Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка методу та засобів реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень»

Виконав: студент IV курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Юдін О. С.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Войтко В. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. ОТ Городецька О. С.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О.Н._____
«06» червня 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Юдіну Олександру Сергійовичу

1. Тема роботи – розробка методу та засобів реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень.

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи: 2 червня 2025 р.

3. Вихідні дані до роботи: середовище розробки – Visual Studio Code; мова розробки – JavaScript; фреймворки – React, Node; операційна система – Windows 8/10/11.

4. Зміст текстової частини: вступ; аналіз стану розвитку програм для реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень та постановка задач розробки; розробка структури, методу та моделі програмного продукту; розробка алгоритмів роботи програми; розробка програмного забезпечення, тестування програмного продукту, висновки, список використаних джерел, додатки.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність розробки; мета, об'єкт та предмет дослідження; постановка задачі; наукова новизна; практична цінність; огляд та аналіз аналогів; структура та метод розробки застосунку; діаграма варіантів використання; модель роботи системи у вигляді діаграми класів та діяльності; алгоритми роботи системи; схеми інтерфейсу; модуль оцінювання статичних зображень; модуль управління обміном статичних зображень; модуль для збирання, обробки та візуалізації статистичних даних; тестування програми; використані технології; функціонал розробленої системи; апробація та публікація результатів роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийня
1-4	Войтко В. В., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.2

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Приміт
1	Аналіз стану розвитку програм для комунікацій в системах телемедицини та постановка задач розробки	14.03.25 – 21.03.25	
2	Розробка методів, моделі та алгоритмів роботи програми	22.03.25 – 04.04.25	
3	Розробка програмного забезпечення	05.04.25 – 03.05.25	
4	Тестування програми	06.05.25 – 17.05.25	
5	Оформлення матеріалів до захисту БКР	20.05.25 – 30.05.25	

Студент _____ **Юдін**
(підпис) (ініціали та пр

Керівник бакалаврської кваліфікаційної роботи _____ **Войтко**
(підпис) (ініціали та пр

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота містить 144 сторінок формату А4, на яких є 78 рисунків і 4 таблиці, а список використаних джерел складається з 40 найменування.

У бакалаврській кваліфікаційній роботі проведено аналіз підходів до візуалізації, аналізу та оцінювання статичних зображень у вебсередовищі. Розглянуто існуючі рішення, визначено їхні переваги та недоліки, а також обґрунтовано доцільність розробки власної вебсистеми для роботи із зображеннями.

У роботі обґрунтовано вибір мови програмування, інструментів розробки та технологій, які були використані при створенні вебзастосунку. Було розроблено повнофункціональну вебсистему для візуалізації, аналізу та оцінювання статичних зображень. Для реалізації фронтенд-частини застосовано мову програмування JavaScript та бібліотеку React. Серверну частину реалізовано за допомогою Node.js. У якості системи керування базами даних використано PostgreSQL. Середовищем розробки слугувало Visual Studio Code.

Отримані результати бакалаврської кваліфікаційної роботи можуть бути використані для впровадження ефективних рішень у сфері аналізу зображень у різних галузях, зокрема в медичних, технічних і наукових системах.

Ключові слова: вебсистема, візуалізація зображень, аналіз зображень, React, Node.js, PostgreSQL, інтерфейс користувача.

ABSTRACT

The bachelor's qualification work comprises 144 A4 pages, including 78 figures and 4 tables. The list of references consists of 40 sources.

The bachelor's qualification work presents an analysis of approaches to the visualization, analysis, and evaluation of static images in a web environment. Existing solutions are reviewed, their advantages and disadvantages are identified, and the rationale for developing a custom web system for working with images is provided.

The work substantiates the choice of programming language, development tools, and technologies used in the creation of the web application. A fully functional web system for visualizing, analyzing, and evaluating static images was developed. The frontend was implemented using the JavaScript programming language and the React library. The backend was developed using Node.js. PostgreSQL was used as the database management system. Visual Studio Code served as the development environment.

The results of the bachelor's qualification work can be applied to implement effective solutions in the field of image analysis across various domains, including medical, technical, and scientific systems.

Keywords: web system, image visualization, image analysis, React, Node.js, PostgreSQL, user interface.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Візуальна інформація стала невід'ємною частиною повсякденного життя. Зображення використовуються в найрізноманітніших сферах: від особистих фотографій у соціальних мережах до ілюстрацій у новинних ресурсах, технічної візуалізації в наукових дослідженнях, медичних діагностичних матеріалів, дизайнерських концептів і візуального супроводу промислових процесів. Такий стрімкий ріст обсягу графічного контенту обумовлює необхідність не лише у зберіганні та перегляді зображень, а й у створенні ефективних засобів їхнього глибокого аналізу та об'єктивного оцінювання. Сучасний користувач очікує не просто бачити зображення, а розуміти його зміст, структуру, технічні параметри, композиційні особливості або ж емоційно-естетичне навантаження [1].

Активний розвиток вебтехнологій у поєднанні з алгоритмами комп'ютерного зору, штучного інтелекту, машинного навчання та новітніми засобами побудови інтерфейсів користувача відкривають широкі перспективи для створення інтелектуальних вебсистем, здатних ефективно обробляти, візуалізувати та аналізувати статичні графічні дані. Такий підхід забезпечує зручний віддалений доступ до функціоналу системи, її адаптивність, інтерактивність, масштабованість і можливість одночасного використання як окремими користувачами, так і цілими організаціями – наприклад, у процесі досліджень, створення архівів, проведення оцінювання або презентації зображень.

Зростання обсягів візуального контенту та підвищені вимоги до якості його аналізу обумовлюють актуальність розробки спеціалізованих програмних рішень. Традиційні способи обробки зображень, які передбачають значну участь людини, вже не відповідають вимогам часу: вони повільні, суб'єктивні та не підходять для роботи з великими обсягами даних. Автоматизовані вебзастосунки дозволяють значно підвищити ефективність цієї роботи, забезпечуючи високу швидкість, точність і зменшення людського фактору [2].

Попри наявність низки готових вебрішень, більшість з них мають істотні обмеження: недостатня глибина аналізу, обмежений функціонал у плані оцінювання, відсутність засобів взаємодії між користувачами (коментарі, голосування, спільна участь у проєктах), а також низький рівень кастомізації під конкретні прикладні завдання. Часто подібні сервіси не підтримують можливості аналітики або категоризації зображень, що унеможливлює гнучке налаштування системи під потреби конкретного користувача.

Саме тому виникає потреба у створенні комплексної вебплатформи, яка б поєднувала засоби інтелектуального аналізу зображень, системи рейтингу, інструменти категоризації та елементи соціальної взаємодії. Такий підхід дозволить не лише автоматизувати процес обробки візуальної інформації, але й значно підвищити її інформативність, залучити аудиторію до обговорення результатів та створити нові підходи до оцінки як технічних, так і естетичних параметрів графічного контенту.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення рівня якості аналізу й оцінювання статичних зображень шляхом розробки методу та засобів реалізації вебсистеми, що дозволить автоматизувати обробку графічного контенту, забезпечити гнучку систему фільтрації та оцінювання візуальних матеріалів.

