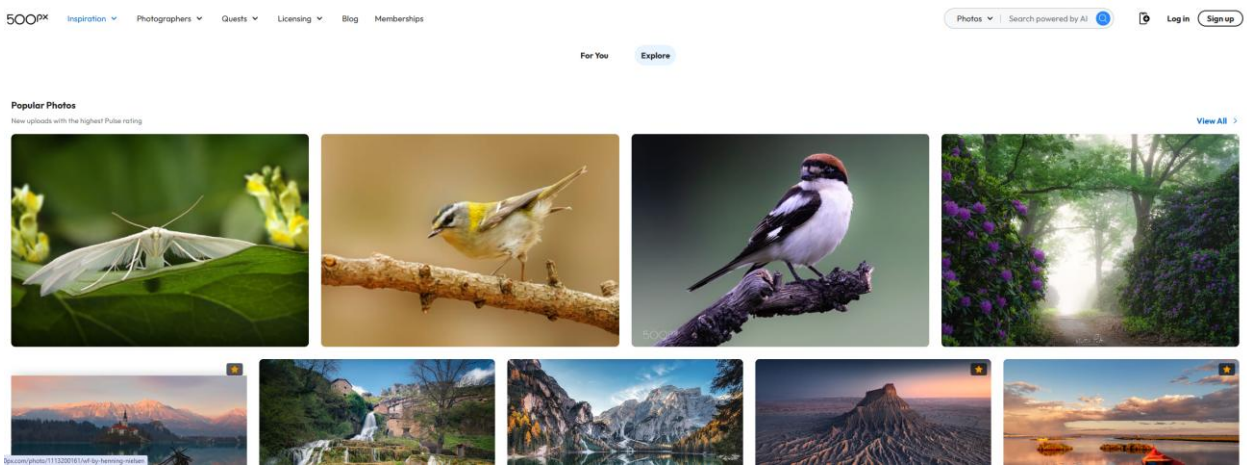


Рисунок 1.1 – Інтерфейс вебзастосунку 500px [1]

Переваги:

- завантаження фотографій у високій роздільній здатності та красиві макети;
- ліцензування та маркетплейси для монетизації;
- голосування спільноти та система зворотного зв'язку;
- чистий і професійний інтерфейс.

Недоліки:

- деякі функції доступні лише для платних акаунтів;
- менша орієнтація на випадкових користувачів або користувачів-початківців;
- обмежені можливості соціальної взаємодії;
- маркетплейс не дуже активний для нових користувачів.

1.3.3 Сервіс Flickr

Flickr - одна з найстаріших платформ для обміну фотографіями, яка підтримує велику, різноманітну спільноту. Вона дозволяє користувачам зберігати, організовувати та ділитися фотографіями в альбомах і групах [2].

Переваги:

- велике сховище для фотографій (1000 для безкоштовних акаунтів);
- розширені можливості тегування та організації альбомів;
- широко визнана платформа.

Недоліки:

- застарілий інтерфейс і повільні оновлення в останні роки;
- багато функцій приховано за Pro (платною) підпискою;
- слабкі інструменти захисту авторських прав.

1.3.4 Сервіс Behance

Behance - це платформа від Adobe, орієнтована на демонстрацію творчих проєктів з усіх галузей дизайну, включаючи фотографію [3]. Вона широко використовується професіоналами для створення онлайн-портфоліо.

Переваги:

- тісна інтеграція з Adobe Creative Cloud;

- чудово підходить для створення портфоліо проєктів із кількох зображень;
- високий рівень видимості контенту;
- професійні інструменти для роботи з мережею.

Недоліки:

- не є специфічно фотографічними - змішані з графічним дизайном, ілюстрацією тощо;
- немає внутрішньо платформного ліцензування або інструментів продажу;
- менший акцент на взаємодію зі спільнотою (коментарі/лайки обмежені);
- немає надійного захисту від завантаження зображень.

1.3.5 Таблиця порівняння існуючих онлайн фото-сервісів для фотографів

В таблиці 1.1 наведено порівняння і оцінка критерії існуючих онлайн фото-сервісів.

Таблиця 1.1 - Дозволяє порівняти і оцінити критерії існуючих онлайн фото-сервісів

Критерій	500px	Flickr	Behance
1	2	3	4
Інтерфейс та зручність	Сучасний, мінімалістичний, з акцентом на фото	Застарілий інтерфейс, але функціональний	Стильний та професійний, інтегрований з Adobe
Презентація портфоліо	Якісна подача фото, чистий макет	Альбомна структура, гнучке сортування	Проєктна структура, зручна для візуального сторітелінгу

Продовження таблиці 1.1

1	2	3	4
Соціальна взаємодія	Вподобання, коментарі, підписка	Вподобання, коментарі, групи, обговорення	Вподобання, обмежена взаємодія
Активність спільноти	Добірки редакції, конкурси користувачів	Великі відкриті групи та тематичні спільноти	Відібрані проєкти, але менше взаємодії між користувачами
Захист авторських прав	Водяні знаки, ліцензування, обмежений контроль	Слабкий захист, можливість заборонити завантаження	Відсутність захисту, зображення легко копіювати
Модерація та адміністрування	Базові інструменти модерації коментарів та профілів	Групова модерація, система скарг	Мінімальні можливості, орієнтація на проєкти
Доступність та вартість	Безкоштовно з обмеженнями, є Pro-підписка	Безкоштовно до 1000 фото, потрібна Pro-версія для більшого	Безкоштовно, потрібен обліковий запис Adobe
Система рекомендацій	Популярне, добірка редакції	Сторінка «Explore», промоція у групах	Вибір редакції Adobe, ручна модерація
Можливість монетизації	Продаж фотографій через маркетплейс	Відсутня вбудована монетизація	Відсутня монетизація, лише для демонстрації портфоліо
Цільова аудиторія	Професійні та аматорські фотографи	Широке коло користувачів, ентузіасти	Креативні фахівці різних напрямків (не лише фотографи)

1.3.6 Висновки

Аналіз існуючих платформ, таких як 500px, Flickr та Behance, показує, що хоча кожна з них надає цінні інструменти для фотографів, жодна з них не задовольняє всі сучасні потреби цієї групи користувачів. Такі питання, як захист авторських прав, бездоганна презентація портфоліо, активне залучення спільноти та гнучка монетизація, залишаються або частково охопленими, або повністю відсутніми. Це підкреслює потребу в новому, комплексному інтернет-сервісі, який поєднає сильні сторони існуючих платформ і водночас усуне їхні недоліки.

1.4 Постановка завдання

Метою цього проєкту є розробка та впровадження сучасної, зручної для користувачів соціальної інтернет-платформи для фотографів. Платформа повинна надавати інструменти для демонстрації та захисту фотографічного контенту, полегшувати взаємодію між користувачами та підтримувати управління особистими та спільнотними колекціями.

1.4.1 Вимоги до інтерфейсу

Інтерфейс користувача повинен бути чистим, інтуїтивно зрозумілим і візуально привабливим. Платформа повинна забезпечувати легку навігацію між розділами (галерея, колекції, профіль тощо) та чіткий зворотній зв'язок з користувачем.

Основні вимоги до інтерфейсу:

- адаптивний дизайн до різних розмірів екрану;
- перегляд фотографій у вигляді сітки для сторінок галереї;
- модальний або повноекранний режим для попереднього перегляду окремих фотографій;

- секції коментарів під кожною фотографією;
- зрозуміла та доступна панель навігації.

1.4.2 Вимоги до безпеки

Безпека є критично важливим компонентом цієї платформи, особливо з огляду на необхідність захисту інтелектуальної власності та персональних даних користувачів.

Основні вимоги до безпеки:

- безпечна автентифікація користувачів через JWT-токен;
- шифрування та безпечне зберігання паролів;
- запобігання несанкціонованому завантаженню фотографій;
- контроль доступу на основі ролей (наприклад, адміністратор проти звичайного користувача);
- захист від поширених вебзагроз (XSS, CSRF, SQL-ін'єкції).

1.4.3 Основні вимоги до системи

Функціональні вимоги:

- реєстрація та аутентифікація користувачів;
- завантаження та публікація фотоматеріалів;
- перегляд робіт інших користувачів;
- створення колекцій з фотографій інших користувачів;
- коментування фотопублікацій;
- модерація коментарів адміністратором;
- відображення популярних фотографій;
- адмін-панель для управління контентом;
- інструменти захисту авторських прав (заборона копіювання).

Нефункціональні вимоги:

- висока доступність та швидкість реагування системи;

- крос-платформна сумісність (десктопні та мобільні браузери);
- масштабована архітектура для підтримки зростаючої кількості користувачів;
- зрозумілий та інтуїтивно зрозумілий UI/UX;
- підтримка сучасних форматів зображень (JPEG, PNG, JPG).

1.5 Висновок розділу

Аналіз предметної області дозволив виявити ключові потреби та очікування сучасних фотографів щодо онлайн-платформ. Зокрема, було виявлено, що попри наявність таких популярних сервісів, як 500px, Flickr та Behance, жоден з них не надає комплексного рішення для демонстрації, захисту та просування фотографічних матеріалів в рамках єдиної екосистеми. Деякі платформи мають сильну візуальну складову, інші - активну спільноту або хорошу систему рекомендацій, але вони часто нехтують такими важливими аспектами, як безпека, легка модерація або захист авторських прав.

Актуальність створення нового соціального онлайн-сервісу для фотографів зумовлена потребою в інтегрованому рішенні, що поєднує якісне портфоліо, механізми соціальної взаємодії, надійний захист контенту та простоту використання. Такий сервіс не лише задовольнить потреби аматорів та професіоналів, але й допоможе сформувати творчу спільноту, де цінується оригінальність, зворотній зв'язок та безпека.

Результати аналізу лягли в основу постановки задачі та формулювання функціональних і нефункціональних вимог до системи, що розробляється в рамках даного дипломного проєкту.

2 АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ

Для того, щоб розроблена платформа відповідала сучасним стандартам продуктивності, юзабіліті та масштабованості, було обрано збалансований та перевірений стек технологій як для фронтенд, так і для бекенд-компонентів. У цьому розділі наведено обґрунтування вибору мов програмування, фреймворків та середовищ розробки.

2.1 Вибір мови програмування

JavaScript (JS) - це легка, інтерпретована мова програмування, яка вирізняється своєю гнучкістю та широкими можливостями. Вона має функції вищого порядку та підтримує декілька парадигм програмування, включаючи об'єктно-орієнтований, імперативний та декларативний стилі. Як однопотокова мова, що базується на прототипах, JavaScript особливо добре підходить для створення динамічних та інтерактивних вебдодатків [4].

Для розробки цієї системи було обрано JavaScript як основну мову програмування. Це рішення було зумовлене її універсальністю та ефективністю, що дозволяє уніфікувати розробку як інтерфейсу, так і бекенду. Завдяки використанню JavaScript у всьому технологічному стеку, процес розробки стає більш раціональним, зводячи до мінімуму перемикання контексту та забезпечуючи безперебійну комунікацію між клієнтськими та серверними компонентами [4]-[5].

Крім того, JavaScript користується широким розповсюдженням і підтримується великою екосистемою інструментів, фреймворків і бібліотек, таких як Vue.js, Node.js, Express та багато інших. Величезна популярність мови в індустрії забезпечення забезпечує постійні вдосконалення, потужну підтримку спільноти та велику кількість навчальних матеріалів.

2.2 Огляд особливостей обраних платформ і бібліотек

Для фронтенду були обрані наступні технології:

- Vue3: сучасний, прогресивний JavaScript-фреймворк, що підтримує реактивне зв'язування даних та архітектуру на основі компонентів. Новий API Composition API, представлений у Vue 3, покращує організацію коду та можливість повторного використання [6];
- Vuetify 3: популярна бібліотека інтерфейсів, заснована на Material Design від Google. Вона надає широкий спектр настроюваних і адаптивних компонентів, які прискорюють розробку інтерфейсу та забезпечують візуальну узгодженість [7];
- Pinia: легка, інтуїтивно зрозуміла бібліотека керування станами, розроблена спеціально для Vue 3. Порівняно з Vuex, вона більш модульна, легша у тестуванні та краще узгоджена з Composition API [8];
- Axios: широко використовуваний HTTP-клієнт для запитів до API. Його синтаксис, заснований на обіцянках, покращує читабельність коду та забезпечує надійну обробку помилок [9];
- Day.js: компактна бібліотека для розбору, перевірки, маніпулювання та форматування дат. Використовується для обробки міток часу у завантаженому користувачем вмісті [10];
- Sass.js: препроцесор CSS, який покращує зручність обслуговування та гнучкість таблиць стилів. Завдяки вкладеності, змінним та міксинам Sass дозволяє писати чисті та масштабовані стилі інтерфейсу [11].

Стек бекенда включає в себе:

- Node.js: середовище виконання JavaScript, побудоване на движку Chrome V8, оптимізоване для створення масштабованих мережевих додатків. Підтримує асинхронний, неблокуючий код, який ідеально підходить для обробки декількох клієнтських запитів [12];

- Express.js: легкий вебфреймворк для Node.js, який спрощує маршрутизацію, управління проміжним програмним забезпеченням та створення REST API [13];
- Sequelize: ORM, який полегшує взаємодію між додатком і реляційною базою даних. Дозволяє визначати моделі в JavaScript і відображати їх безпосередньо в таблиці PostgreSQL [14];
- PostgreSQL: потужна реляційна база даних з відкритим вихідним кодом, відома своєю надійністю, продуктивністю та підтримкою розширених типів даних і запитів [15].

2.3 Вибір середовища розробки

Visual Studio Code поєднує в собі простоту редактора вихідного коду з потужними інструментами розробника.

В основі Visual Studio Code лежить блискавичний редактор вихідного коду, який ідеально підходить для щоденного використання. Завдяки підтримці сотень мов VS Code допомагає миттєво підвищити продуктивність завдяки підсвічуванню синтаксису, узгодженню дужок, автоматичним відступам, виділенню блоків, фрагментів тощо [4]. Інтуїтивно зрозумілі комбінації клавіш, легке налаштування та створені спільнотою відображення клавіатурних скорочень дозволять вам легко орієнтуватися у вашому коді.

Postman відіграв важливу роль у процесі розробки та тестування бекенду. Це потужний API-клієнт, який спрощує створення, тестування та документування RESTful API. Під час розробки Postman використовувався для імітації HTTP-запитів (GET, POST, PUT, DELETE) до бекенд-сервера та перевірки коректності відповідей [16]. Це допомогло виявити проблеми з автентифікацією, перевіркою даних та бізнес-логікою на ранній стадії процесу. Колекції Postman також дозволили повторно використовувати тестові кейси.

GitHub у поєднанні з Git використовувався для контролю версій та спільної розробки. GitHub надавав централізоване сховище коду, де зберігалася, версіонувалася та документувалася вся база коду. Завдяки використанню гілок, комітів та запитів на витягування, зміни можна було відстежувати, переглядати та об'єднувати в організований спосіб [17]. Це гарантувало, що розробка залишалася стабільною, а функції інтегрувалися лише після ретельної перевірки.

2.4 Схема розробки

У проєкті була використана клієнт-серверна архітектура, яка забезпечує взаємодію клієнта з сервером через зручний інтерфейс вебзастосунку. Клієнт має змогу через зручний інтерфейс взаємодіяти з сервером, посилати запити через HTTP у форматі JSON або FormData у випадку відправки файлів. Сервер у відповідь віддає результат операції в JSON форматі, завдяки реактивності фреймворку Vue 3 застосунок реагує на оновлення і відображає користувачу нову інформацію.

Приклад роботи клієнт-серверної архітектури на рис. 2.1 [18].

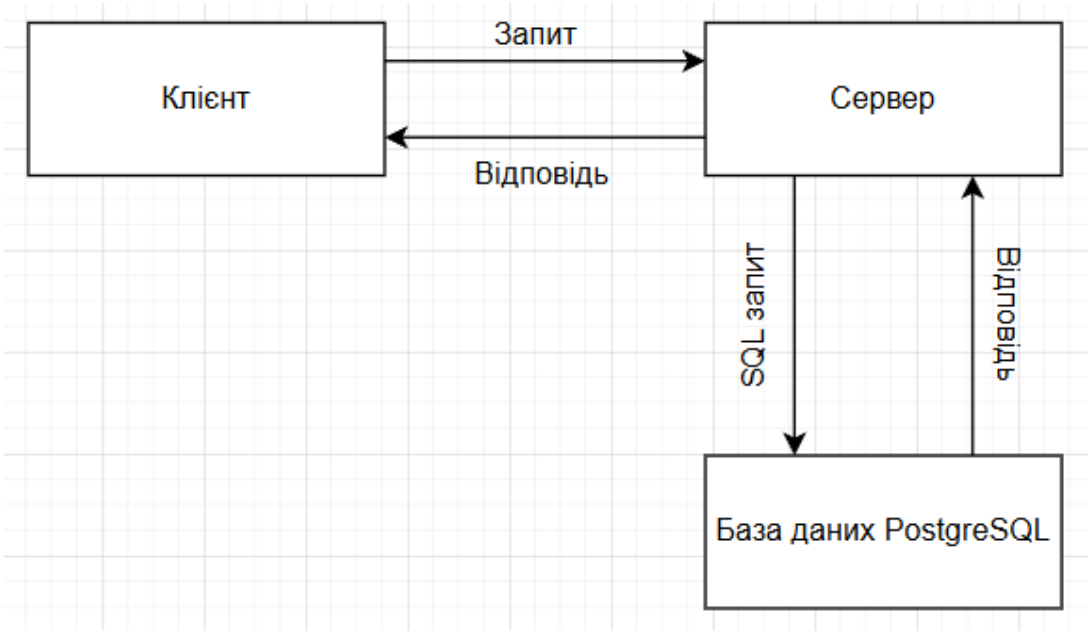


Рисунок 2.1 – Схема роботи клієнт-серверної архітектури [18]

3 РОЗРОБКА ТА РЕАЛІЗАЦІЯ СИСТЕМИ

Цей розділ містить детальний опис того, як була реалізована система, охоплюючи кожен основний компонент і функціональність розробленого додатку. На етапі реалізації теоретичні та проєктні засади перетворюються на повнофункціональну вебплатформу. Він включає в себе розробку як фронтенд, так і бекенд логіки, а також інтеграцію бази даних і ключових користувацьких функцій.