Відповідно до поставленої мети потрібно вирішити такі завдання:

- здійснити аналіз стану питання та методів розв’язання задачі;
- спроєктувати модель даних та структуру вебзастосунку;
- розробити метод оцінювання якості зображення;
- розробити метод управління обміном статичних даних;
- розробити алгоритм взаємодії користувача з особистим кабінетом;
- розробити алгоритм взаємодії користувача з публікаціями;
- розробити алгоритм оцінювання якості зображення;

– розробити програмне забезпечення модуля оцінювання статичних зображень;

– розробити програмне забезпечення модуля управління обміном статичних зображень;

– розробити програмне забезпечення модуля для збирання, обробки та візуалізації статистичних даних;

– провести тестування програмного застосунку;

– розробити інструкцію користувача та описати системні вимоги програми для візуалізації, аналізу й оцінювання статичних зображень.

Об'єкт дослідження – процес візуалізації, аналізу та оцінювання статичних зображень у вебсередовищі.

Предмет дослідження – методи й засоби реалізації вебсистеми для обробки зображень, що включає механізми аналізу і обміну, взаємодії з користувачами та оцінювання графічного контенту.

Методи дослідження. У процесі дослідження використовувалися такі методи:

– методи побудови компонентів інтерфейсу користувача для створення функціональних і зручних елементів взаємодії з користувачем;

– методи тестування для перевірки коректності роботи програмних модулів, виявлення помилок та забезпечення надійності програмного забезпечення;

– методи організації баз даних для збереження та обробки даних про статичні зображення у локальному сховищі;

– методи моделювання алгоритмів для формалізації логіки обробки даних, візуалізації процесів і оптимізації алгоритмічних рішень;

– методи порівняльного аналізу для оцінювання переваг і недоліків різних підходів за заданими критеріями.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод оцінювання якості зображення як контентної користувацької експертизи, що відрізняється від існуючих аналогів

поетапною інтерактивною структурою оцінювання візуального та текстового контенту через модульний інтерфейс, реалізований на основі React-компонентів і станового менеджера Zustand, що дозволяє підвищити рівень якості аналізу й оцінювання статичних зображень з використанням інтерактивної, адаптивної системи поетапного збору оцінок.

2. Подальшого розвитку набув метод управління обміном статичних зображень, який вирізняється від аналогів інтеграцією механізму обміну даними між публікаціями в режимі реального часу з використанням модального вікна, впровадженням технологій асинхронної обробки функцій (async/await), динамічної маршрутизації (Next.js Router), модульної побудови інтерфейсу з компонентами Modal, UserExchangePostList та використанням станів і умовного рендерингу для керування вибором і підтвердженням обміну інформацією, що забезпечує безпосередню взаємодію користувачів без зайвих запитів і підвищує рівень взаємодії між користувачами.

Практичне значення отриманих результатів полягає у можливості використання розробленого вебзастосунку в різних сферах діяльності, зокрема в освіті, творчості, науці, дизайні та медичній візуалізації, де потрібна робота з великою кількістю статичних зображень. Реалізовані програмні модулі забезпечують взаємодію між користувачами, дозволяють оцінювати, коментувати, фільтрувати та обмінюватися графічним контентом. Крім того, система формує статистику для кожної публікації з подальшою побудовою графіків, що сприяє аналізу активності та ефективності контенту.

Створене програмне забезпечення може бути використане розробниками для впровадження аналогічних систем у таких напрямах, як онлайн-платформи для художніх конкурсів, освітні ресурси, фотобанки або спеціалізовані інформаційні системи для обробки візуальних матеріалів. Система легко адаптується до потреб конкретної галузі, що дозволяє масштабувати її функціонал і підвищити ефективність управління візуальним контентом.

Особистий внесок здобувача. Усі результати, подані в бакалаврській кваліфікаційній роботі, були отримані автором особисто. У науковій праці [3],

опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, блок-схема алгоритму роботи вебсистеми візуалізації, аналізу та оцінювання статичних зображень, розрахунок зведеного коефіцієнта релевантності аналогів.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати бакалаврської кваліфікаційної роботи доповідались на конференції «Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (НТКП ВНТУ – 2025)» та на конференції XV Міжнародна науково-технічна конференція «Інформаційно-комп'ютерні технології» (НТКП ЖДТУ – 2025).

Публікації. Результати дослідження були опубліковані у матеріалах конференції – «Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ – 2025)» [2] та конференції – XV Міжнародній науково-технічній конференції «Інформаційно-комп'ютерні технології» (НТКП ЖДТУ – 2025) [1].

Структура та обсяг БКР. БКР складається зі вступу, чотирьох розділів, висновків, списку літератури, що містить 40 найменувань, 4 додатків. Робота містить 78 ілюстрації, 4 таблиці. Загальний обсяг роботи складає 144 сторінки, основний зміст викладено на 94 сторінках друкованого тексту.

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ ВІЗУАЛІЗАЦІЇ, АНАЛІЗУ Й ОЦІНЮВАННЯ СТАТИЧНИХ ЗОБРАЖЕНЬ ТА ПОСТАНОВКА ЗАДАЧ РОБОТИ

1.1 Аналіз предметної області

Початок еволюції вебсистем для роботи з візуальним контентом пов'язаний із появою перших онлайн-галерей і форумів, які дозволяли користувачам завантажувати зображення без додаткової обробки, описів або сортування. Такі ресурси не передбачали можливості персоналізації, взаємодії між користувачами чи аналітичної обробки даних. Контент зберігався у вигляді статичних зображень, без функцій оцінювання, пошуку або організації за категоріями [3]. Основний функціонал обмежувався лише переглядом і, у деяких випадках, коментуванням.

З розвитком інтернет-технологій та стрімким збільшенням кількості візуального контенту виникла потреба у створенні більш динамічних і функціонально насичених платформ. Нові покоління вебсистем почали не лише демонструвати зображення, а й пропонувати інструменти для їх аналізу, структурування та оцінювання. Користувачі отримали змогу створювати персональні профілі, завантажувати та редагувати зображення, додавати описи, ключові слова та надсилати роботи до публічних галерей і на конкурси.

Важливим кроком у розвитку таких систем стала інтеграція механізмів оцінювання, що дозволяє формувати рейтинги, виявляти популярні роботи та підтримувати мотивацію спільноти. Окрім цього, функціонал сучасних вебплатформ включає додавання зображень у вибране, фільтрацію, пошук за тегами або текстовим описом, що суттєво покращує навігацію й загальне користувацьке враження [3].

Особливої популярності останніми роками набули рішення, що базуються на штучному інтелекті та машинному навчанні. Ці технології дозволяють автоматично класифікувати контент, аналізувати вміст зображень, визначати візуальну схожість, генерувати відповідні теги й формувати персоналізовані рекомендації. За даними звіту «The Business Research Company» [4] обсяг

глобального ринку платформ для обміну зображеннями зросте з $5,69 млрд у 2025 році до $7,68 млрд у 2029 році (див. рисунок 1.1), що свідчить про стійкий попит на такі рішення.