Система була побудована з використанням сучасного технологічного стеку, включаючи Vue 3 і Vuetify для фронтенду, і Node.js з Express і Sequelize для бекенду. В якості системи реляційних баз даних було обрано PostgreSQL. Дизайн відповідає модульній та масштабованій архітектурі, що забезпечує зручність обслуговування та потенційні майбутні вдосконалення.

3.1 Загальна архітектура системи

Додаток базується на клієнт-серверній архітектурі. Фронтенд реалізований як односторінковий додаток (SPA) з використанням Vue.js третьої версії та Vuetify 3, що забезпечує плавний та динамічний користувацький досвід без повних перезавантажень сторінки. Фронтенд взаємодіє з бекендом через кінцеві точки RESTful API.

Бекенд побудований на Node.js та Express, що забезпечує ефективне та легке серверне середовище для обробки HTTP-запитів, обробки бізнес-логіки та управління автентифікацією користувачів. Sequelize ORM використовується для взаємодії з базою даних PostgreSQL, яка зберігає постійні дані, такі як профілі користувачів, фотографії, колекції та коментарі.

Ключові компоненти архітектури включають:

- фронтенд (Vue.js та Vuetify): відповідає за візуалізацію та взаємодію з користувацьким інтерфейсом;

- бекенд (Node.js та Express): керує маршрутами API, автентифікацією, перевіркою даних та бізнес-логікою;
- база даних (PostgreSQL та Sequelize): зберігає структуровані дані в реляційних таблицях зі зв'язками (наприклад, один-до-багатьох між користувачами та фотографіями);
- аутентифікація (JWT): Система входу користувачів на основі токенів для захисту доступу до приватних ресурсів;
- завантаження та зберігання зображень: ображення завантажуються через API і зберігаються в захищеному каталозі, а посилання на них зберігаються в базі даних;
- контроль версій (gitHub): підтримує цілісність вихідного коду та підтримує спільну розробку;
- тестування API (postman): широко використовується для ручного тестування кінцевих точок бекенда.

Ця архітектура забезпечує чіткий розподіл завдань, масштабованість і гнучкість для розширення функцій. Кожен компонент відповідає за свою власну задачу, що спрощує налагодження, тестування та майбутню розробку.

3.2 Розробка бази даних

База даних є фундаментальним компонентом програми, що відповідає за організацію, зберігання та управління всіма постійними даними, такими як користувачі, фотографії, теги, коментарі та колекції. Реляційна база даних (PostgreSQL) була обрана через її надійність, узгодженість та підтримку складних запитів. Sequelize, використовувався для визначення моделей та їхніх зв'язків у декларативний спосіб, що спрощує взаємодію з базою даних та забезпечує цілісність даних. Ця надійна комбінація забезпечує безперебійну взаємодію між додатком і його рівнем даних, закладаючи міцний фундамент для майбутнього розширення функцій і обслуговування системи.

3.2.1 Схема бази даних

Схема бази даних була розроблена таким чином, щоб відображати реальні взаємозв'язки, такі як:

- користувач може завантажити багато фотографій і створити кілька колекцій;
- фотографії можуть бути позначені тегами і згруповані в колекції;
- користувачі можуть залишати коментарі до фотографій і вибирати бажані теги.

Схема бази даних зображена на рисунку 3.1.

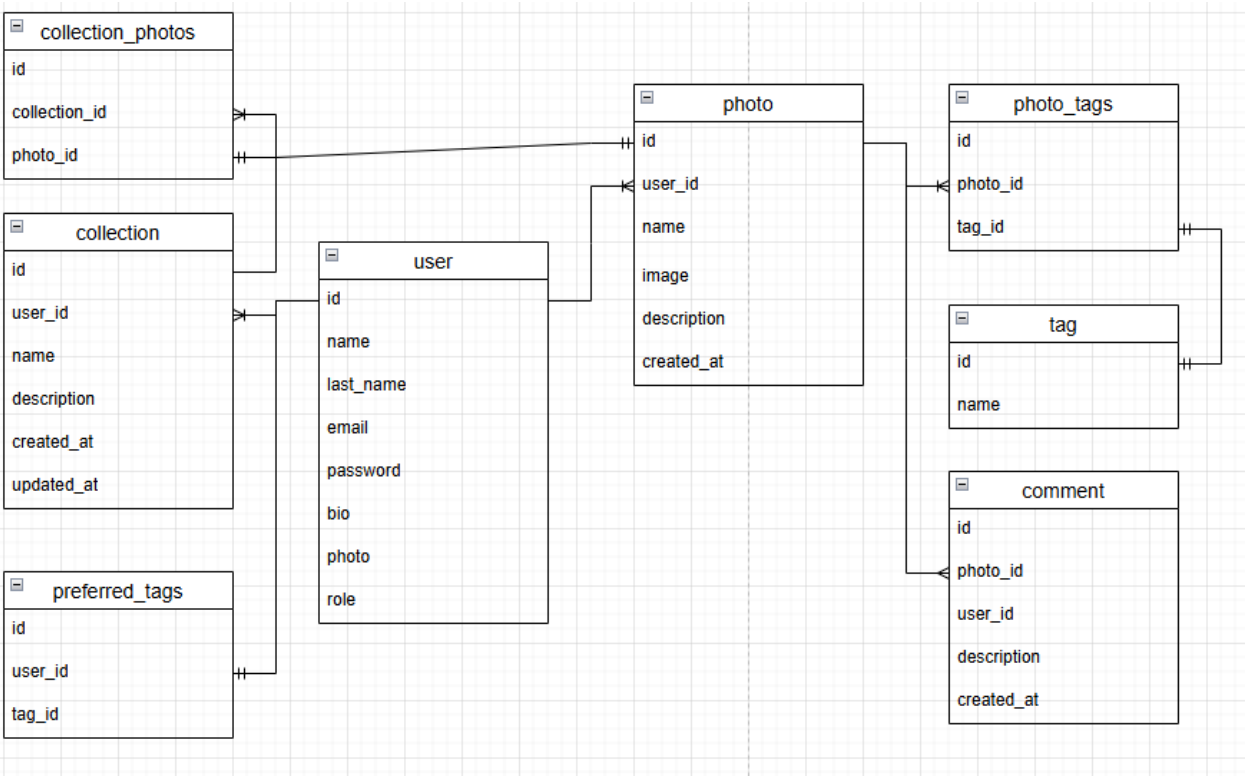


Рисунок 3.1 – Схема бази даних онлайн сервісу для фотографів

3.2.2 Опис таблиць

Опис таблиць:

- таблиця користувача (user): зберігає інформацію про зареєстрованих користувачів, включаючи дані автентифікації та особисті дані;

- таблиця фотографій (photo): зберігає інформацію про завантажену фотографію, включаючи унікальне ім’я фотографії та шлях до директорії де зберігається фотографія;
- таблиця тегів (tag): зберігання створених тегів, які використовують користувачі для пошуку та фільтрації фотографій;
- таблиця тегів фотографій (photo_tags): використовується для встановлення зв'язку «багато-до-багатьох» між таблицями «фото» і «тег». Вона дозволяє одній фотографії мати кілька тегів, а одному тегу - бути пов'язаним з багатьма фотографіями. Це важливо для організації та категоризації фотографій для ефективного пошуку та фільтрації;
- таблиця колекцій (collection): зберігається інформація про створені користувачем колекції фотографій. Кожна колекція прив'язана до одного користувача і може містити різні фотографії, згруповані за певною темою або уподобаннями;
- таблиця фотографій в колекціях (collection_photos): допоміжна таблиця, яка створює зв'язок «багато-до-багатьох» між колекцією та фотографією. Вона дозволяє одній фотографії бути частиною кількох колекцій, а одній колекції містити кілька фотографій. Така гнучкість важлива для динамічної організації та повторного використання контенту в різних кураторських наборах;
- таблиця коментарів (comment): зберігаються коментарі користувачів, пов'язані з конкретними фотографіями. Кожен коментар пов'язаний як з користувачем (автором), так і з фотографією. Це підтримує соціальну взаємодію, зворотній зв'язок і залучення на платформі;
- таблиця бажаних тегів користувача (user_preffered_tags): пов'язує користувачів з їхніми улюбленими тегами. Це дозволяє платформі зберігати та аналізувати інтереси користувачів на основі їхніх уподобань щодо тегів.

В таблиці 3.1 наведено поля таблиці user.

В таблиці 3.2 наведено поля таблиці photo.

В таблиці 3.3 наведено поля таблиці photo_tags.

Таблиця 3.1 – Опис таблиці user

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
name	STRING	Ні	Ім’я користувача
last_name	STRING	Ні	Прізвище користувача
email	STRING	Так	Унікальна електронна пошта
password	STRING	Так	Пароль
bio	STRING	Ні	Біографія
avatar	STRING	Ні	Унікальне ім’я картинки
avatar_folder	STRING	Ні	Шлях до аватара
role	STRING	Ні	Роль (за замовчуванням USER)

Таблиця 3.2 – Опис таблиці photo

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
user_id	INTEGER	Так	Зовнішній ключ користувача
name	STRING	Ні	Назва фотографії
description	STRING	Ні	Опис фотографії
image	STRING	Так	Шлях до зображення
image_folder	STRING	Ні	Шлях до папки
created_at	DATE	Ні	Дата створення

Таблиця 3.3 – Опис таблиці photo_tags

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
photo_id	INTEGER	Так	Зовнішній ключ фотографії
tag_id	INTEGER	Так	Зовнішній ключ тега

В таблиці 3.4 наведено поля таблиці tag.

Таблиця 3.4 – Опис таблиці tag

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
name	STRING	Ні	Назва тегу

В таблиці 3.5 наведено поля таблиці collection.

Таблиця 3.5 – Опис таблиці collection

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
user_id	INTEGER	Так	Зовнішній ключ користувача
name	STRING	Ні	Назва колекції
description	STRING	Ні	Опис колекції

В таблиці 3.6 наведено поля таблиці collection_photos.

Таблиця 3.6 – Опис таблиці collection_photos

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
collection_id	INTEGER	Так	Зовнішній ключ колекції
photo_id	INTEGER	Так	Посилання на фотографію

В таблиці 3.7 наведено поля таблиці comment.

В таблиці 3.8 наведено поля таблиці user_preffered_tags.

Таблиця 3.7 – Опис таблиці comment

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
user_id	INTEGER	Так	Зовнішній ключ користувача
photo_id	INTEGER	Так	Зовнішній ключ фотографії
description	STRING	Ні	Текст коментаря

Таблиця 3.8 – Опис таблиці user_preffered_tags

Ключ	Тип даних	Обов’язкове поле	Опис
id	INTEGER	Так	Первинний ключ
user_id	INTEGER	Так	Зовнішній ключ користувача
tag_id	INTEGER	Так	Зовнішній ключ тега

Розроблена структура бази даних забезпечує міцну основу для функціональності додатку. Кожна таблиця цілеспрямовано створена для підтримки основних функцій, таких як управління користувачами, публікація фотографій, колекцій, тегування та коментування. Ретельно встановлені зв'язки між сутностями, забезпечують узгодженість даних, масштабованість та ефективні запити.

3.3 Реалізація реєстрації та авторизації користувачів

Реєстрація та автентифікація користувачів є критично важливими компонентами безпеки та персоналізації додатку. Під час реєстрації користувачі надають свою електронну пошту, пароль та особисту інформацію. Паролі надійно хешуються перед збереженням у базі даних, щоб запобігти несанкціонованому доступу.

Аутентифікація здійснюється через форму входу, де користувач надає свої облікові дані. Після успішного входу видається JWT-токен для автентифікації майбутніх запитів. Це гарантує, що користувачі можуть

отримати доступ до свого особистого контенту, безпечно взаємодіяти з платформою та запобігає несанкціонованому доступу до захищених маршрутів, наприклад адмін панелі.

Також реалізовано контроль доступу на основі ролей. За замовчуванням кожному користувачеві призначається роль користувача, тоді як адміністратори можуть мати додаткові привілеї, такі як модерація контента та фотографіями.

Розроблено метод при котрому користувач залишається авторизованим протягом доби, якщо користувач не відкриває додаток більше доби, авторизація очищається, в іншому випадку сесія користувача продовжується на добу після відкриття додатку.

3.4 Публікація та збереження фотоматеріалів

Основна функціональність програми полягає у можливості публікувати та зберігати фотографії. Користувачі можуть завантажувати зображення разом з описовими метаданими, такими як назва та опис. Кожна фотографія прив'язується до конкретного користувача і зберігається в організованій структурі папок на сервері.

До кожної фотографії можна додати теги для категоризації, що дозволяє користувачам більш ефективно фільтрувати та шукати контент. Крім того, користувачі можуть групувати свої фотографії в колекції, забезпечуючи спосіб організації та обміну пов'язаними медіа.

База даних і файлова система працюють разом, щоб забезпечити швидкий доступ до медіа, зберігаючи цілісність метаданих. Платформа також підтримує зв'язки «багато-до-багатьох» між фотографіями та колекціями, що дозволяє фотографіям з'являтися в декількох колекціях без дублювання.

3.5 Валідація та обробка помилок

Для забезпечення цілісності даних і надійної роботи користувачів додаток включає в себе надійні механізми перевірки та обробки помилок. Під час будь-якого процесу введення, наприклад, реєстрації, завантаження фотографій або коментарів дані перевіряються на відповідність певним правилам (наприклад, формат електронної пошти, тип зображення, наявність непорожніх полів).

Якщо виникає помилка, користувачеві повертаються змістовні повідомлення про помилку. Ці повідомлення допомагають користувачеві виправити введені дані. На стороні сервера винятки перехоплюються і реєструються належним чином, щоб запобігти падінню програми і допомогти в налагодженні.

Крім того, відповіді API стандартизовані, що забезпечує послідовний зворотній зв'язок у разі виникнення проблем на стороні клієнта або сервера. Це підвищує загальну надійність програми і забезпечує кращій досвід користування вебзастосунком для користувача.

3.6 Висновок до розділу

Успішна реалізація системи відображає продуманий і структурований підхід до розробки повного стеку. Від архітектурного дизайну до інтеграції фронтенд і бекенд-компонентів, кожен елемент розроблявся з акцентом на зручність використання, продуктивність і масштабованість.

Архітектура була розроблена таким чином, щоб чітко розділити проблеми між клієнтською та серверною сторонами, що дозволило покращити ремонтопридатність та розширюваність. Vue 3 з Vuetify 3 забезпечили сучасний і чуйний користувальницький інтерфейс, а Pinia забезпечила чисте і централізоване управління станом. На бекенді Node.js з Express та Sequelize

дозволили створити RESTful API, який безперешкодно взаємодіє з базою даних PostgreSQL, пропонуючи надійний та ефективний рівень даних.

Використовуючи належним чином нормалізовані таблиці та чітко визначені зв'язки, система підтримує такі функції, як профілі користувачів, публікація фотографій, колекції, тегування та коментування. Ці компоненти є критично важливими для забезпечення багатого користувацького досвіду та уможливлюють майбутні вдосконалення без необхідності перебудови всієї структури даних.

Механізми реєстрації та автентифікації гарантують безпеку користувачів і дозволяють керувати доступом на основі ролей, забезпечуючи відповідний доступ для різних типів користувачів. Логіка публікації та керування фотографіями, а також можливість їх категоризації за допомогою тегів і колекцій, підтримує пошук і організацію контенту. Включення валідації та обробки помилок підвищує стабільність системи та покращує взаємодію з користувачами, забезпечуючи постійну перевірку даних та надання інформативного зворотного зв'язку у разі виникнення проблем.

Загалом, реалізація не лише відповідає функціональним вимогам, але й закладає міцний фундамент для майбутнього розвитку. Система розроблена таким чином, щоб бути масштабованою, підтримуваною та зручною для користувачів, забезпечуючи сучасну цифрову платформу, здатну розвиватися відповідно до потреб її користувачів та зацікавлених сторін.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

Мета цього розділу - продемонструвати, що додаток відповідає своїм початковим цілям, надійно працює при очікуваному використанні і готовий до подальшого масштабування або розгортання.