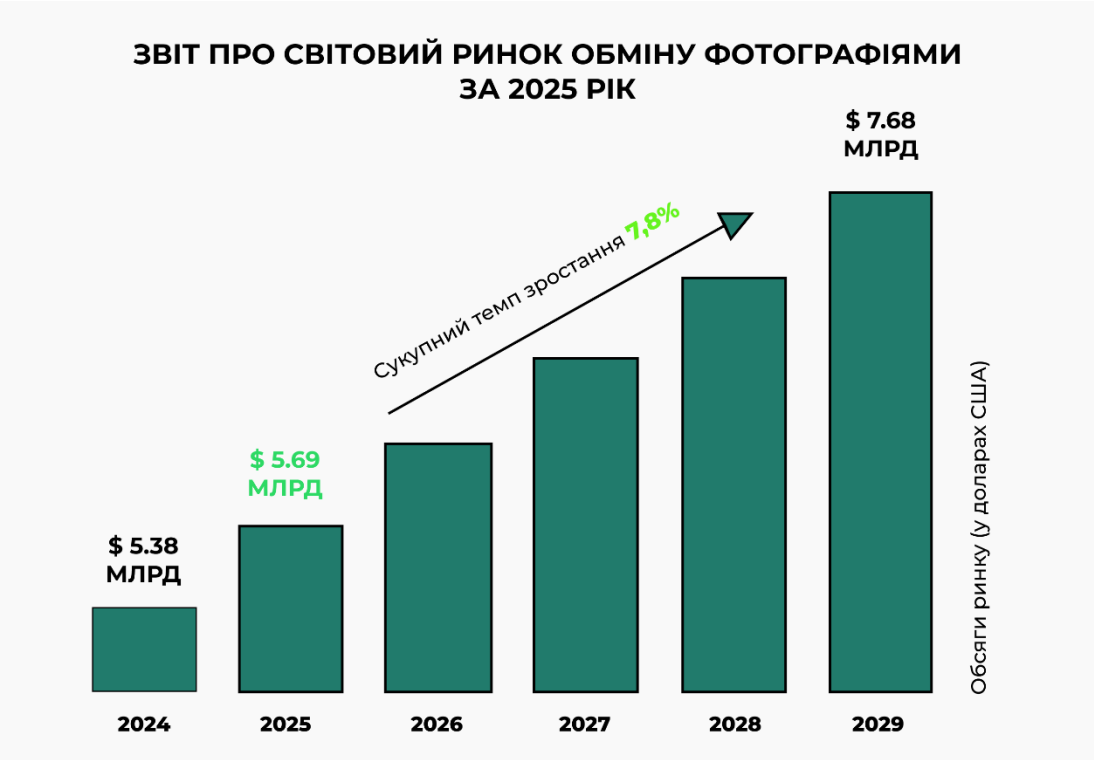

Рисунок 1.1 – Звіт «The Business Research Company» про світовий ринок обміну фотографіями [4]

Крім того, згідно з аналітикою дизайнерського агенства «Crackitt», візуальний контент у соціальних мережах демонструє значно вищу ефективність: публікації з фото отримують на 2,3 рази більше взаємодій у Facebook, ніж текстові пости, а 69% користувачів Instagram вважають візуальний контент більш привабливим, ніж текстовий [5].

Ще однією важливою тенденцією стала персоналізація контенту – платформи навчаються аналізувати поведінку користувачів і пропонувати зображення, авторів або події, які відповідають інтересам конкретної аудиторії. Це забезпечується завдяки ефективному збору та обробці даних про активність користувачів.

Сьогодні вебплатформи для обробки та аналізу зображень стали складними цифровими екосистемами, які включають функціонал збереження, аналітики, соціальної взаємодії, обробки даних та адаптації інтерфейсу до потреб користувача. Успішна реалізація таких систем вимагає не лише технічної надійності, а й врахування етичних, соціальних та безпекових аспектів, зокрема захисту особистих даних і створення зрозумілого та привабливого інтерфейсу.

Таким чином, трансформація вебсистем для роботи із зображеннями охопила шлях від примітивного відображення вмісту до гнучких, інтелектуальних платформ із розширеними можливостями взаємодії, аналізу та персоналізації. Це відкриває нові горизонти для творчої діяльності, взаємного оцінювання та активного залучення користувачів у цифрове середовище.

Отже, в умовах сучасних вимог особливої актуальності набуває розробка вебзастосунку, орієнтованого на візуалізацію, оцінювання та аналіз статичних зображень. Це дозволяє не лише оптимізувати повторювані дії, але й сприяє більш глибокій взаємодії користувачів із платформою, підвищуючи якість обслуговування та цінність цифрового контенту.

1.2 Порівняльний аналіз аналогів

З розвитком сучасних вебтехнологій зростає потреба у створенні спеціалізованих систем, орієнтованих на обробку, аналіз та ефективну взаємодію з візуальним контентом. Особливої актуальності набувають вебсистеми, які дозволяють автоматизувати процеси оцінювання графічних матеріалів, організовувати цифровий контент та забезпечувати зручну взаємодію між користувачами. Такі платформи мають не лише реалізовувати функціональність завантаження зображень, а й надавати можливості для створення постів, персоналізації, соціальної активності, участі у конкурсах та аналітичного опрацювання контенту [2].

Крім того, сучасні вимоги користувачів диктують необхідність впровадження інтелектуальних технологій, таких як машинне навчання та штучний інтелект, для забезпечення автоматизованого аналізу зображень, формування

персоналізованих рекомендацій та покращення навігації системою. Значну роль відіграє й інтерактивність інтерфейсу – користувачі очікують інтуїтивно зрозумілого дизайну, швидкого доступу до функцій, можливості взаємодії з іншими учасниками платформи через коментарі, оцінки або повідомлення.

У сучасному цифровому середовищі вже функціонують численні вебзастосунки, які частково реалізують функції аналізу, оцінювання та організації графічного контенту. Серед них можна виокремити найбільш популярні платформи, що надають користувачам інструменти для роботи із зображеннями, створення постів та взаємодії з контентом. Зокрема, розглянуто платформи: ImgPost, Unsplash, Pexels, Pixabay.

ImgPost – це вебплатформа, що дозволяє користувачам завантажувати зображення та обмінюватися ними. Основна функціональність включає можливість додавання короткого опису до кожного посту, перегляд стрічки новин інших користувачів та базову систему лайків. Однак сервіс не підтримує розширені можливості для коментування, додавання тегів або проведення детального аналізу зображень [6]. Інтерфейс сервісу зображено на рисунку 1.2.

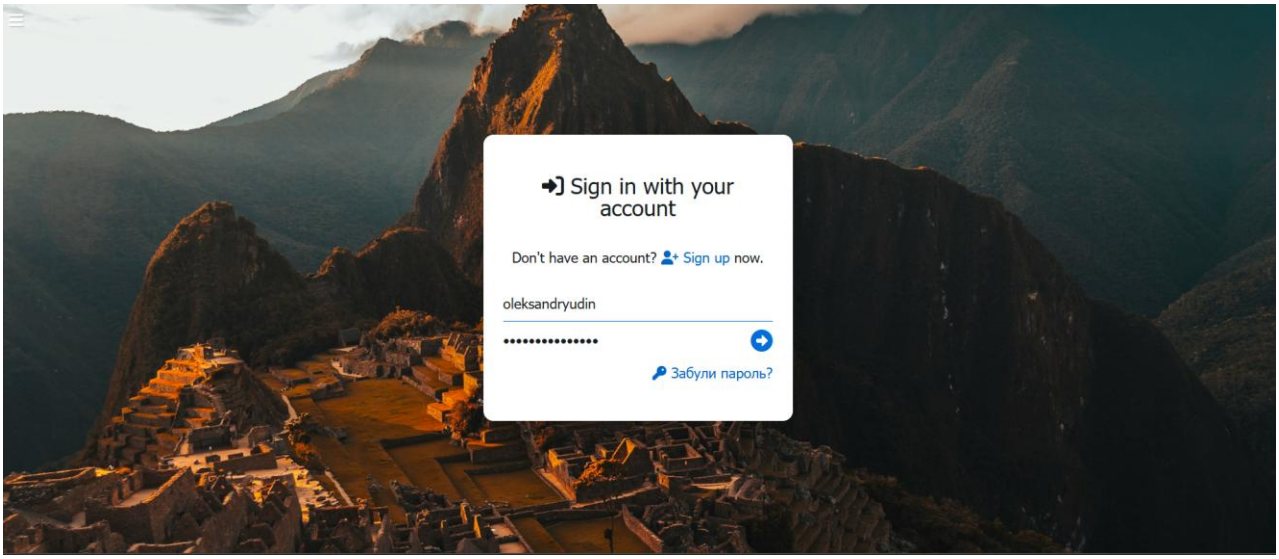

Рисунок 1.2 – Форма входу вебресурсу ImgPost

Unsplash – це популярний ресурс для фотографів і дизайнерів, що дозволяє користувачам завантажувати, переглядати та безкоштовно використовувати

високоякісні зображення. Платформа містить систему тегів для зручного пошуку, але відсутня функція коментування або оцінювання зображень [7]. Також Unsplash не має конкурсного оцінювання або можливості створення персонального профілю з усіма публікаціями користувача (див. рисунок 1.3).

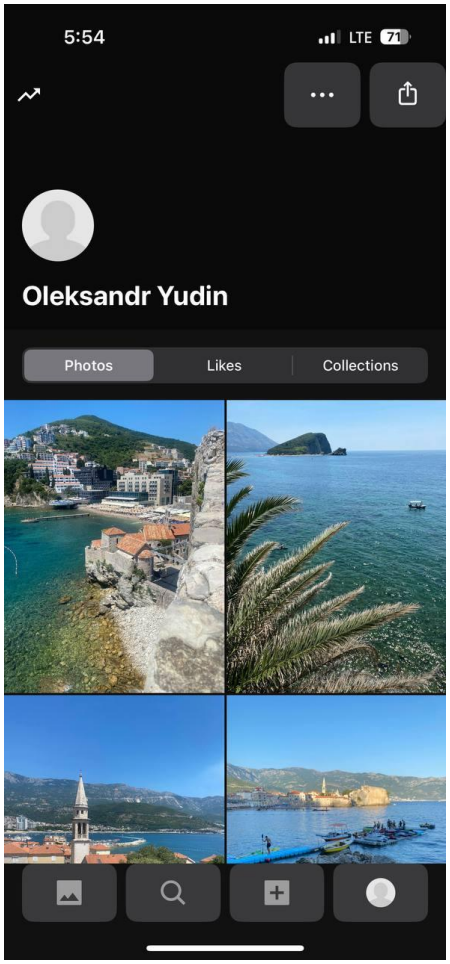

Рисунок 1.3 – Вкладка завантажених особистих фото сервісу Unsplash

Pexels – це ще один популярний фотосток, який дозволяє завантажувати та безкоштовно використовувати зображення як для особистих, так і для комерційних цілей. Платформа відзначається великою бібліотекою якісного контенту, що постійно оновлюється, а також простим і зручним інтерфейсом для перегляду та завантаження фото [8].

На відміну від Unsplash**,** Pexels має базову систему лайків, що дозволяє користувачам висловлювати вподобання щодо зображень. Вебресурс не підтримує

функціональність коментування, не пропонує розширених інструментів для взаємодії між користувачами та аналітичного опрацювання зображень. Інтерфейс сервісу зображено на рисунку 1.3.

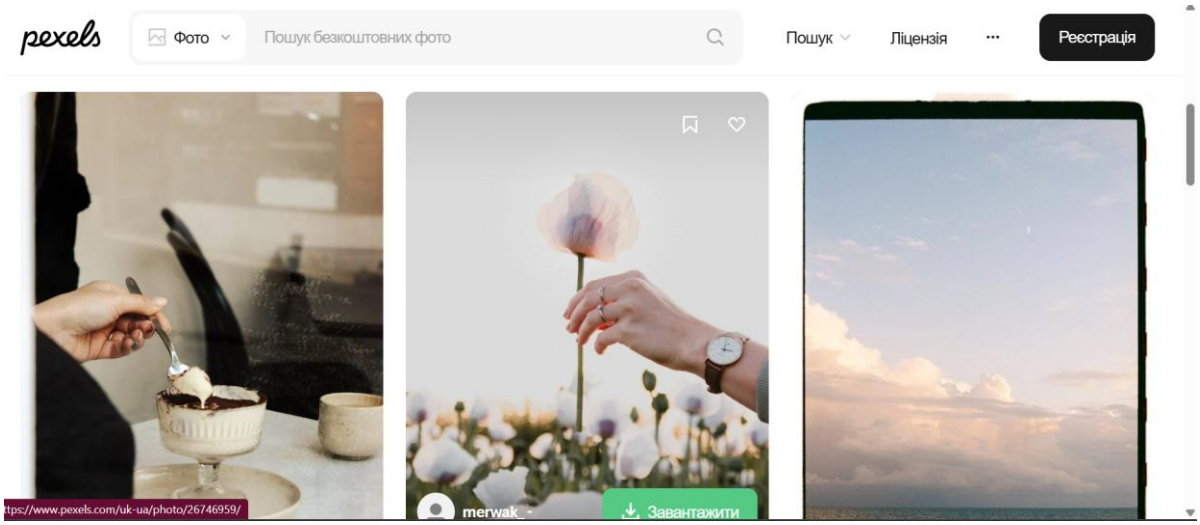

Рисунок 1.3 – Головна сторінка вебресурсу Pexels

Pixabay – це ресурс, що містить безкоштовні та платні зображення, відео та ілюстрації (див. рисунок 1.4). Користувачі можуть переглядати та завантажувати контент, використовуючи систему тегів та пошук за ключовими словами. Однак на платформі відсутня можливість коментування чи оцінювання зображень, а також обмежена функціональність для соціальної взаємодії між користувачами [9].

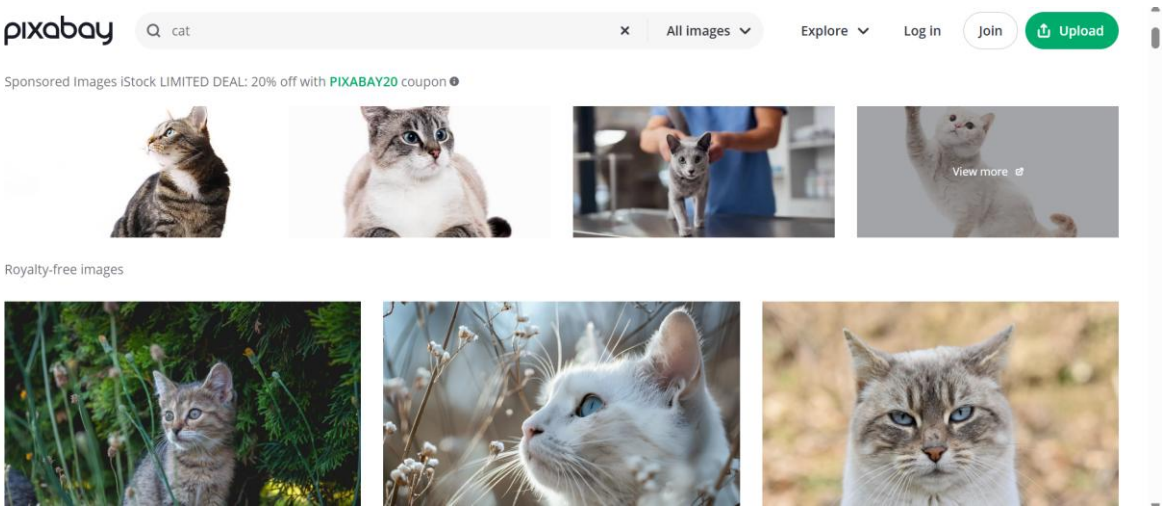

Рисунок 1.4 – Відображення фото за запитом «cat» вебресурсу Pixabay

З метою наочного зіставлення проаналізованих вебресурсів їхні функціональні можливості, переваги та обмеження були узагальнені у таблиці порівняння (див. таблицю 1.1). Такий підхід дозволяє виокремити ключові характеристики кожного рішення, визначити їх сильні сторони та виявити недоліки. Проведення аналізу аналогічних систем є важливим етапом, оскільки дає змогу запозичити успішні практики, уникнути типових помилок інших проєктів і виявити потенціал для вдосконалення. Це, у свою чергу, сприяє створенню більш досконалого, зручного для користувачів і конкурентоспроможного продукту, що відповідає сучасним вимогам.