4.1 Призначення й умови застосування програми

Додаток розроблений як платформа для обміну, організації та пошуку фотографічного контенту. Він дозволяє користувачам створювати особисті профілі, завантажувати та публікувати свої фотографії, позначати їх для зручності пошуку, взаємодіяти з іншими користувачами за допомогою коментарів та організовувати зображення в колекції.

Платформа дозволяє користувачам реєструватися та створювати персоналізовані профілі, завантажувати та публікувати свої фотографії, організовувати їх у колекції, додавати описові теги та взаємодіяти за допомогою коментарів. Вона має на меті забезпечити впорядкований і зручний досвід як для фотографів-аматорів, так і для професійних фотографів.

Додаток надає наступні можливості:

- реєстрація та автентифікація користувачів за допомогою безпечних облікових даних;
- завантаження та зберігання фотоматеріалів у впорядкованих папках;
- створення тематичних фотоколекцій для категоризації та зручного доступу;
- додавання та керування тегами для полегшення пошуку та фільтрації фотографій;
- залишати та переглядати коментарі, заохочувати взаємодію з користувачами та зворотній зв'язок;

- керування профілями користувачів, включно з фото профілю та особистими біографіями.

4.2 Вимоги до інтерфейсу програми

Інтерфейс користувача (UI) розроблений таким чином, щоб бути інтуїтивно зрозумілим, чуйним і візуально доступним, що робить додаток простим у використанні як для технічних, так і для нетехнічних користувачів. Він зосереджений на ясності, зручності та мінімалізмі, щоб забезпечити безперебійну роботу користувача.

а) загальні вимоги до інтерфейсу:

1) мінімалістичний дизайн, який відповідає сучасним тенденціям UI/UX;

2) інтуїтивна зрозуміла структура сторінок;

3) єдиний стиль всього вебзастосунку;

4) швидкодія інтерфейсу вебзастосунку;

б) головна сторінка:

1) показ останніх завантажених фотографій і популярних тегів;

2) великі картки з фотографіями, при наведені показувати ім’я автора та опис;

3) доступ до пошуку, профілю або авторизації користувача;

в) вікно з фотографією:

1) відображення зображення у гарній якості;

2) показ автора, тегів, опису фотографії;

3) кнопка для додання фотографії;

4) можливість перейти до наступної фотографії;

5) можливість переглядати та залишати коментарі;

г) акаунт користувача:

1) перегляд фотографій користувача;

2) можливість переглянути колекції користувача;

3) для зареєстрованих користувачів, функціонал для додання нових фотографія та колекцій;

4) можливість оновлювати особисту інформацію;

д) адміністративна панель:

1) зрозумілий дизайн для управління акаунтами користувачів, коментарями та фотографіями;

2) контроль над ролями користувачів;

е) сторінка авторизації та реєстрації:

1) поля для введення даних;

2) перевірка на правильність ведених даних.

4.3 Умови застосування програми

Додаток був розроблений для забезпечення стабільної роботи, масштабованості та безпеки. Він може бути розгорнутий як на серверах розробки, так і на виробничих серверах з мінімальними зусиллями по налаштуванню, що робить його адаптованим до різних середовищ - від локального тестування до повного розгортання в хмарній інфраструктурі.

Наведені нижче рекомендації щодо конфігурації призначені для забезпечення оптимальної продуктивності та безпечної роботи програми як для кінцевих користувачів, так і для адміністраторів.

Вимоги до апаратного забезпечення, на якому буде працювати серверна частина онлайн сервісу для фотографів:

- процесор: 4 ядра 2.0 ГГц;
- оперативна пам’ять: мінімум 4 Гб;
- сховище: SSD з принаймні 50 Гб вільного місця;
- node.js версія: мінімум 18 версія;
- версія PostgreSQL: 13 версія і вище.

Вимоги до апаратного забезпечення, на якому буде працювати клієнтська частина онлайн сервісу для фотографів:

- процесор: двоядерний 1,8 ГГц або вище;
- оперативна пам’ять: мінімум 4 Гб;
- браузер: остання версія Google Chrome, Mozilla Firefox або Opera;
- стабільне інтернет-з’єднання.

4.4 Тестування програми

Для забезпечення надійності, функціональності та крос-платформної сумісності розробленого додатку було проведено комплексний етап тестування. Основною метою цього процесу була перевірка правильності роботи всіх функцій за різних умов, а також виявлення будь-яких помилок або проблем з юзабіліті перед розгортанням.

Додаток було ретельно протестовано в останніх версіях трьох основних веббраузерів, щоб забезпечити узгодженість поведінки та зовнішнього вигляду:

- Google Chrome (версія 137.0.7);
- Mozilla Firefox (версія 138.0.1);
- Opera (версія 119.0.5).

Детальний опис тестових сценаріїв наведено в таблиці 4.1.

Таблиця 4.1 – Тестові сценарії

Тестовий сценарій	Google Chrome	Mozilla Firefox	Opera
1	2	3	4
Реєстрація користувача з правильним введенням даних	Пройшов	Пройшов	Пройшов
Реєстрація користувача з невірним введенням даних	Пройшов	Пройшов	Пройшов
Вхід користувача з правильними обліковими даними	Пройшов	Пройшов	Пройшов

Продовження таблиці 4.1

1	2	3	4
Вхід користувача з неправильними обліковими даними	Пройшов	Пройшов	Пройшов
Хешування та перевірка паролів	Пройшов	Пройшов	Пройшов
Створення нової колекції	Пройшов	Пройшов	Пройшов
Додавання фотографії в колекцію	Пройшов	Пройшов	Пройшов
Додавання нового коментаря	Пройшов	Пройшов	Пройшов
Редагування даних користувача	Пройшов	Пройшов	Пройшов
Доступ до панелі адміністратора	Пройшов	Пройшов	Пройшов
Керування адміном користувачами та колекціями	Пройшов	Пройшов	Пройшов
Пошук за тегами та назвою	Пройшов	Пройшов	Пройшов
Захищені сторінки від неавторизованих користувачів	Пройшов	Пройшов	Пройшов
Заборона на завантаження фотографій	Пройшов	Пройшов	Пройшов

Згідно з описаними вище вимогами, наводжу приклад користування користувача з онлайн сервісом для фотографів.

На рисунку 4.1 зображено приклад помилок при спробі ввести неправильні дані для реєстрації.

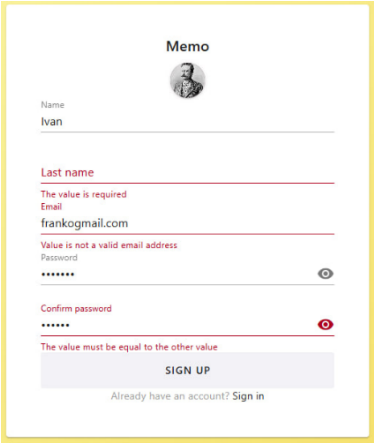


Рисунок 4.1 – Приклад помилок при реєстрації

На рисунку 4.2 зображена сторінка авторизації користувача.

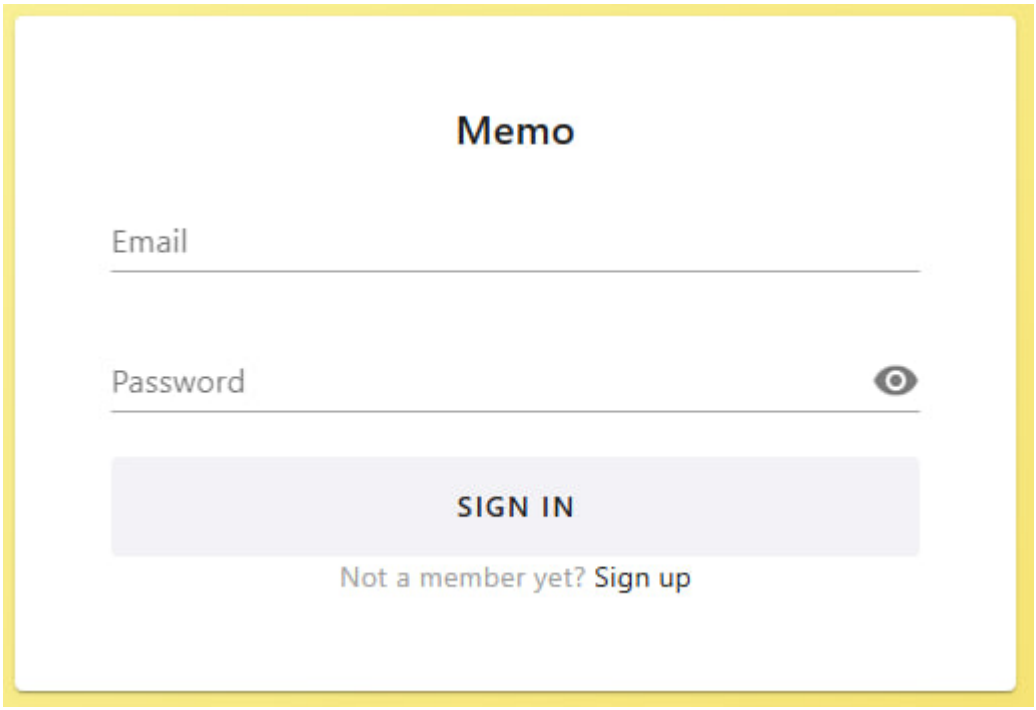


Рисунок 4.2 – Сторінка авторизації користувача

На рисунку 4.3 зображена головна сторінка вебзастосунку.

Рисунок 4.3 – Головна сторінка вебзастосунку

На рисунку 4.4 зображено поле пошуку по ключовим словам, тегам, перелік фотографій.

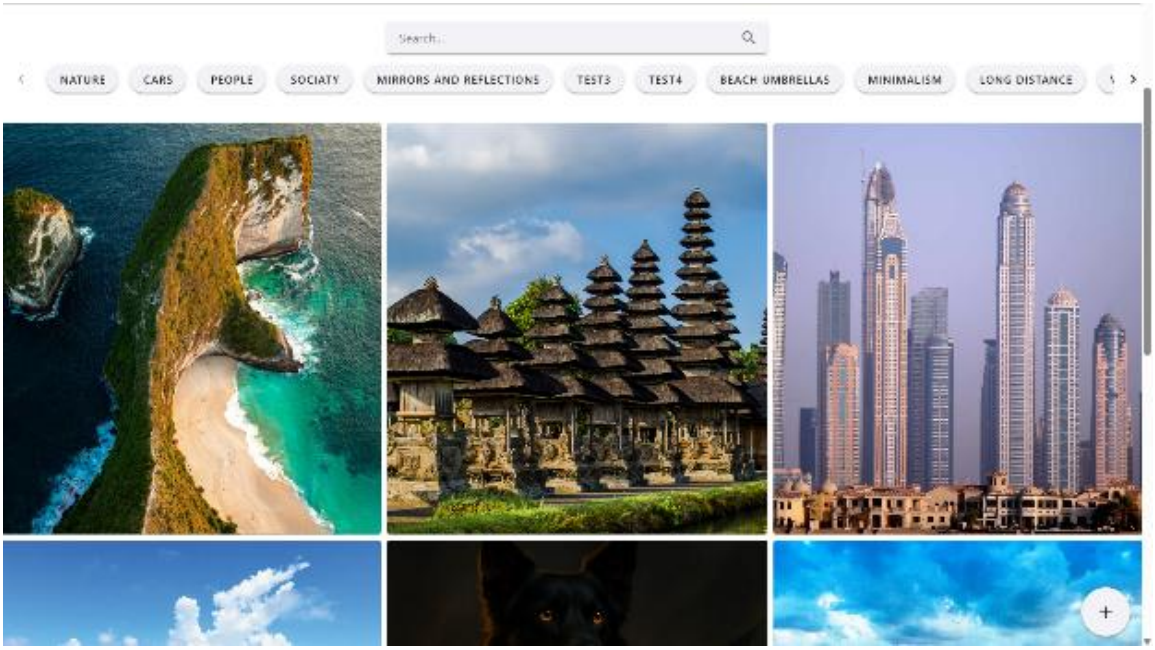


Рисунок 4.4 – Інструменті фільтрації та перелік фотографій

На рисунку 4.5 зображено вікно для створення нових фотографій, яке відкривається при натисканні на кнопку в ніжній правій частини екрану.

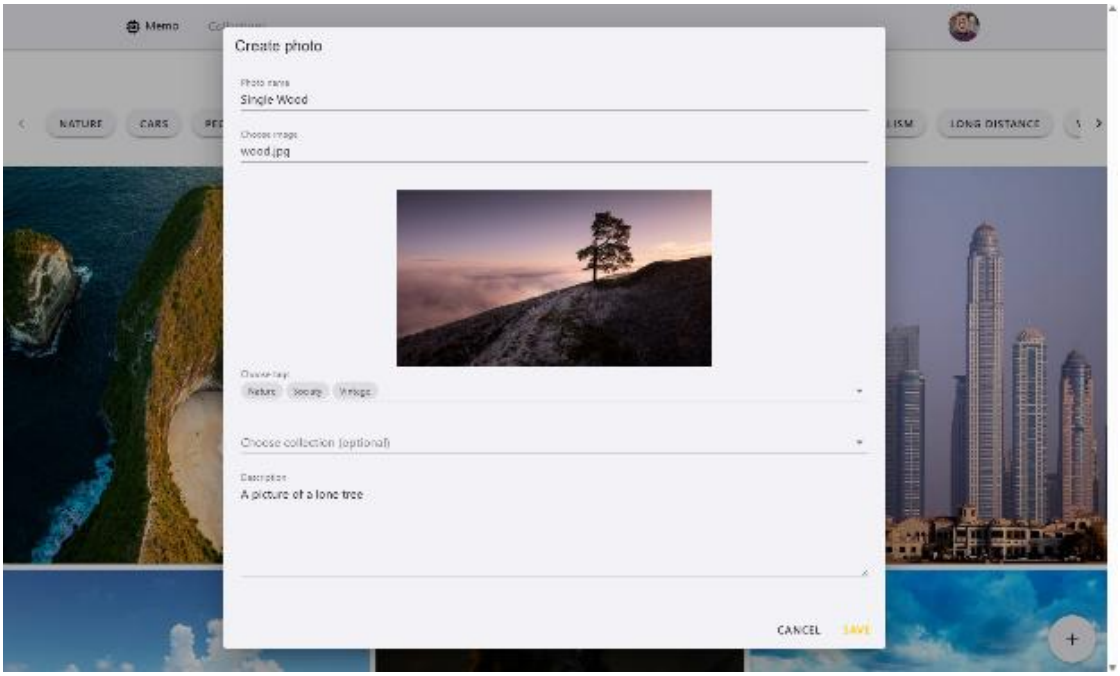


Рисунок 4.5 – Вікно для створення нових фотографій

На рисунку 4.6 зображено меню користувача.

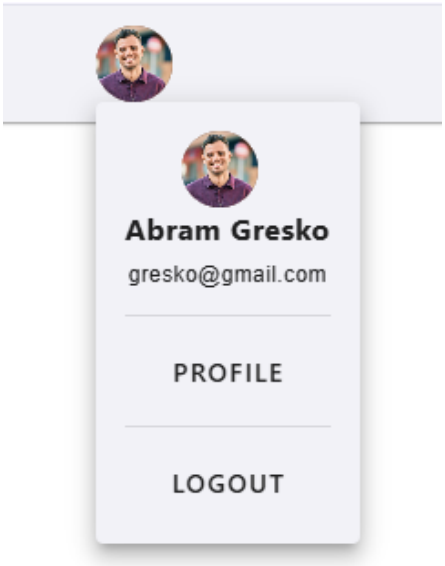


Рисунок 4.6 – Меню користувача

На рисунку 4.7 зображена сторінка зі всіма існуючими колекціями користувачів.

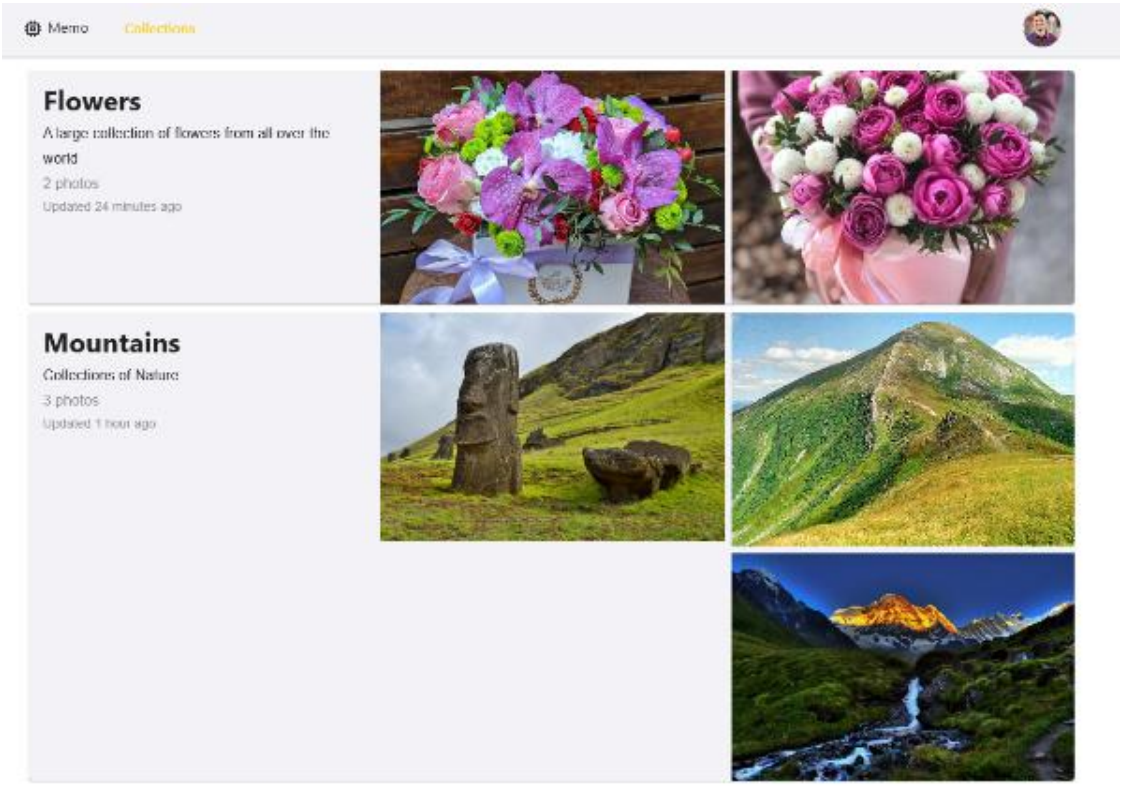


Рисунок 4.7 – Сторінка з колекціями

На рисунку 4.8 зображена сторінка користувача з фотографіями.

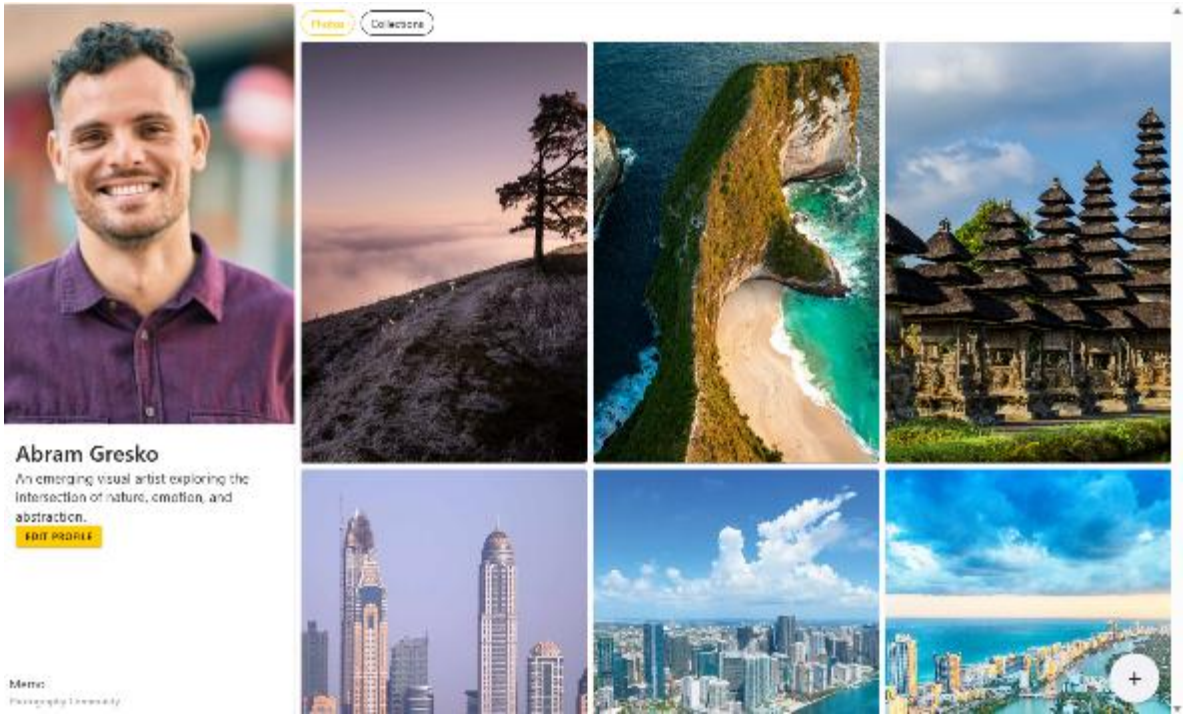


Рисунок 4.8 – Сторінка користувача з фотографіями

На рисунку 4.9 зображено фільтр на сторінці користувача між фотографіями і колекціями.

Рисунок 4.9 – Вибір між фотографіями і колекціями

На рисунку 4.10 зображено сторінку користувача з колекціями, колекція «Collection of Cities» зроблена з власними фотографіями, а колекція «Mountains» має фотографії іншого користувача. Також колекції можна редагувати та видаляти. Додавати колекції можна натиснувши на кнопку в нижній правій частині екрана.

На рисунку 4.11 зображено вікно редагування колекції.

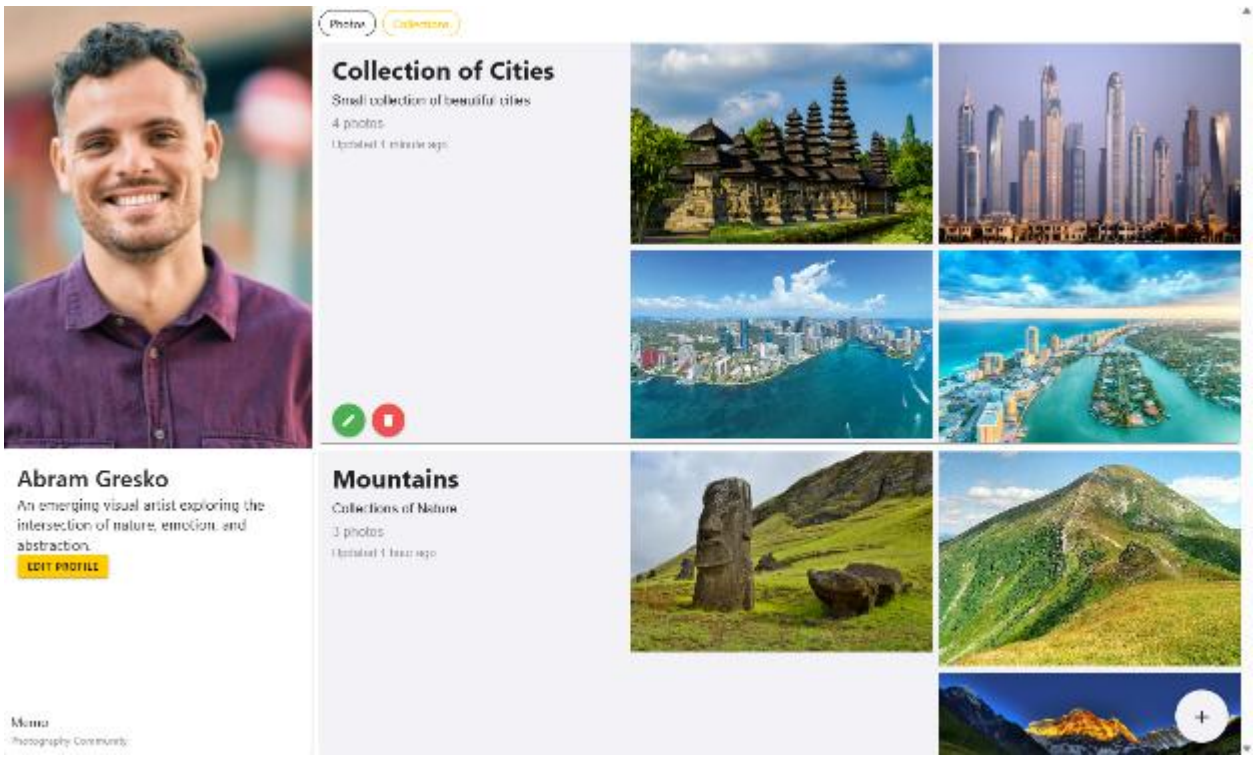


Рисунок 4.10 – Сторінка користувача з колекціями

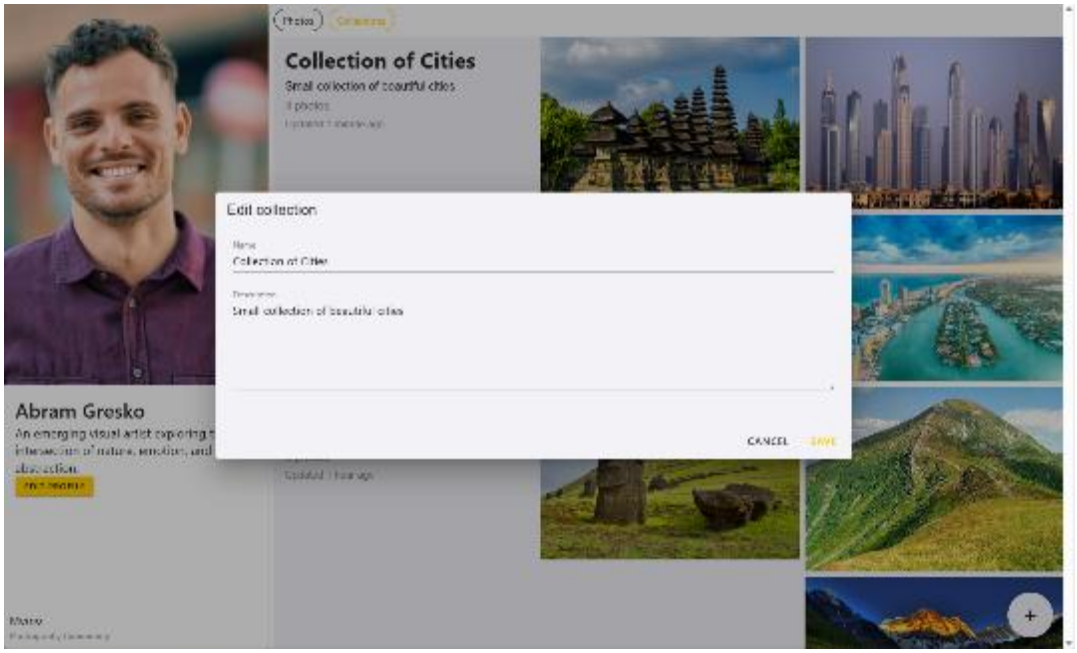


Рисунок 4.11 – Вікно редагування колекції

На рисунку 4.12 зображено вікно видалення колекції.

На рисунку 4.13 зображено вікно створення колекції.

На рисунку 4.14 зображено сторінку з фотографіями колекції.

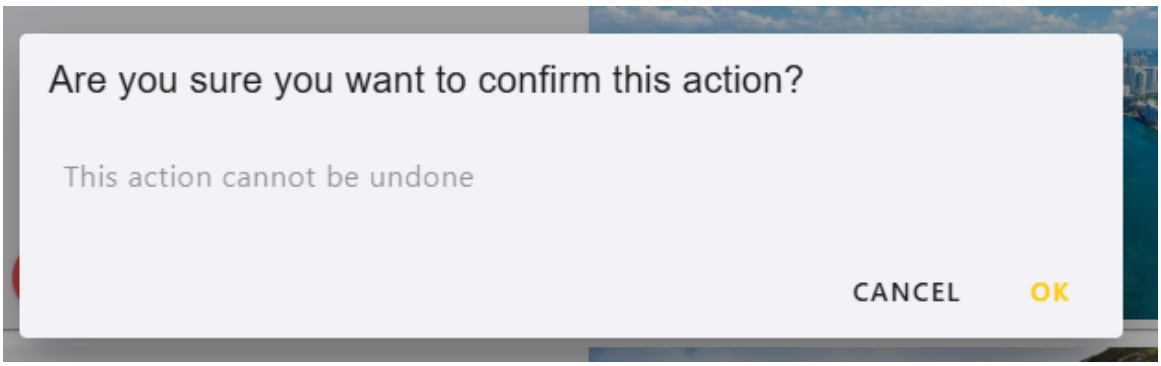


Рисунок 4.12 – Вікно видалення колекції

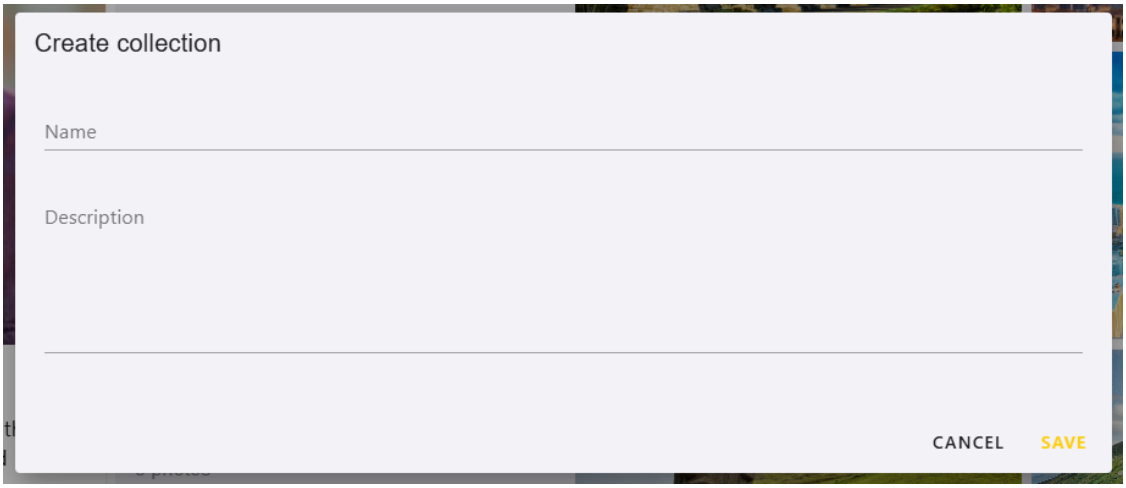


Рисунок 4.13 – Вікно створення колекції

Рисунок 4.14 – Сторінка з фотографіями колекції

На рисунку 4.15 зображено вікно редагування інформації користувача, кнопка для відкриття доступна тільки власнику аккаунта.

Рисунок 4.15 – Вікно редагування інформації користувача

На рисунку 4.16 зображено вікно з фотографією, яке відкривається при натисканні на фотографію. У користувача є можливість, закрити вікно, через стрілочки «вліво» та «вправо» користувач має змогу переключатись між фотографіями.

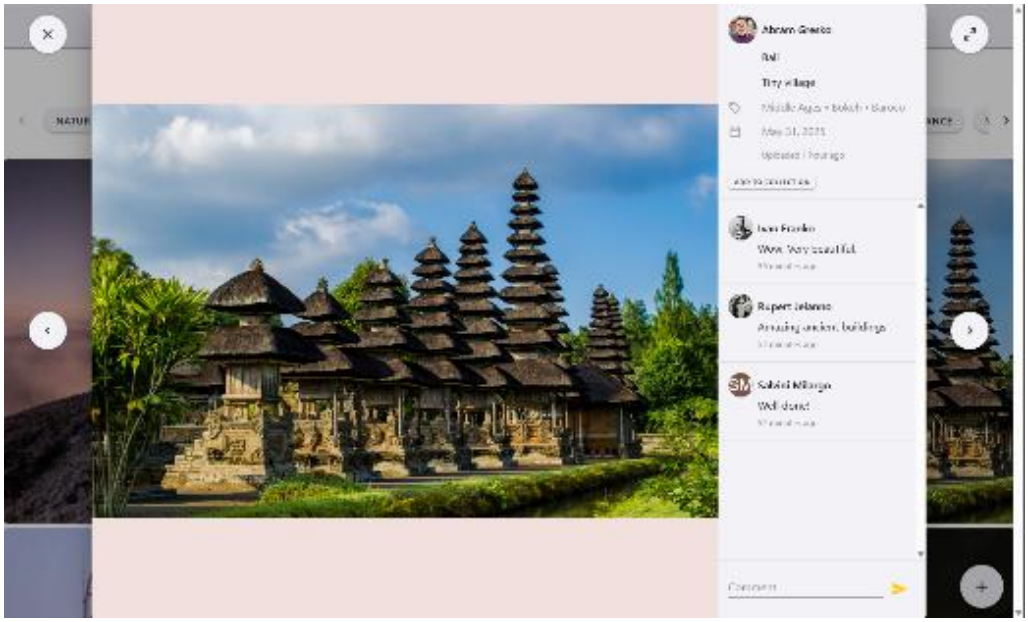


Рисунок 4.16 – Вікно з інформацією про фотографію

На рисунку 4.17 зображено секцію при перегляді фотографії з додатковою інформацією та можливістю додати фото до колекції.

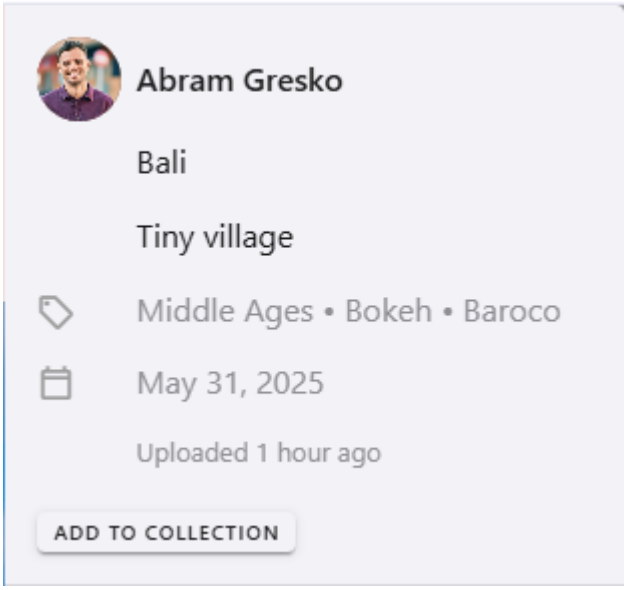


Рисунок 4.17 – Секція з інформацією про фотографію, та можливістю додати фото до колекції

На рисунку 4.18 зображено секцію з коментарями фотографії.