Таблиця 1.1 – Порівняльна характеристика аналогів

Функціональні можливості	ImgPost	Unsplash	Pexels	Pixabay	Власна розробка
Додавання коментарів	0	0	0	0	1
Можливість додавати теги до постів	0	1	1	1	1
Система фільтрації контенту	0	1	1	1	1
Персональний профіль з усіма публікаціями	1	1	1	1	1
Конкурсне оцінювання зображень	0	0	1	0	1
Обмін постами між користувачами	0	0	0	0	1
Сумарний коефіцієнт	1	3	4	3	6

Аналізуючи таблицю 1.1, власна розробка має вищий сумарний коефіцієнт за розглянутими критеріями у порівнянні з аналогами Unsplash та Pixabay на 50% (100% - 3/6*100% = 50%), у порівнянні з додатком Pexels – на 33% (100% - 4/6*100% = 33%), та у порівнянні з ImgPost – на 83% (100 - 1/6*100% = 83%). Аналіз свідчить про значну перевагу власної розробки, яка надає користувачам більше

можливостей для взаємодії, оцінювання та персоналізації контенту [2]. Такі додаткові функції, як можливість додавання коментарів, створення персональних профілів, конкурсне оцінювання зображень, а також можливість обміну постами між користувачами, роблять платформу більш привабливою та інтуїтивно зрозумілою для користувачів.

1.3 Аналіз методів розв'язання задачі

Розробка вебзастосунку для обміну, візуалізації та оцінювання зображень передбачає використання різних методів і підходів, включаючи вибір архітектурного стилю, мови програмування, фреймворків для фронтенду та бекенду, а також методів зберігання та обробки мультимедійних даних. Розглянемо деякі з найпоширеніших підходів до реалізації подібних систем [10]:

1. Монолітна архітектура. У цьому випадку всі компоненти системи (інтерфейс, логіка, база даних) реалізуються в одному проєкті. Такий підхід простий у реалізації на початковому етапі, проте обмежує масштабованість та ускладнює підтримку у разі розширення функціональності, особливо коли передбачається обробка великої кількості зображень та взаємодія користувачів.

2. Клієнт-серверна архітектура. Найпоширеніший варіант для сучасних вебдодатків. Застосунок розділяється на клієнтську частину, яка відповідає за інтерфейс користувача (фронтенд), та серверну частину, що опрацьовує запити, керує збереженням і обробкою даних (бекенд). Це дозволяє ефективно розподіляти навантаження, спрощує обслуговування системи та дає змогу легко реалізувати авторизацію, фільтрацію, оцінювання та персоналізований функціонал.

3. Мікросервісна архітектура. Цей підхід передбачає створення окремих незалежних сервісів для кожної функціональної частини (наприклад, окремий сервіс для завантаження зображень, оцінювання, управління користувачами тощо). Мікросервіси добре масштабуються, проте вимагають більшої складності в налаштуванні взаємодії між компонентами та підвищеної уваги до безпеки й продуктивності [10].

У власній розробці було вирішено використати клієнт-серверну архітектуру, яка оптимально підходить для реалізації вебплатформи з інтерактивним інтерфейсом користувача, гнучким бекендом та інтеграцією з базою даних. Для реалізації клієнтської частини обрано React, який підтримує компонентну структуру та дозволяє зручно реалізовувати SPA-функціональність. На сервері використано Node.js з фреймворком Express, що забезпечує обробку REST-запитів та підключення до бази даних PostgreSQl.

Розглянемо також підходи до організації процесу взаємодії з візуальним контентом у вебзастосунках.

Одним з основних методів є ручне завантаження зображень, що дозволяє користувачам додавати власні зображення через спеціальну форму. Такий функціонал зазвичай включає попередню валідацію типу файлу та його розміру, після чого зображення зберігається на сервері, а шлях до нього фіксується у базі даних (наприклад, у PostgreSQL), що забезпечує упорядковане зберігання та подальшу обробку контенту.

Системи взаємодії з візуальним контентом також можуть містити механізми пошуку та каталогізації. Зазвичай користувачам надається можливість самостійно додавати теги до зображень під час публікації, а надалі фільтрувати контент за категоріями, тегами або назвами. Функція пошуку може бути реалізована з підтримкою часткового збігу, регулярних виразів і незалежністю від регістру символів, що дозволяє ефективно знаходити потрібні зображення навіть за неповною інформацією [11].

Ще одним поширеним елементом є функція оцінювання зображень. Системи такого типу можуть реалізовувати рейтингування контенту за кількома критеріями, з можливістю підрахунку середніх балів та подальшим формуванням рекомендацій. Оцінки можуть зберігатися в окремих таблицях бази даних з використанням зовнішніх ключів для зв’язку з користувачами та зображеннями.

Крім того, подібні вебзастосунки зазвичай передбачають наявність особистого кабінету, який містить можливості для перегляду власних публікацій, редагування описів, видалення зображень, а також перегляду історії оцінювання.

Для цього можуть створюватися окремі інтерфейсні компоненти та серверні маршрути.

Щоб забезпечити захищений доступ до персоналізованого функціоналу, системи зазвичай реалізують механізми авторизації та реєстрації. До їх складу може входити валідація вхідних даних, хешування паролів за допомогою сучасних бібліотек, а також збереження токенів (наприклад, JWT) для подальшої перевірки доступу до захищених ресурсів.

Загалом, подібна система може бути орієнтована як на звичайних користувачів, зацікавлених у зберіганні та обміні зображеннями, так і на організаторів тематичних фотоконкурсів. У власному вебдодатку планується впровадження більшості з описаних підходів з метою підвищення доступності та якості взаємодії з продуктом.

1.4 Постановка задач роботи

У сучасному цифровому середовищі зростає попит на сервіси, які не лише дозволяють користувачам обмінюватися візуальним контентом, а й активно взаємодіяти з ним за допомогою інструментів оцінювання, аналітики, фільтрації, пошуку та персоналізованих рекомендацій. Проте більшість наявних платформ мають обмежену функціональність у цих аспектах: вони не надають гнучких можливостей налаштування, часто позбавлені глибокої статистичної аналітики й не забезпечують зручного багатокритеріального пошуку зображень. Така ситуація ускладнює користування подібними системами, знижує рівень взаємодії користувачів із контентом і стримує розвиток креативних спільнот.

У зв’язку з цим постала актуальна потреба у створенні веборієнтованої системи, що дозволить візуалізувати, аналізувати та оцінювати статичні зображення з урахуванням сучасних технологічних вимог і потреб користувачів. Така система повинна підтримувати широкий набір функцій, які сприятимуть зручній, інтуїтивно зрозумілій взаємодії з контентом, а також залученню активної аудиторії.