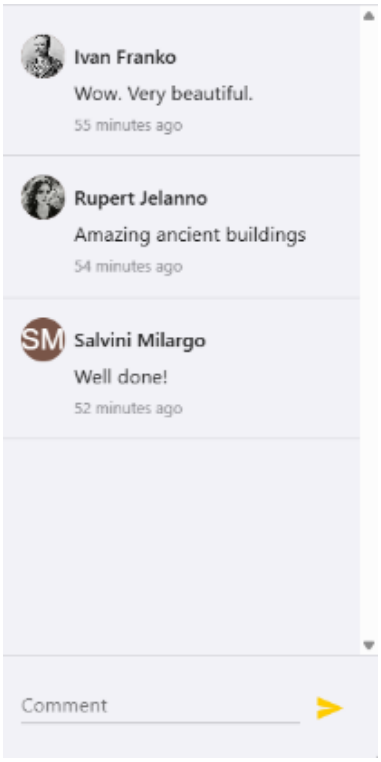


Рисунок 4.18 – Секція з коментарями фотографії

На рисунку 4.19 зображено секцію з коментарями фотографії, коли користувач не авторизований коментування недоступне.

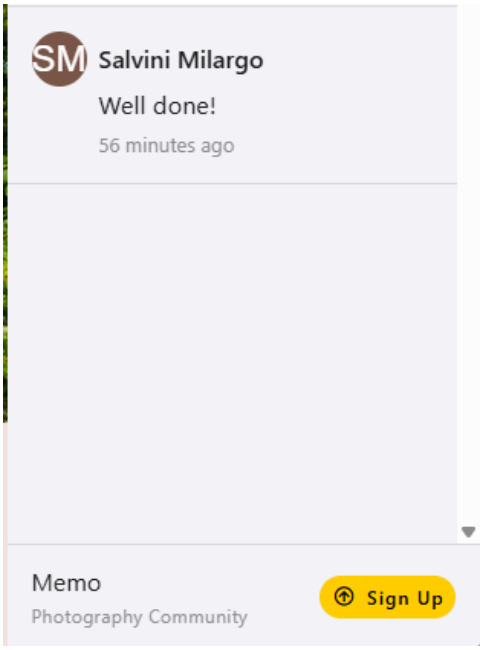


Рисунок 4.19 – Секція з коментарями фотографії коли користувач не авторизований

На рисунку 4.20 зображено меню профіля і кнопку переходу в admin панель, яка показана тільки для користувача з роллю адмін.

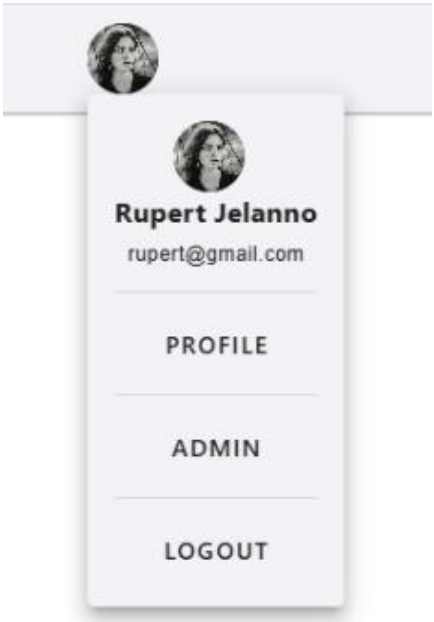


Рисунок 4.20 – Кнопка «Admin» для переходу на адмін панель

На рисунку 4.21 наведено таблицю користувачів в адмін панелі і кнопка «Go back» для того, щоб повернутись на головну сторінку.

На рисунку 4.22 наведено таблицю фотографій в адмін панелі.

На рисунку 4.23 наведено таблицю коментарів в адмін панелі.

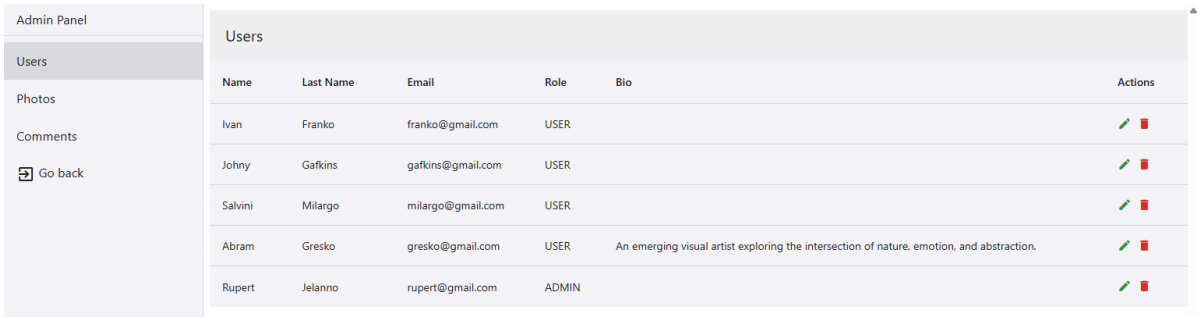


Рисунок 4.21 – Адмін панель, таблиця користувачів

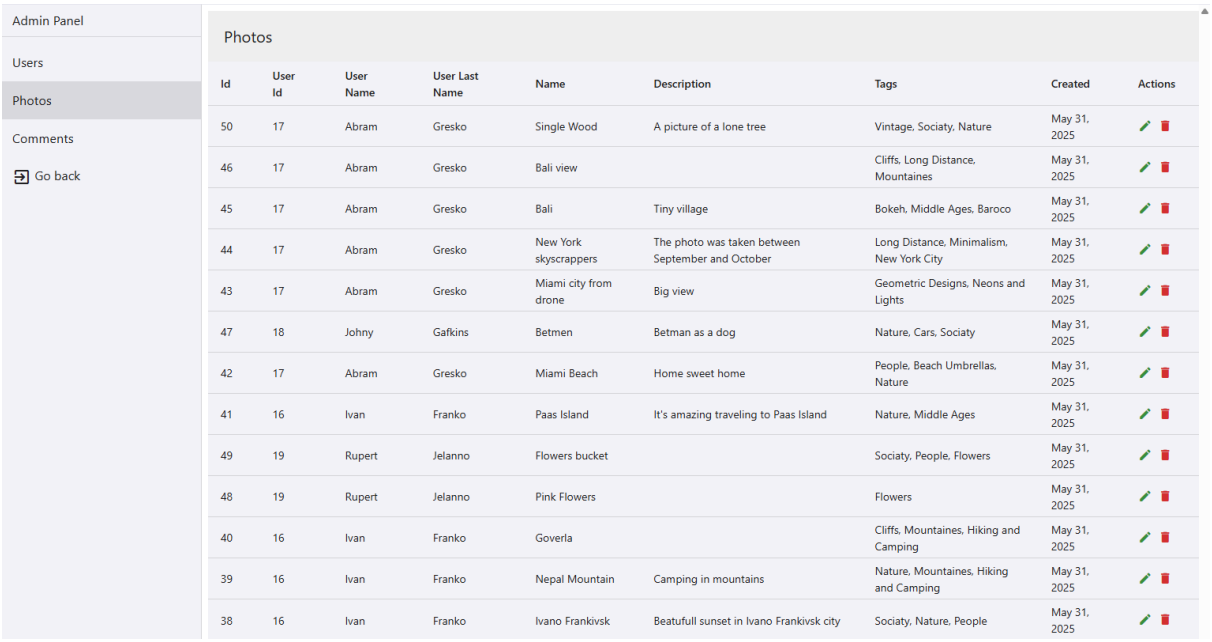


Рисунок 4.22 – Адмін панель, таблиця фотографій

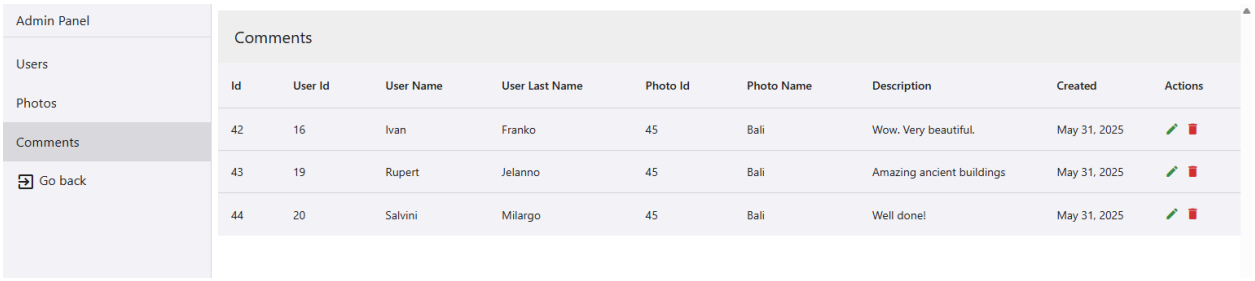


Рисунок 4.23 – Адмін панель, таблиця коментарів

4.5 Висновок до розділу

Умови використання розробленого додатку базуються на ретельно підібраному стеку сучасних технологій та розумних апаратних вимогах. Розділення між фронтендом і бекендом забезпечує модульність, масштабованість і ремонтопридатність, що робить його придатним для розгортання як в локальних середовищах, так і на виробничих серверах.

Завдяки використанню широко розповсюджених фреймворків та інструментів, таких як Vue.js для фронтенду та Node.js з Sequelize для бекенду, додаток гарантує швидку продуктивність, чуйність та легкість подальшого розвитку. Необхідні серверні ресурси є помірними, що дозволяє розгортання навіть на бюджетній хмарній інфраструктурі.

Заходи безпеки, такі як хешування паролів за допомогою bcrypt, а також структуровані моделі баз даних і валідовані користувацькі дані, сприяють надійності та безпеці системи.

Загалом, система є надійною, легкою та гнучкою, що дозволяє їй відповідати як поточним потребам користувачів, так і майбутнім цілям масштабування з мінімальними витратами на конфігурацію.

ВИСНОВКИ

В ході виконання даної дипломної кваліфікаційної роботи всі поставлені цілі були успішно досягнуті. Було проведено всебічний аналіз предметної області, особливу увагу було приділено потребам користувачів та адміністраторів, які взаємодіють з платформами фотоконтенту. Це дослідження заклало основу для проєктування та реалізації повнофункціонального вебдодатку, який є зручним для користувачів та масштабованим.

Реалізація проєкту базувалася на сучасному та ефективному технологічному стеку. Фронтенд був розроблений за допомогою Vue.js, що забезпечило чуйний та інтуїтивно зрозумілий користувацький інтерфейс. Бекенд був реалізований за допомогою Node.js та Express.js, підтримувався базою даних PostgreSQL та управлявся за допомогою Sequelize ORM. Обробка файлів та автентифікація були покращені завдяки інтеграції з bcrypt та JWT (JSON Web Tokens) для безпечної обробки даних та авторизації.

Фінальний додаток підтримує основні функції, такі як реєстрація та вхід користувачів, додавання та видалення фотоматеріалів, розміщення та управління коментарями, а також надає адміністративну панель для ефективного управління контентом та користувачами. Особливу увагу було приділено безпеці додатку за допомогою таких практик, як хешування паролів та безпечне управління сесіями.

Крім того, система дозволяє користувачам створювати колекції фотографій та керувати ними, додавати теги до зображень, а також взаємодіяти з контентом у цікавий та зручний спосіб. Адміністраторам надаються потужні інструменти для нагляду за діяльністю користувачів і модерації контенту.

Отримана програма є швидкою, легкою, безпечною та оптимізованою для сучасних браузерів. Він забезпечує плавну і візуально привабливу роботу як на настільних, так і на мобільних пристроях. Його модульна архітектура

дозволяє легко оновлювати і масштабувати його в майбутньому, що робить його придатним для подальшого розвитку або розгортання в реальних умовах.

Таким чином, розроблений вебдодаток не тільки відповідає цілям проєкту, але й забезпечує міцну основу для подальших удосконалень і потенційної комерціалізації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 500px / [Electronic resource] – Access mode: https://500px.com/.
2. Про Flickr / [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Flickr.
3. Про Behance / [Електрон. ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Behance.
4. MDN web docs / [Electronic resource] – Access mode: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
5. JavaScript Info: / [Electronic resource] – Access mode: https://javascript.info/intro.
6. Vue 3 / [Electronic resource] – Access mode: https://vuejs.org/.
7. Vuetify 3 / [Electronic resource] – Access mode: https://vuetifyjs.com/en/.
8. Pinia / [Electronic resource] – Access mode: https://pinia.vuejs.org/.
9. Axios / [Electronic resource] – Access mode: https://axios-http.com/docs/intro.
10. Day.js / [Electronic resource] – Access mode: https://day.js.org/.
11. Sass / [Electronic resource] – Access mode: https://sass-lang.com/.
12. Node.js / [Electronic resource] – Access mode: https://nodejs.org/en.
13. Express / [Electronic resource] – Access mode: https://expressjs.com/.
14. Sequelize ORM/ [Electronic resource] – Access mode: https://sequelize.org/.
15. PostgreSQL / [Electronic resource] – Access mode: https://www.postgresql.org/.
16. Postman / [Electronic resource] – Access mode: https://www.postman.com/.
17. What is GitHub? / [Electronic resource] – Access mode: https://www.techtarget.com/searchitoperations/definition/GitHub.

18. Розуміння Клієнт-Серверної Архітектури / [Електрон. ресурс] – Режим доступу: https://dou.ua/forums/topic/44636/.

ДОДАТОК А

Технічне завдання

Вступ

Програмне забезпечення призначене для використання фотографами зі всього світу, для публікації, коментування та створення колекцій фотографій.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація онлайн-сервісу для фотографів», затверджене наказом Національного університету «Запорізька політехніка» №209 від 28 квітня 2025 р.

А.2 Основні вимоги до програми, що розробляється

Додаток повинен відповідати сучасним стандартам розробки програмного забезпечення, щоб забезпечити стабільну, безпечну та зручну для користувача роботу. Він повинен підтримувати основну функціональність для кінцевих користувачів і адміністраторів, підтримувати узгоджений інтерфейс на різних платформах і демонструвати надійну поведінку в очікуваних умовах експлуатації. Ці вимоги необхідні для того, щоб гарантувати безперебійну та ефективну роботу користувачів на всіх етапах взаємодії з системою.

А.2.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик:

- реєстрація та аутентифікація користувачів;
- завантаження та публікація фотоматеріалів;
- перегляд робіт інших користувачів;
- створення колекцій з фотографій інших користувачів;
- коментування фотопублікацій;

- модерація коментарів адміністратором;
- відображення популярних фотографій;
- адмін-панель для управління контентом;
- інструменти захисту авторських прав (заборона копіювання).

А.2.2 Вимоги до інтерфейсу програми

Вимоги до інтерфейсу програми:

- адаптивний дизайн до різних розмірів екрану;
- перегляд фотографій у вигляді сітки для сторінок галереї;
- модальний або повноекранний режим для попереднього перегляду окремих фотографій;
- секції коментарів під кожною фотографією;
- зрозуміла та доступна панель навігації.

А.2.3 Вимоги до надійності

Вимоги до надійності:

- відмовостійка автентифікація користувачів та керування сеансами;
- безпечна робота з завантаженням та зберіганням медіафайлів;
- надійна обробка помилок та валідація як на рівні фронтенду, так і на рівні бекенду;
- збереження та узгодженість даних завдяки PostgreSQL та Sequelize ORM;
- заборона на копіювання авторського контенту.

А.2.4 Вимоги до складу та параметрів технічний засобів

Створений вебзастосунок має наступні вимоги апаратного забезпечення, на якому буде працювати серверна частина онлайн сервісу для фотографів:

- процесор: 4 ядра 2.0 ГГц;
- оперативна пам’ять: мінімум 4 Гб;
- сховище: SSD з принаймні 50 Гб вільного місця;
- node.js версія: мінімум 18 версія;
- версія PostgreSQL: 13 версія і вище.