У межах бакалаврської кваліфікаційної роботи передбачається реалізація комплексу завдань, що охоплюють такі ключові напрями:

– здійснити аналіз стану питання та методів розв’язання задачі;

– спроектувати модель даних та структуру вебзастосунку;

– розробити метод оцінювання якості зображення;

– розробити метод управління обміном статичних зображень;

– розробити алгоритм взаємодії користувача з особистим кабінетом;

– розробити алгоритм взаємодії користувача з публікаціями;

– розробити алгоритм оцінювання якості зображення;

– розробити програмне забезпечення модуля оцінювання статичних зображень;

– розробити програмне забезпечення модуля управління обміном статичних даних;

– розробити програмне забезпечення модуля для збирання, обробки та візуалізації статистичних даних;

– провести тестування програмного застосунку;

– розробити інструкцію користувача та описати системні вимоги програми для візуалізації, аналізу й оцінювання статичних зображень.

Реалізація цих завдань дозволить створити функціональну і зручну систему для обміну, оцінки та візуалізації статичних зображень, що забезпечить комфортну взаємодію користувачів, дозволить залучати нових учасників та підвищить ефективність зворотного зв'язку на платформі. Очікується, що впровадження даної системи значно полегшить процес оцінки та обміну зображеннями, а також підвищить активність користувачів та покращить загальний досвід взаємодії з платформою.

1.5 Висновки

Аналіз сучасного стану вебсистем для роботи з візуальним контентом засвідчує стрімку еволюцію – від елементарного перегляду зображень до створення

багатофункціональних цифрових платформ із розширеними можливостями аналізу, оцінювання та персоналізації. У процесі розвитку ці платформи стали невід'ємною частиною цифрового середовища, яке охоплює не лише технічні аспекти, але й соціальні, етичні та безпекові вимоги. Особливу роль у цьому відіграє інтеграція технологій штучного інтелекту та машинного навчання, які забезпечують автоматизовану обробку зображень, формування релевантних тегів, рекомендацій і підвищення загальної ефективності взаємодії користувача з системою.

Порівняльний аналіз існуючих рішень (зокрема платформ ImgPost, Unsplash, Pexels, Pixabay) свідчить, що попри наявність певного базового функціоналу, більшість із них не забезпечують комплексної системи фільтрації, гнучкого аналізу, соціальної взаємодії та персоналізованого оцінювання контенту. Вони переважно орієнтовані на зберігання зображень і базову взаємодію (лайки, перегляди), залишаючи поза увагою аналітичну складову, яка є надзвичайно важливою в умовах інформаційного перевантаження.

У зв'язку з цим актуальною і доцільною є розробка методу та засобів реалізації вебсистеми, що дозволить автоматизувати обробку графічного контенту, забезпечити гнучку систему фільтрації та оцінювання візуальних матеріалів. Така система має задовольняти потреби сучасного користувача у швидкому доступі до якісного контенту, надаючи водночас інструменти для аналітики, персоналізації та активної взаємодії з платформою.

Отже, подальші етапи дослідження будуть зосереджені на проектуванні архітектури такої вебсистеми, виборі оптимальних технологій її реалізації та формуванні ефективної моделі оцінювання статичних зображень. Реалізація цієї мети сприятиме підвищенню рівня якості контенту та створенню інноваційного середовища для користувачів.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛЕЙ ТА АЛГОРИТМІВ РОБОТИ СИСТЕМИ

2.1 Аналіз вхідних та вихідних даних системи

У процесі розробки вебдодатка для візуалізації, аналізу й оцінювання статичних зображень важливим етапом є визначення структури вхідних та вихідних даних для кожного з функціональних модулів. Це дозволяє забезпечити узгодженість у взаємодії між користувачем та системою, а також оптимізувати обробку інформації на стороні сервера. Система включає чотири основні функціональні модулі: реєстрації та авторизації, обміну публікаціями, оцінювання зображень і статистики. Кожен із цих модулів виконує чітко окреслену роль і працює з відповідним набором вхідних та вихідних даних.

Модуль реєстрації та авторизації відповідає за створення та обслуговування облікових записів користувачів. Вхідними даними для цього модуля є інформація, яку користувач вводить у форму: ім’я, електронна адреса, пароль і підтвердження пароля. Додатково система отримує IP-адресу клієнта та дату й час спроби входу, що використовується для забезпечення безпеки. У процесі обробки здійснюється валідація введених даних, перевірка існування облікового запису, хешування пароля та створення сесійного токена. Як вихідні дані модуль формує повідомлення про результат реєстрації або авторизації, повертає сесію доступу, дані поточного користувача, а також повідомлення про помилки у разі некоректного входу.

Модуль обміну публікаціями реалізує механізм обміну речами між користувачами. На вхід модуль отримує ідентифікатори публікацій, обраних для обміну, а також сигнали підтвердження або скасування дії. Внутрішня логіка модуля включає перевірку актуальності обраних публікацій і фіксацію згоди обох сторін на обмін. У результаті система генерує повідомлення про успішний або відхилений обмін, оновлює статус відповідних публікацій (наприклад, «обміняно» або «недоступно») та формує список отриманих публікацій, які згодом можуть бути переглянуті користувачем у його профілі.

Модуль оцінювання зображень забезпечує функціональність взаємодії користувачів із вмістом публікацій шляхом виставлення оцінок, залишення коментарів або натискання позначок «подобається». На вхід цей модуль отримує ідентифікатор користувача, вибране зображення, обрані критерії оцінювання, текст коментаря та дату з часом взаємодії. Під час обробки система перевіряє, чи здійснював користувач попередню оцінку, і у випадку нового зворотного зв’язку зберігає його, оновлюючи статистику. Як вихідні дані модуль формує загальну оцінку зображення, історію оцінювання, що може бути використана для подальшого аналітичного аналізу, та виводить результати безпосередньо під відповідною публікацією.

Модуль статистики акумулює та аналізує дані щодо активності користувачів. До вхідних параметрів належать ідентифікатор користувача, фільтри перегляду статистики (наприклад, за датою або кількістю взаємодій), а також вибір конкретної публікації. У процесі обробки модуль рахує загальну кількість публікацій, аналізує частоту оцінювання, кількість переглядів, лайків і коментарів та генерує рейтинг користувача. Результатом роботи модуля є формування списку публікацій зі статистичними показниками, визначення позиції користувача в загальному рейтингу, а також повідомлення про відповідність умовам для участі в конкурсах.

Таким чином, кожен з модулів виконує визначені функції, що базуються на чітко структурованих вхідних та вихідних даних, що в сукупності забезпечує ефективну та прозору роботу вебдодатка.

2.2 Розробка структури програмного застосунку

Створення структури програмного застосунку є важливим етапом розробки будь-якої інформаційної системи, оскільки вона визначає логіку взаємодії між компонентами, функціональність та шляхи користувацької взаємодії. Якісно спроєктована структура дозволяє уникнути плутанини в коді, покращує масштабованість системи, а також полегшує її подальший супровід і розвиток. Одним із найбільш ефективних способів відображення структури та

функціональних можливостей програмного застосунку є використання UML-діаграм (Unified Modeling Language), зокрема, діаграм варіантів використання.

Для візуального представлення структури застосунку та взаємозв'язків між різними його частинами було створено діаграму варіантів використання, що зображена на рисунку 2.1. Діаграма варіантів використання (англ. Use Case Diagram) – це один із видів UML-діаграм, який описує функціональність системи з точки зору взаємодії користувача (акторів) із системою [12].

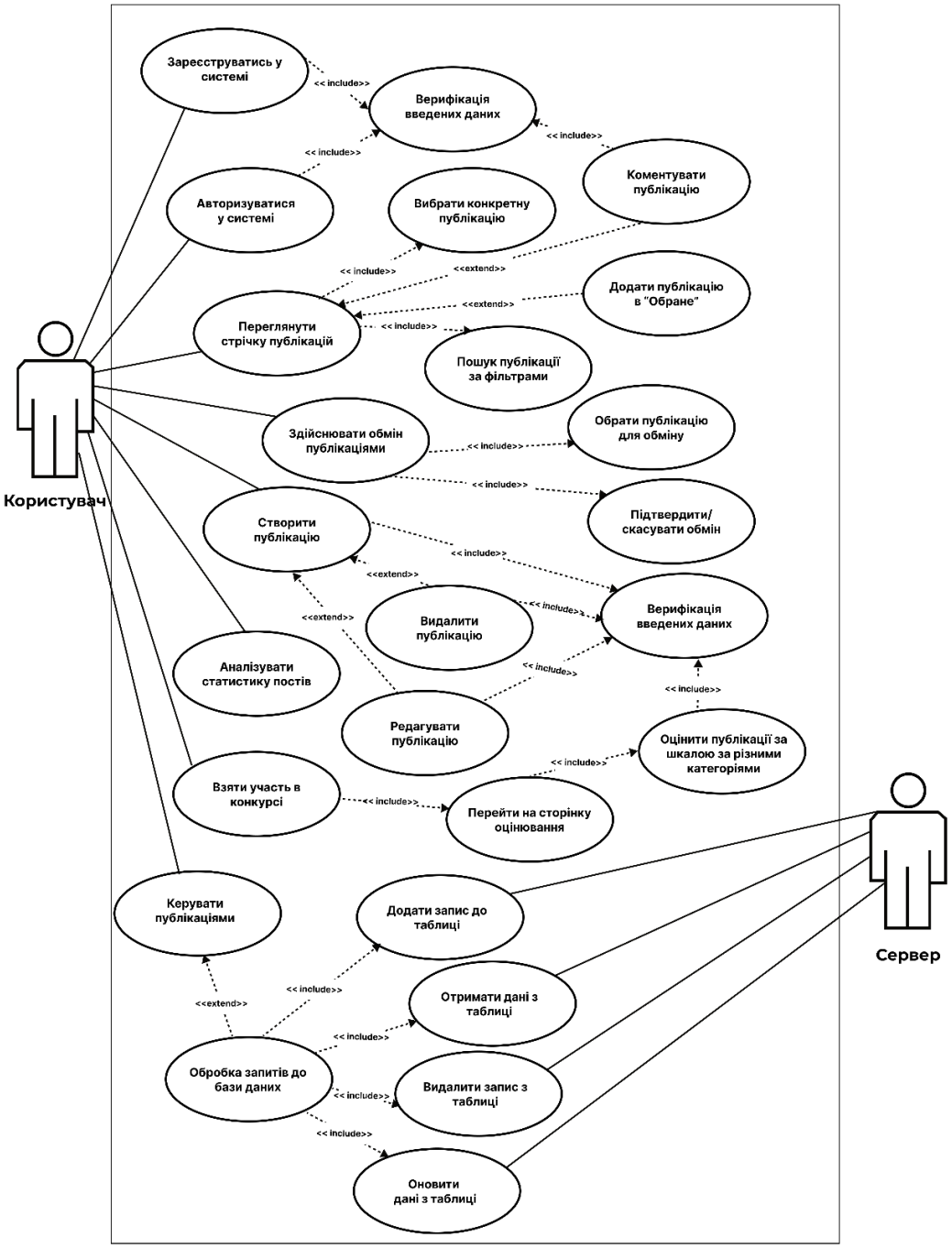

Рисунок 2.1 – Діаграма варіантів використання

Отже, діаграма варіантів використання, представлена на рисунку, демонструє загальну структуру програмного застосунку, відображаючи взаємодію між двома основними суб'єктами – користувачем та сервером. Вона охоплює ключові функціональні можливості, які надає система, а також логіку виконання цих можливостей із залученням внутрішніх процесів і перевірок.

У центрі уваги перебуває користувач, якому доступна широка низка дій. Початкові етапи роботи з системою включають реєстрацію та авторизацію, обидві з яких обов’язково супроводжуються верифікацією введених даних. Це свідчить про важливість перевірки достовірності інформації з боку системи ще на етапі доступу до функціоналу. Після входу до системи користувач має змогу переглядати стрічку публікацій, що є базовим способом взаємодії з контентом. Із цієї стрічки можна обрати конкретну публікацію, після чого відкриваються додаткові можливості: коментування, додавання до обраного, а також аналіз змісту з перспективи подальшої взаємодії – наприклад, обміну.

Обмін публікаціями є розширеним сценарієм використання. Для його реалізації користувач спершу повинен обрати публікацію для обміну, а потім підтвердити або скасувати цю дію. Процес також включає верифікацію введених даних, що гарантує правильність і безпеку виконання обміну. Для зручності реалізовано функцію пошуку публікацій за фільтрами, яка інтегрується в загальну систему перегляду контенту. Створення публікацій – ще один важливий функціонал, що дозволяє користувачам наповнювати систему новим контентом. Створені публікації згодом можуть бути відредаговані або видалені, що забезпечує гнучкість у керуванні власним контентом. Додатково користувачі мають змогу аналізувати статистику своїх постів, що допомагає відстежувати ефективність взаємодії з аудиторією.

Окремо передбачено можливість участі в конкурсах, що супроводжується переходом на сторінку оцінювання. Тут користувачі можуть оцінювати публікації за різними категоріями, що знову ж таки включає верифікацію введених даних. Такий механізм дозволяє будувати систему чесної та об'єктивної оцінки, важливої для визначення результатів конкурсів і підвищення взаємодії між учасниками.

На стороні сервера відображено ключові дії, пов’язані з обробкою даних: додавання, оновлення, отримання та видалення записів з таблиць бази даних. Усі ці дії згруповані під єдиним процесом – обробка запитів до бази даних. Сервер також виконує верифікацію даних, отриманих від користувача, що є частиною кожної важливої операції – як у момент обміну, так і при створенні публікацій чи участі в конкурсах. Таким чином, сервер відіграє роль не лише сховища, але й логічного посередника, який забезпечує безпеку, точність і цілісність даних, що передаються та змінюються у системі.

Уся діаграма побудована з використанням зв’язків типу <<include>> та <<extend>>, що демонструє логічну взаємозалежність функціональних частин. Наприклад, процес створення публікації автоматично включає її перевірку, тоді як можливість коментування є розширенням функції вибору конкретної публікації. Це забезпечує гнучкість системи та можливість масштабування функціоналу без порушення основної логіки застосунку [13].

Проєктування інтерфейсу користувача є одним із ключових етапів розробки сучасних вебдодатків. Від того, наскільки зручною, зрозумілою та послідовною буде взаємодія користувача із системою, значною мірою залежить загальний успіх проєкту. Інтерфейс є безпосередньою точкою контакту між людиною та цифровим продуктом, тому його якість визначає ефективність виконання цільових дій, рівень задоволеності користувачів та загальне сприйняття сервісу.

Під час проєктування інтерфейсу особлива увага приділялася аспектам UX (user experience), що включають логічну структуру елементів, інтуїтивну навігацію, зрозумілі візуальні підказки та передбачувану поведінку системи у відповідь на дії користувача. Було важливо забезпечити комфортну взаємодію як для нових, так і для досвідчених користувачів [14].