ДОДАТОК Б

Текст програми

Б.1 Текст файлу filesHelper.js

```
const fs = require('fs')
const uuid = require('uuid')
const path = require('path')

class FilesHelper {
  userName = ''
  userLastName = ''
  userId = ''
  folderName = ''
  fileName = ''

  constructor(userName = '', userLastName = '', userId = '', filePrefix = '') {
    this.userName = userName
    this.userLastName = userLastName
    this.userId = userId

    this.generateFolderName()
    this.generateFileName(filePrefix)
  }

  generateFolderName() {
    const { userName, userLastName, userId } = this
    this.folderName = `${userName}_${userLastName}_${userId}`.trim()
  }

  generateFileName(filePrefix) {
    this.fileName = filePrefix + uuid.v4() + '.jpg'
  }

  uploadPhotoToFolder(image) {
    /* Upload photo to folder and create folder if needed */
    try {
      const { folderName, fileName } = this
      const uploadPath = path.resolve(__dirname, '..', 'static', folderName)
      const filePath = path.join(uploadPath, fileName)

      if (!fs.existsSync(uploadPath)) {
        fs.mkdirSync(uploadPath)
      }

      image.mv(filePath)
    } catch (error) {
```

```
      throw error
    }
  }
}

module.exports = FilesHelper
```

Б.2 Текст файлу helper.js

```
const jsonwebtoken = require('jsonwebtoken')

class Helper {
  static getJwtTokenFromRequest(req = {}) {
    const { authorization = '' } = req.headers
    const token = authorization.split(' ')[1]


    return token || null
  }

  static decodeJwtToken(req) {
    const token = Helper.getJwtTokenFromRequest(req)
    if (!token) return {}

    const decode = jsonwebtoken.decode(token, process.env.JWT_SECRET)
    return {
      userId: decode.id,
      userName: decode.name,
      userLastName: decode.last_name,
    }
  }

  static verifyJwtToken(req) {
    const token = Helper.getJwtTokenFromRequest(req)
    if (!token) return false

    return jsonwebtoken.verify(token, process.env.JWT_SECRET)
  }

  static createPhotoUrl(photo) {
    if (typeof photo !== 'object') return ''
    const { image, image_folder } = photo
    if (!image || !image_folder) return null
```

```
    return `${process.env.HOST_NAME}/${image_folder}/${image}`
  }
}

module.exports = Helper
```

Б.3 Текст файлу user.js

```
const createUserAvatarUrl = (user) => {
  if (typeof user !== 'object') return ''
  const { avatar, avatar_folder } = user
  if (!avatar || !avatar_folder) return null

  return `${process.env.HOST_NAME}/${avatar_folder}/${avatar}`
}

const prepareUserData = (user = {}) => {
  if (typeof user !== 'object') return {}

  const { id, name, last_name, email, role, avatar, bio, avatar_folder } = user
  const avatar_src = avatar && avatar_folder ? createUserAvatarUrl(user) : null

  return { id, name, last_name, email, role, bio, avatar_src }
}

module.exports = { prepareUserData, createUserAvatarUrl }
```

Б.4 Текст файлу collectionController.js

```
const ApiError = require('../error/ApiError')
const { prepareUserData } = require('../helpers/user')
const Helper = require('../helpers/helper')
const {
  Collection,
  CollectionPhotos,
  Photo,
  User,
  Tag,
} = require('../models/models')

const userObject = {
```

```
  model: User,
}

const tagObject = {
  model: Tag,
  attributes: ['id', 'name'],
  through: { attributes: [] },
}

class CollectionController {
  async createCollection(req, res, next) {
    const { name, description } = req.body
    const { user } = req

    if (!name) return next(ApiError.badRequest('Name is required'))
    if (!user?.id) return next(ApiError.badRequest('Something went wrong'))

    try {
      const collection = await Collection.create({
        user_id: user?.id,
        name,
        description,
      })

      if (!collection)
        return next(
          ApiError.badRequest(
            'Error when creating a collection. Try again later'
          )
        )

      res.status(200).json({
        data: collection,
        message: 'Collection was successfully created',
        success: true,
      })
    } catch (error) {
      next(ApiError.badRequest(error.message))
    }
  }

  async editCollection(req, res, next) {
    const { id } = req.params
    const { name, description } = req.body
    if (!id) return next(ApiError.badRequest('Collection ID is required'))
```

```
  const collection = await Collection.findByPk(id)

  if (!collection) {
    return next(ApiError.notFound('Collection not found'))
  }

  if (name !== undefined) collection.name = name
  if (description !== undefined) collection.description = description

  await collection.save()

  return res.status(200).json({
    data: collection,
    message: 'Collection was successfully updated',
    success: true,
  })
}

async addPhotoToCollection(req, res, next) {
  const { photo_ids = [], collection_id } = req.body

  if (!Array.isArray(photo_ids) || !photo_ids.length || !collection_id)
    return next(ApiError.badRequest('Invalid data'))

  try {
    const collection = await Collection.findByPk(collection_id)
    if (!collection) return next(ApiError.badRequest('Collection not found'))

    const existingLinks = await CollectionPhotos.findAll({
      where: {
        collection_id,
        photo_id: photo_ids,
      },
    })

    const existingPhotoIds = existingLinks.map((link) => link.photo_id)
    const newPhotoIds = photo_ids.filter(
      (id) => !existingPhotoIds.includes(Number(id))
    )

    const photos = await Photo.findAll({
      where: { id: newPhotoIds },
    })
```

```
    if (!photos.length)
      return res.status(200).json({
        message: 'No new photos to add',
        success: true,
      })

    await collection.addPhotos(photos)

    res.status(200).json({
      message: 'Photo was added succesfully',
      success: true,
    })
  } catch (error) {
    next(ApiError.badRequest(error.message))
  }
}

async getCollectionsPreview(req, res, next) {
  try {
    const { user_id } = req.query
    const where = user_id ? { user_id } : {}
    const photoLimit = 4

    const collections = await Collection.findAll({
      where,
      include: [userObject],
      order: [['createdAt', 'DESC']],
    })

    const formatted = await Promise.all(
      collections.map(async (collection) => {
        const data = collection.toJSON()
        const photos = await collection.getPhotos({
          limit: photoLimit,
          order: [['createdAt', 'DESC']],
        })
        const photosCount = await collection.countPhotos()

        return {
          ...data,
          photos: photos.map((item) => ({
            id: item.id,
            name: item.name,
            src: Helper.createPhotoUrl(item),
          })),
```

```
          photosCount,
          user: prepareUserData(data.user),
        }
      })
    )

    res.status(200).json({
      data: formatted,
      success: true,
    })
  } catch (error) {
    next(ApiError.badRequest(error.message))
  }
}

async getCollection(req, res, next) {
  try {
    const { id } = req.query
    if (!id) return next(ApiError.badRequest('Please, try later'))

    const collection = await Collection.findByPk(id, {
      include: [userObject],
    })
    if (!collection)
      return next(ApiError.badRequest("Collection doesn't exist"))
    const photosCount = await collection.countPhotos()
    const collectionData = collection.toJSON()

    res.status(200).json({
      data: {
        ...collectionData,
        user: prepareUserData(collectionData.user),
        photosCount,
      },
      success: true,
    })
  } catch (error) {
    next(ApiError.badRequest(error.message))
  }
}

async getCollectionPhotos(req, res, next) {
  try {
    const { id: collection_id } = req.query
    const { page = 1, limit = 25 } = req.query
```

```
    const offset = page * limit - limit
    if (!collection_id) return next(ApiError.badRequest('Invalid id'))

    const collection = await Collection.findByPk(collection_id)

    if (!collection) return next(ApiError.badRequest('Collection not found'))

    const photos = await Photo.findAndCountAll({
      include: [
        {
          model: Collection,
          where: { id: collection_id },
          attributes: [],
          through: { attributes: [] },
        },
        tagObject,
        userObject,
      ],
      distinct: true,
      limit: limit,
      offset: offset,
      order: [['createdAt', 'DESC']],
    })

    if (!photos) return next(ApiError.badRequest('Photos is not defined'))

    // add photo src
    photos.rows = photos.rows.map((item) => {
      item.dataValues.src = Helper.createPhotoUrl(item)
      return item
    })

    res.status(200).json({ data: photos, message: '', success: true })
  } catch (error) {
    next(ApiError.badRequest(error.message))
  }
}

async getUserCollectionsNames(req, res, next) {
  const { user_id } = req.query
  if (!user_id) return next(ApiError.badRequest('Wrong user id'))

  const collections = await Collection.findAll({ where: { user_id } })

  if (!collections) return next(ApiError.badRequest('No collections'))
```

```
    const formatted = collections.map((item) => {
      const { id, name } = item.toJSON()
      return { value: id, name }
    })

    res.status(200).json({
      data: formatted,
      success: true,
    })
  }

  async deleteCollection(req, res, next) {
    const { id } = req.params

    const collection = await Collection.findByPk(id)
    if (!collection) return next(ApiError.badRequest('No collection'))

    collection.destroy()

    res.status(200).json({
      message: 'Collection was successfully removed',
      success: true,
    })
  }
}

module.exports = new CollectionController()
```

Б.5 Текст файлу commetController.js

```
const ApiError = require('../error/ApiError')
const { createUserAvatarUrl } = require('../helpers/user')
const { Comment, User, Photo } = require('../models/models')

const userObject = {
  model: User,
  attributes: ['id', 'name', 'last_name', 'avatar', 'avatar_folder'],
}

class CommentController {
  async create(req, res, next) {
    const { photo_id, description } = req.body
```

```
  if (!req.user || !photo_id)
    return next(ApiError.badRequest('Invalid user or photo id'))

  if (!description)
    return next(ApiError.badRequest('The comment text is required'))

  const comment = await Comment.create({
    user_id: req.user?.id,
    photo_id,
    description,
  })

  if (!comment) return next(ApiError.badRequest())

  const findComment = await Comment.findByPk(comment.id, {
    include: [userObject],
  })
  const { user } = findComment.dataValues

  user.dataValues.avatar_src = createUserAvatarUrl(user)

  res.status(200).json({
    data: findComment,
    message: 'The comment was successfully created',
    success: true,
  })
}

async getPhotoComments(req, res, next) {
  const { photo_id } = req.query

  if (!photo_id) return next(ApiError.badRequest('Invalid photo_id'))

  const comments = await Comment.findAll({
    where: { photo_id },
    include: [userObject],
  })
  if (!comments) return next(ApiError.badRequest())

  comments.forEach((item) => {
    const { user } = item
    user.dataValues.avatar_src = createUserAvatarUrl(user)
  })
```

```
    res.status(200).json({
      data: comments,
      success: true,
    })
  }

  async getAllComments(req, res, next) {
    const comments = await Comment.findAll({
      include: [
        userObject,
        {
          model: Photo,
          attributes: ['id', 'name'],
        },
      ],
    })

    if (!comments) return res.status(200).json({ data: [], success: true })

    res.status(200).json({
      data: comments,
      success: true,
    })
  }
}

module.exports = new CommentController()
```

Б.6 Текст файлу photoController.js

```
const ApiError = require('../error/ApiError')
const Helper = require('../helpers/helper')
const FilesHelper = require('../helpers/filesHelper')
const { createUserAvatarUrl } = require('../helpers/user')
const { Photo, Tag, User, Collection } = require('../models/models')

const tagObject = {
  model: Tag,
  attributes: ['id', 'name'],
  through: { attributes: [] },
}

const userObject = {
```

```
  model: User,
  attributes: ['id', 'name', 'last_name', 'avatar', 'avatar_folder'],
}

const handleGetOnePhoto = async (id) => {
  const photo = await Photo.findOne({
    where: { id },
    include: [tagObject, userObject],
  })
  if (!photo) return null

  // add photo src
  photo.dataValues.src = Helper.createPhotoUrl(photo)

  return photo
}

class PhotoController {
  async createPhoto(req, res, next) {
    try {
      const { name, description = '', tags = [], collection = null } = req.body
      const { image } = req.files
      const { userId, userName, userLastName } = Helper.decodeJwtToken(req)
      const photoTags = JSON.parse(tags)

      /* Validations */
      if (!name) return next(ApiError.badRequest('Invalid data'))

      if (Array.isArray(image))
        return next(ApiError.badRequest('Required only one image'))
      if (!image) return next(ApiError.badRequest('Image required'))

      if (!userId || !userName || !userLastName)
        return next(ApiError.badRequest('Error with user data'))

      /* Upload photo to folder and db */
      const filesHelper = new FilesHelper(userName, userLastName, userId)
      filesHelper.uploadPhotoToFolder(image)

      const photo = await Photo.create({
        user_id: userId,
        name,
        description,
        image: filesHelper.fileName,
        image_folder: filesHelper.folderName,
```

```
    })

    if (!photo) return next(ApiError.badRequest())

    /* Find and assign tags to photo */
    const tagRecords = await Promise.all(
      photoTags.map(async (id) => {
        const tag = await Tag.findOne({
          where: { id },
        })
        return tag
      })
    )
    await photo.setTags(tagRecords)

    // add photo to collection (optional)
    if (collection) {
      const targetCollection = await Collection.findByPk(collection)
      if (!targetCollection) {
        return next(ApiError.badRequest('Collection not found'))
      }

      await photo.addCollection(targetCollection)
    }

    const fullPhoto = await handleGetOnePhoto(photo.dataValues.id)
    if (!fullPhoto) return next(ApiError.badRequest('Photo is not defined'))

    return res.status(200).json({
      data: fullPhoto,
      success: true,
      message: 'Photo has been successfully created',
    })
  } catch (error) {
    next(ApiError.badRequest(error.message))
  }
}

async getOne(req, res, next) {
  try {
    const { id } = req.params
    if (!id) return next(ApiError.badRequest('Invalid id'))

    const photo = await handleGetOnePhoto(id)
    if (!photo) return next(ApiError.badRequest('Photo is not defined'))
```

```
    res.status(200).json({
      data: { ...photo.dataValues },
      message: '',
      success: true,
    })
  } catch (error) {
    next(ApiError.badRequest(error.mesage))
  }
}

async getAllUserPhotos(req, res, next) {
  try {
    const { id } = req.params
    const { page = 1, limit = 25 } = req.query
    const offset = page * limit - limit
    if (!id) return next(ApiError.badRequest('Invalid id'))

    const photos = await Photo.findAndCountAll({
      where: { user_id: id },
      include: [tagObject, userObject],
      order: [['createdAt', 'DESC']],
      distinct: true,
      limit,
      offset,
    })

    if (!photos) return next(ApiError.badRequest('Photos is not defined'))

    // add photo src
    photos.rows = photos.rows.map((item) => {
      item.dataValues.src = Helper.createPhotoUrl(item)
      item.dataValues.user.dataValues.avatar_src = createUserAvatarUrl(
        item.dataValues.user
      )
      return item
    })

    res.status(200).json({ data: photos, message: '', success: true })
  } catch (error) {
    next(ApiError.badRequest(error.message))
  }
}

async getAll(req, res, next) {
```

```
    try {
      const { page = 1, limit = 25 } = req.query
      const offset = page * limit - limit
      const photos = await Photo.findAndCountAll({
        include: [tagObject, userObject],
        order: [['createdAt', 'DESC']],
        distinct: true,
        limit,
        offset,
      })

      if (!photos) return next(ApiError.badRequest('Photos is not defined'))

      // add photo src
      photos.rows = photos.rows.map((item) => {
        item.dataValues.src = Helper.createPhotoUrl(item)
        item.dataValues.user.dataValues.avatar_src = createUserAvatarUrl(
          item.dataValues.user
        )
        return item
      })

      res.status(200).json({ data: photos, message: '', success: true })
    } catch (error) {
      next(ApiError.badRequest(error.message))
    }
  }
}

module.exports = new PhotoController()
```

Б.7 Текст файлу tagController.js

```
const ApiError = require('../error/ApiError')
const { Tag } = require('../models/models')

class TagController {
  async create(req, res, next) {
    const { name } = req.body
    if (!name) return next(ApiError.badRequest('Invalid name'))

    const tag = await Tag.create({ name }, { fields: ['id', 'name'] })
    if (!tag) return next(ApiError.badRequest())
```

```
    const minimalTag = { id: tag.id, name: tag.name }

    res.status(200).json({
      data: minimalTag,
      message: 'Tag created successfully',
      success: true,
    })
  }

  async delete(req, res, next) {
    const { id } = req.body
    if (!id) return next(ApiError.badRequest('Invalid id'))

    const tag = await Tag.destroy({ where: { id } })
    if (!tag) return next(ApiError.badRequest('Tag is not defined'))

    res.status(200).json({ message: 'Tag deleted successfully', success: true })
  }

  async getTags(req, res, next) {
    const tags = await Tag.findAll()
    if (!tags) return next(ApiError.badRequest())

    const formatted = tags.map((item) => {
      const { id, name } = item.toJSON()
      return { value: id, name }
    })

    res.status(200).json({
      data: formatted,
      message: '',
      success: true,
    })
  }
}

module.exports = new TagController()
```

Б.8 Текст файлу userController.js

```
const bcrypt = require('bcrypt')
const jsonwebtoken = require('jsonwebtoken')
```

```
const FilesHelper = require('../helpers/filesHelper')
const ApiError = require('../error/ApiError')
const validator = require('../helpers/validator')
const Helper = require('../helpers/helper')
const { User } = require('../models/models')
const { prepareUserData } = require('../helpers/user')

const createJWTToken = (user = {}) => {
  if (typeof user !== 'object') return ''
  const { id, name, last_name, email } = user

  return jsonwebtoken.sign(
    { id, name, last_name, email },
    process.env.JWT_SECRET,
    { expiresIn: '24h' }
  )
}

class UserController {
  async registration(req, res, next) {
    const { name, last_name, email, password, desciption } = req.body
    const { avatar = null } = req.files || {}

    if (!validator.validatePassword(password))
      return next(
        ApiError.badRequest('Password must be greater than 6 symbols')
      )

    if (!name || !last_name || !email)
      return next(ApiError.badRequest('Invalid data'))

    const findUser = await User.findOne({ where: { email } })

    if (findUser)
      return next(ApiError.badRequest('User with current email already exist'))

    const hashPassword = await bcrypt.hash(password, 5)

    const user = await User.create({
      name,
      last_name,
      email,
      password: hashPassword,
      desciption,
```

```
    })

    if (!user) return next(ApiError.badRequest())

    // save avatar image
    if (avatar) {
      const filesHelper = new FilesHelper(name, last_name, user.id, 'avatar_')
      filesHelper.uploadPhotoToFolder(avatar)

      user.avatar = filesHelper.fileName
      user.avatar_folder = filesHelper.folderName
      await user.save()
    }

    res.status(200).json({
      data: prepareUserData(user),
      jwt: createJWTToken(user),
      message: 'Registration successfuly',
      success: true,
    })
  }

  async login(req, res, next) {
    const { email, password } = req.body

    if (!email || !password) return next(ApiError.badRequest('Invalid data'))

    const user = await User.findOne({ where: { email } })
    if (!user)
      return next(ApiError.badRequest("User with current email doesn't exist"))

    const passwordMatch = await bcrypt.compare(password, user.password)
    if (!passwordMatch) return next(ApiError.badRequest('Wrong password'))

    res.status(200).json({
      data: prepareUserData(user),
      jwt: createJWTToken(user),
      message: 'Login successfuly',
      success: true,
    })
  }

  async checkAuth(req, res, next) {
    const { userId } = Helper.decodeJwtToken(req)
    const user = await User.findOne({ where: { id: userId } })
```

```
    if (!user) next(ApiError.badRequest())

    res.status(200).json({
      data: prepareUserData(user),
      jwt: createJWTToken(user),
      message: '',
      success: true,
    })
  }

  async getUserInfo(req, res, next) {
    const { id } = req.params
    if (!id) return next(ApiError.badRequest('Wrong user id'))

    const user = await User.findByPk(id)

    if (!user) return next(ApiError.badRequest("User doesn't exist"))

    res.status(200).json({
      data: prepareUserData(user),
      success: true,
    })
  }

  async getAllUsers(req, res, next) {
    try {
      const users = await User.findAll()

      if (!users) return res.status(200).json({ data: [], success: true })

      const prepearedUsers = users.map((item) => {
        const data = item.toJSON()
        return prepareUserData(data)
      })

      res.status(200).json({
        data: prepearedUsers,
        success: true,
      })
    } catch (error) {
      console.error(error)
    }
  }
}
module.exports = new UserController()
```

ДОДАТОК В

Слайди презентації

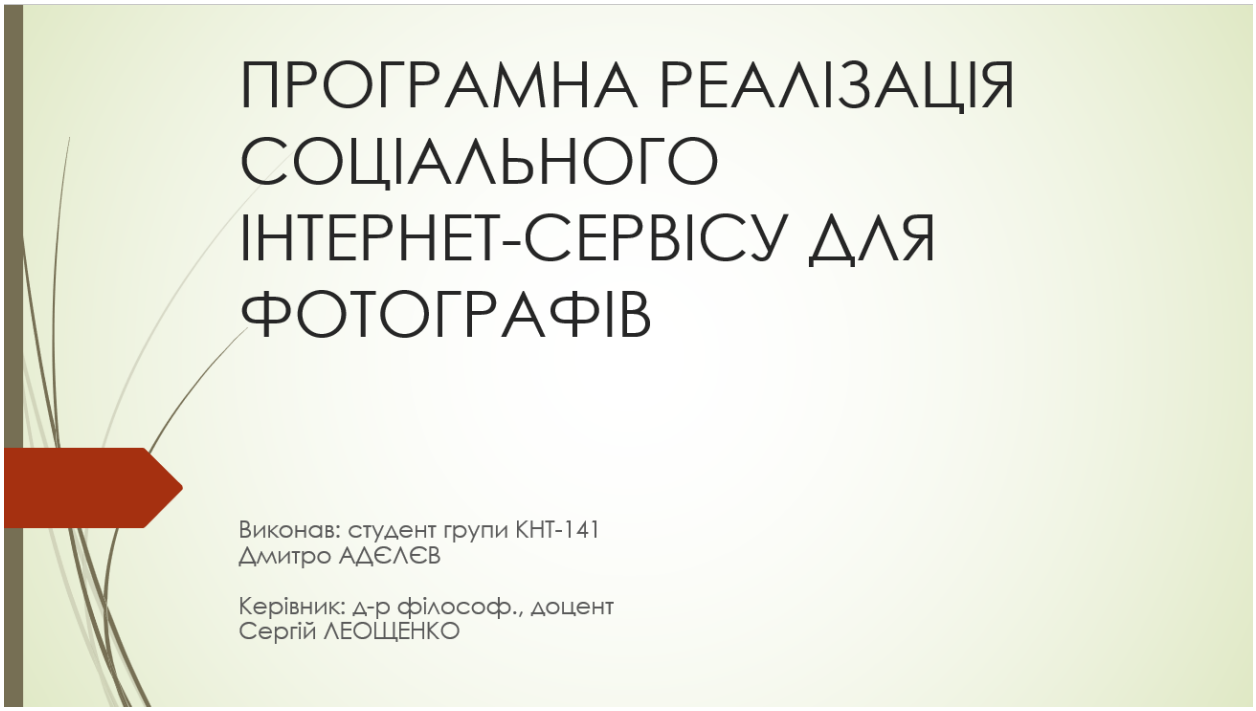


Рисунок В.1 – Слайд 1

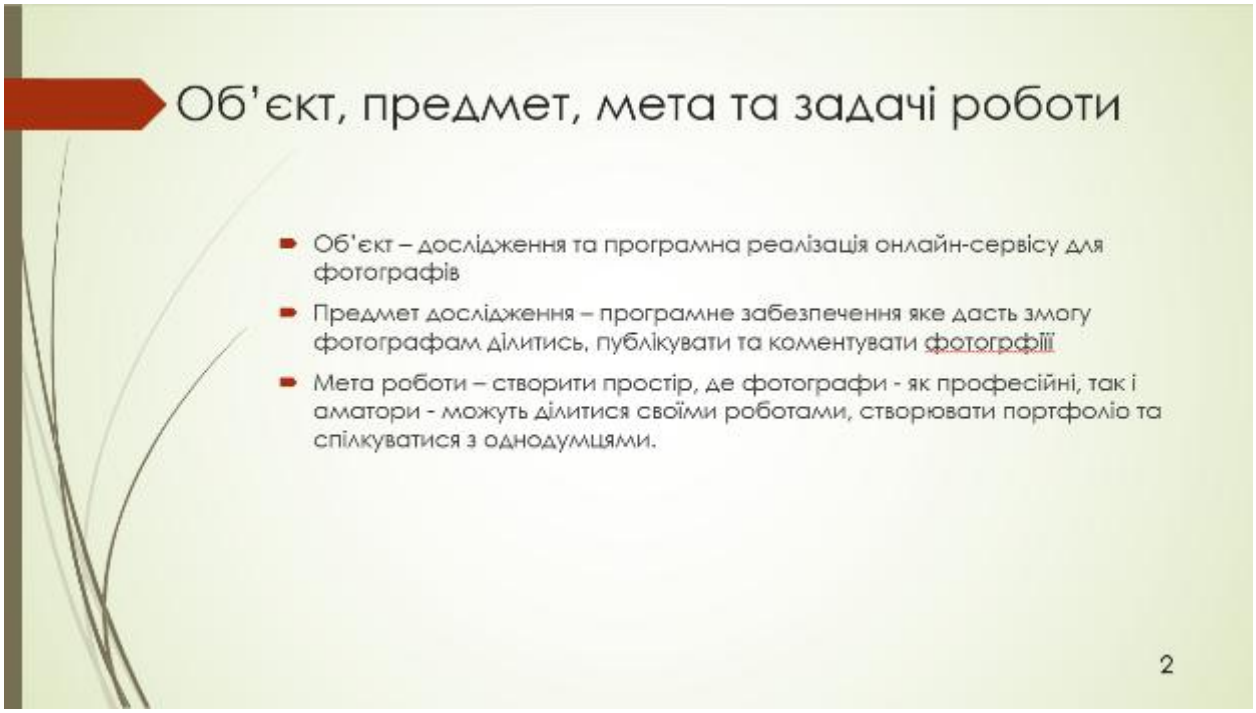


Рисунок В.2 – Слайд 2

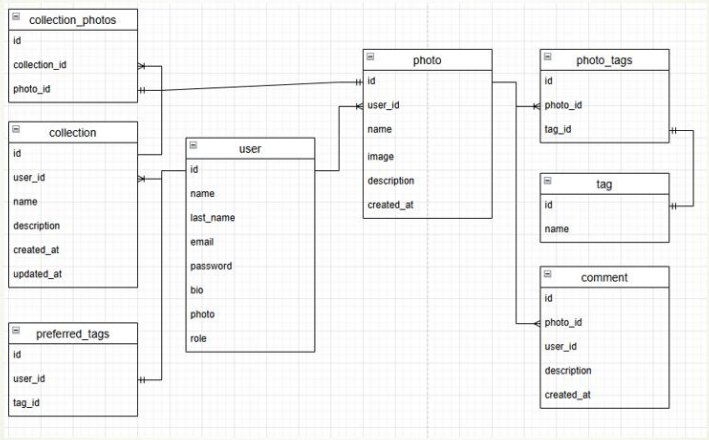
Порівняння існуючих аналогів

Критерій	500px	Flickr	Behance
1	2	3	4
Інтерфейс та зручність	Сучасний, мінімалістичний, з акцентом на фото	Застарілий інтерфейс, але функціональний	Стильний та професійний, інтегрований з Adobe
Презентація портфоліо	Якісна подача фото, чистий макет	Альбомна структура, гнучке сортування	Проєктна структура, зручна для візуального сторітелінгу
Соціальна взаємодія	Вподобання, коментарі, підписка	Вподобання, коментарі, групи, обговорення	Вподобання, "апрісієйшени", обмежена взаємодія
Активність спільноти	Добірки редакції, конкурси користувачів	Великі відкриті групи та тематичні спільноти	Відібрані проєкти, але менше взаємодії між користувачами
Захист авторських прав	Водяні знаки, ліцензування, обмежений контроль	Слабкий захист, можливість заборонити завантаження	Відсутність захисту, зображення легко копіювати
Модерація та адміністрування	Базові інструменти модерації коментарів та профілів	Групова модерація, система скарг	Мінімальні можливості, орієнтація на проєкти
Доступність та вартість	Безкоштовно з обмеженнями, є Pro-підписка	Безкоштовно до 1000 фото, потрібна Pro-версія для більшого	Безкоштовно, потрібен обліковий запис Adobe
Система рекомендацій	Популярне, добірка редакції	Сторінка «Explore», промоція у групах	Вибір редакції Adobe, ручна модерація
Можливість монетизації	Продаж фотографій через маркетплейс	Відсутня вбудована монетизація	Відсутня монетизація, лише для демонстрації портфоліо
Цільова аудиторія	Професійні та аматорські фотографи	Широке коло користувачів, ентузіасти	Креативні фахівці різних напрямків (не лише фотографи)

3

Рисунок В.3 – Слайд 3

Схема бази даних



4

Рисунок В.4 – Слайд 4

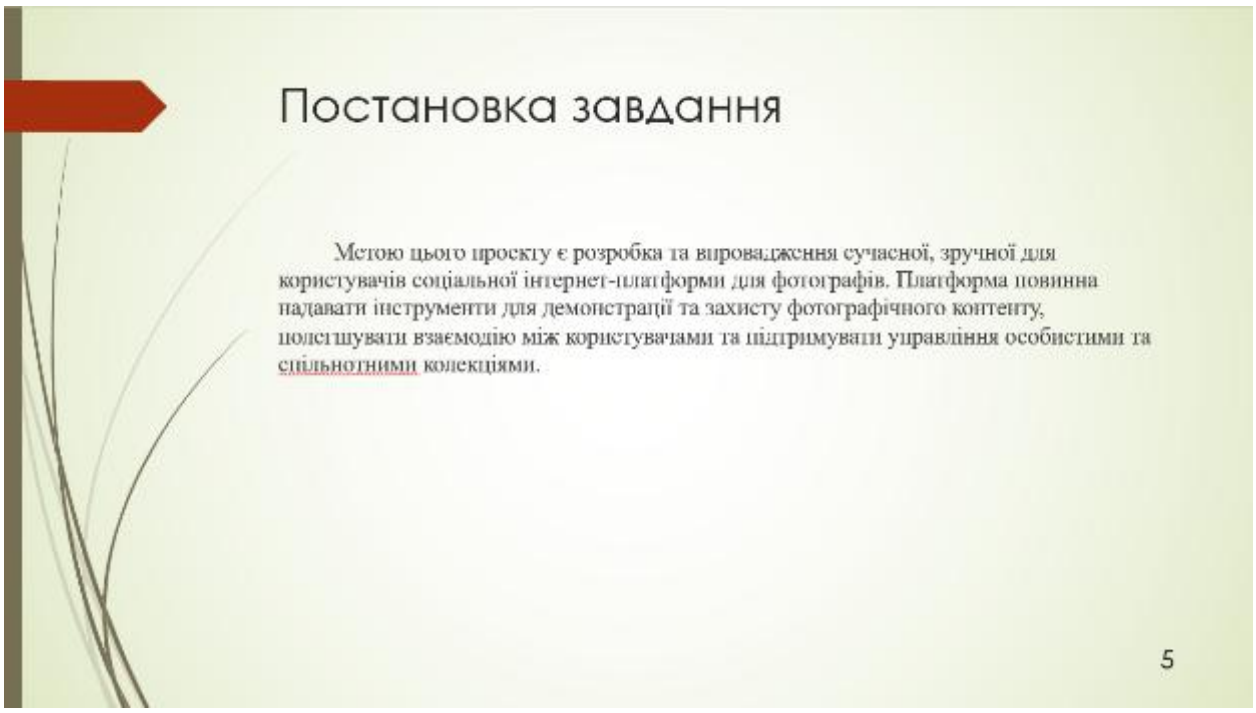


Рисунок В.5 – Слайд 5

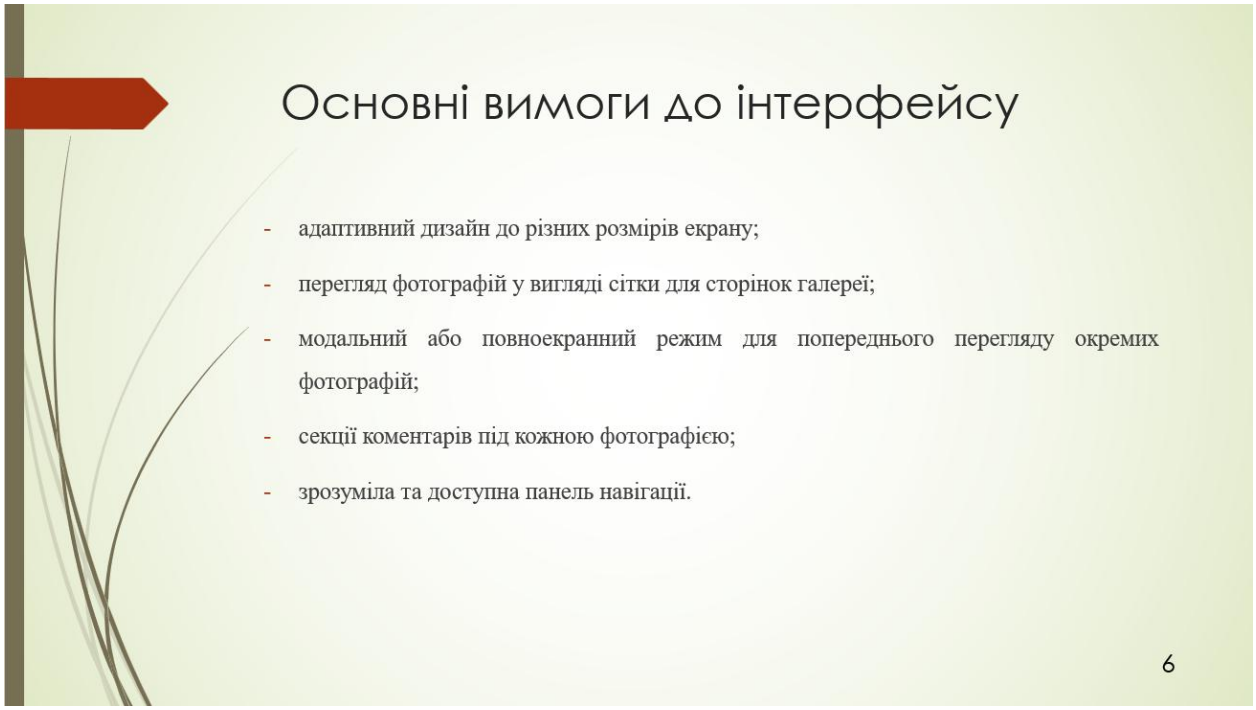


Рисунок В.6 – Слайд 6

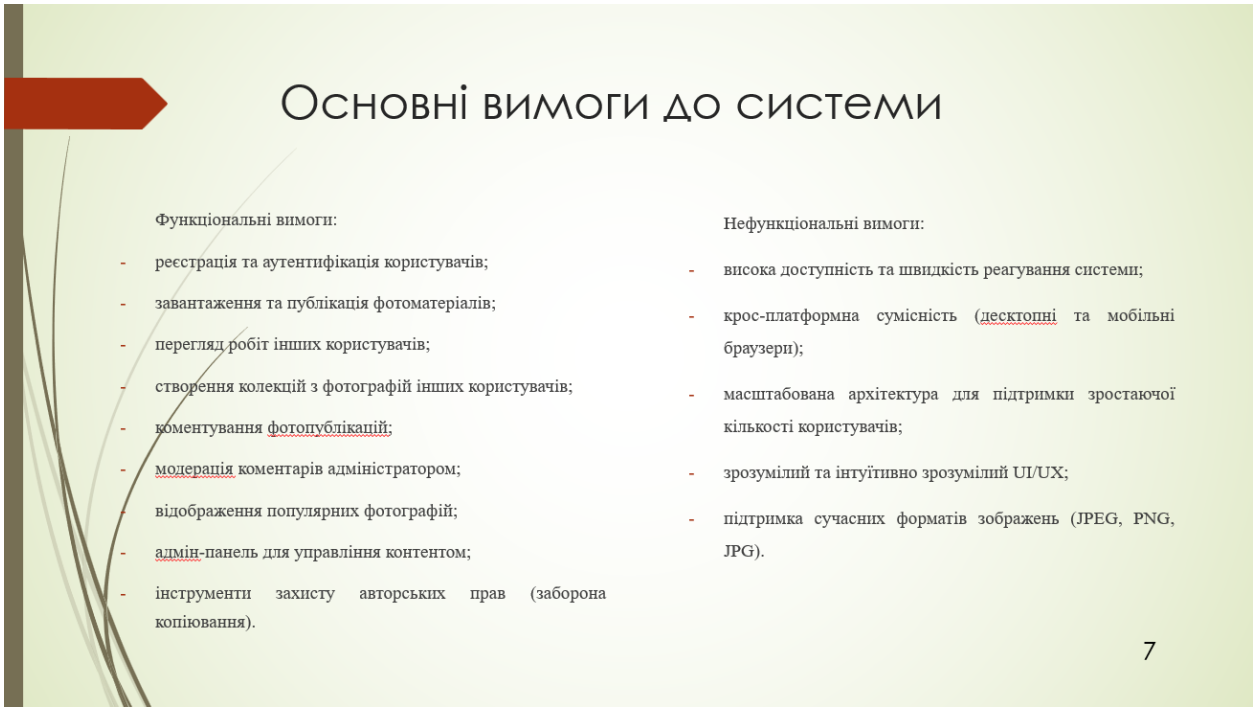


Рисунок В.7 – Слайд 7

Рисунок В.8 – Слайд 8

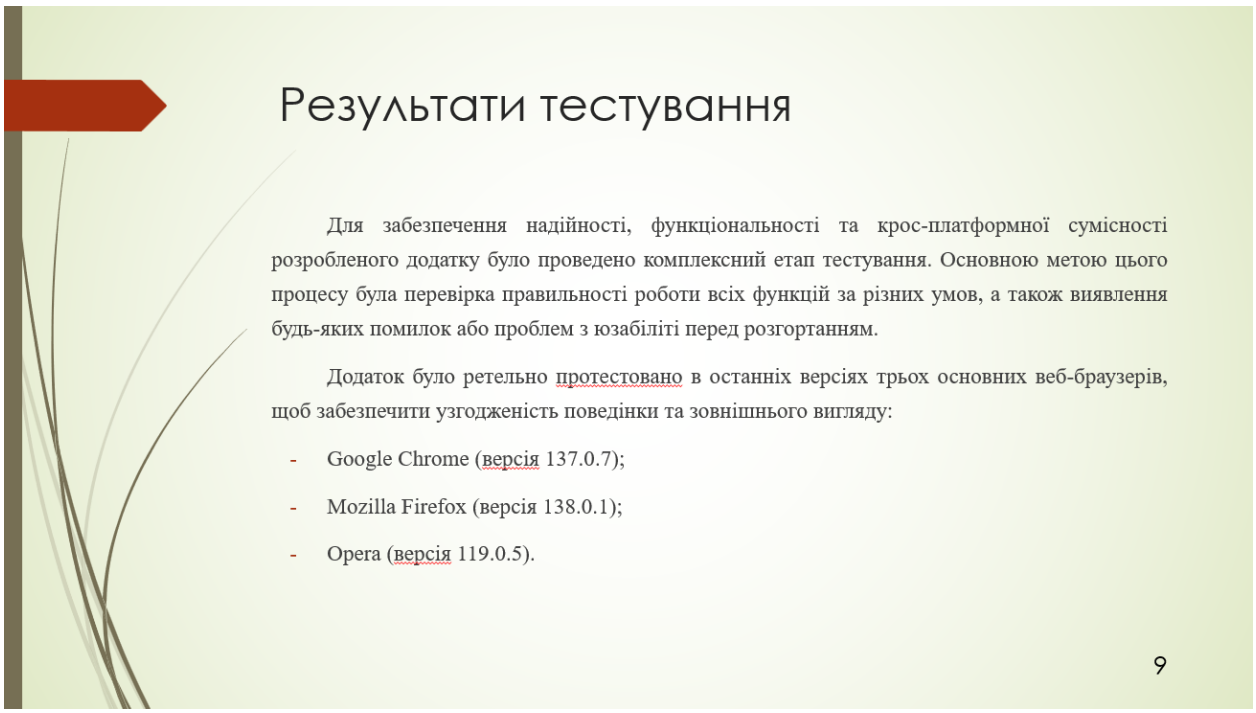


Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

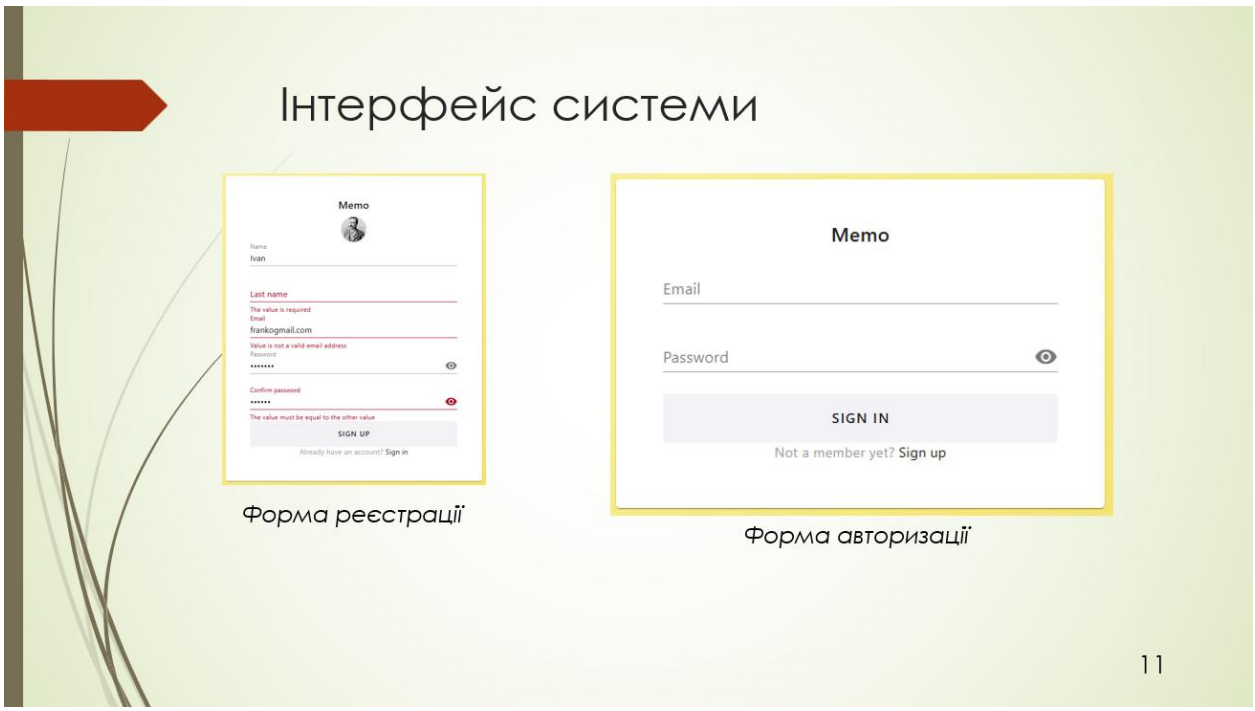


Рисунок В.11 – Слайд 11

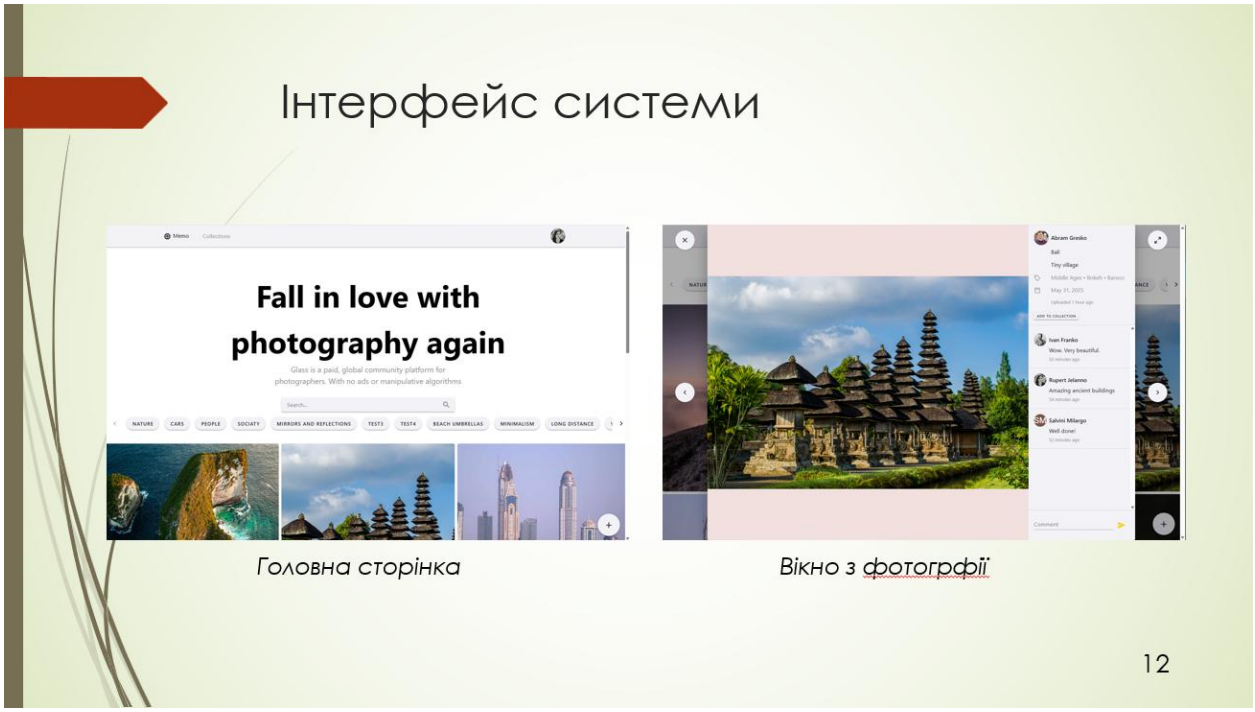


Рисунок В.12 – Слайд 12

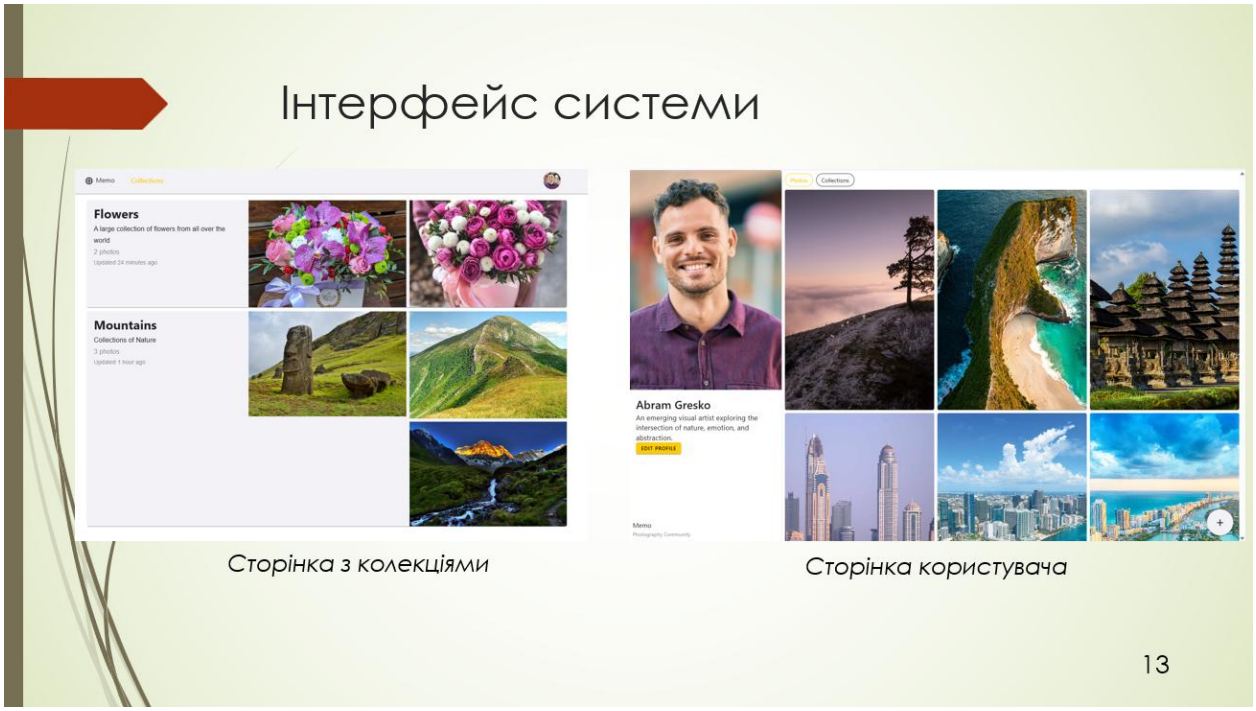


Рисунок В.13 – Слайд 13

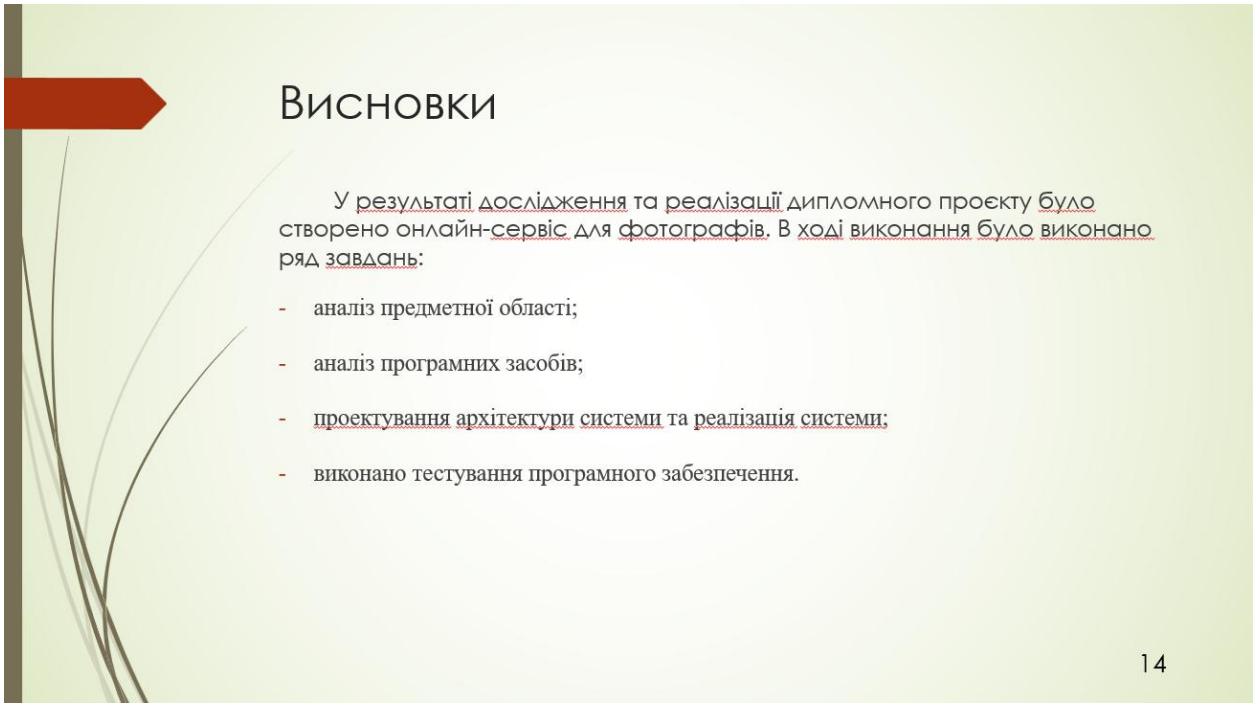


Рисунок В.14 – Слайд 14