З метою візуалізації логіки взаємодії та раннього тестування зручності користування ще до початку реалізації було створено прототип інтерфейсу. Цей прототип дозволив оцінити загальний вигляд майбутнього вебдодатка, протестувати ключові сценарії користувацької поведінки та вчасно виявити недоліки у структурі або дизайні. Прототип включав основні екрани системи –

форму реєстрації та входу, головну сторінку з публікаціями, модуль обміну, інтерфейс оцінювання та блок статистики. Прототип головного вікна зображена на рисунку 2.2.

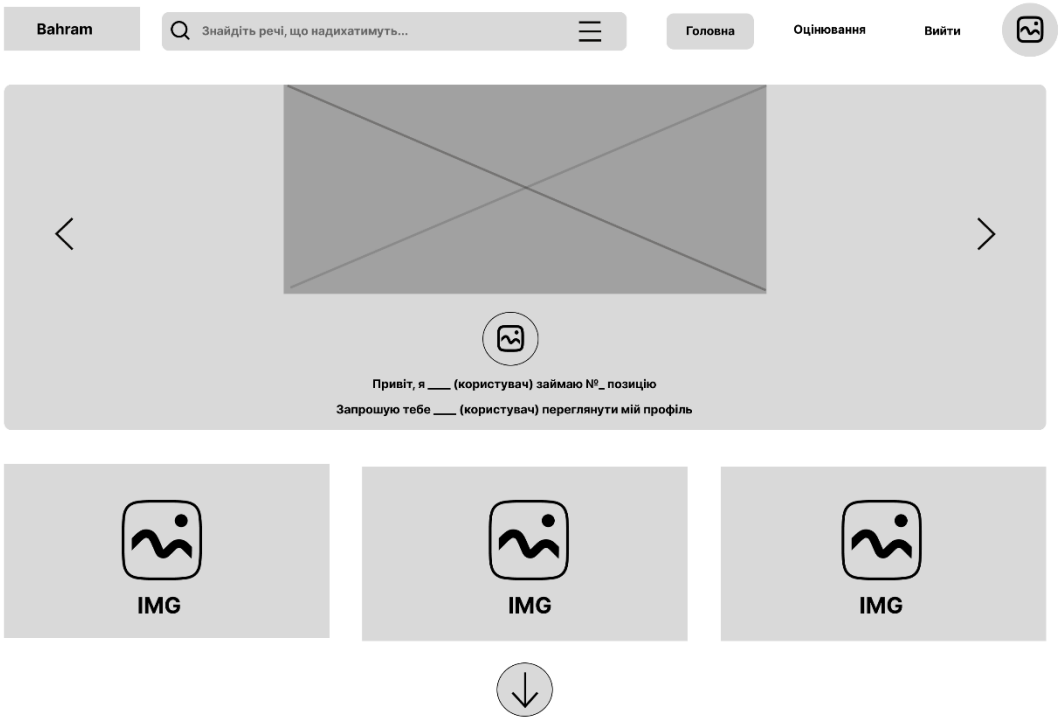

Рисунок 2.2 – Прототип головної сторінки вебзастсоунку

Представлений прототип головної сторінки є макетом користувацького інтерфейсу вебзастосунку, що має на меті забезпечити зручну навігацію, швидкий доступ до основних функцій та візуально привабливу презентацію контенту. У верхній частині сторінки розташована панель навігації, яка включає логотип (Bahram), поле для пошуку з підказкою «Знайдіть речі, що надихатимуть...», а також іконку меню для додаткових налаштувань. Праворуч розміщені кнопки «Головна», «Оцінювання», «Вийти», а також іконка профілю, що веде до особистого кабінету користувача.

Центральну частину сторінки займає великий слайдер з можливістю перемикання між елементами за допомогою стрілок вліво та вправо. У цьому блоці також передбачено аватар користувача та коротке привітання, що свідчить про

наявність рейтингової системи. Нижче подано запрошення до взаємодії, що підкреслює соціальну спрямованість платформи та стимулює до перегляду профілів інших учасників.

Під основним візуальним блоком розміщено три графічні елементи з підписами «IMG», що символізують публікації, рекомендовані матеріали або актуальні дописи. Вони виконані у форматі плиток однакового розміру, що створює симетричну та гармонійну композицію. У нижній частині екрану розміщено іконку стрілки вниз, яка слугує для прокручування сторінки вниз.

Зважаючи на сучасні тенденції користування вебзастосунками, особливо серед молодої аудиторії, адаптивність інтерфейсу під мобільні пристрої є критично важливою. Користувачі все частіше взаємодіють із платформами саме через смартфони, тому зручність перегляду та навігації на малих екранах повинна бути на такому ж рівні, як і в десктопній версії [15].

У мобільній версії головної сторінки зберігаються ключові елементи, проте вони адаптовані під вертикальне розміщення контенту, характерне для мобільних екранів. Навігаційне меню трансформується у випадаючий або бічний список, активується за допомогою «бургер»-іконки, що дозволяє зберегти простір. Пошуковий рядок, кнопки переходу та головний слайдер масштабуються відповідно до ширини екрану, зберігаючи при цьому читабельність і зручність натискання елементів.

Особливу увагу в мобільній версії приділено швидкому доступу до особистого кабінету користувача. Іконка профілю, що традиційно розміщується в правому верхньому куті, зберігається й тут, однак завдяки інтуїтивному дизайну стає ще помітнішою. Це полегшує навігацію й забезпечує швидкий перехід до персональної сторінки, де користувач може переглянути свої публікації, рейтинг, статистику чи брати участь у конкурсах. Такий підхід не лише підвищує зручність взаємодії, а й робить платформу більш привабливою для постійного користування з мобільних пристроїв.

Саме тому далі представлено прототип адаптивної мобільної версії сторінки з акцентом на зручний доступ до особистого кабінету (див. рисунок 2.3).

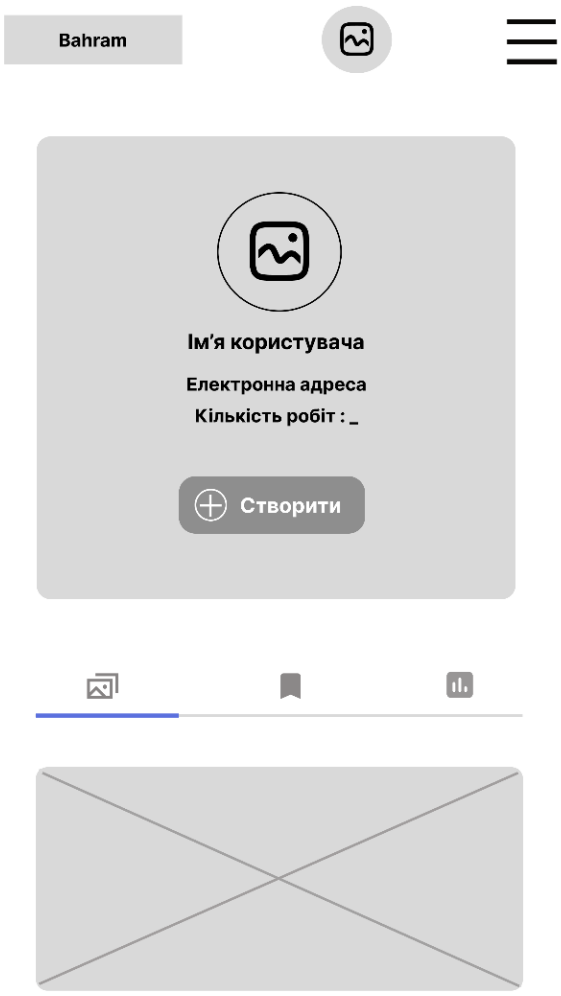

Рисунок 2.3 – Прототип особистого кабінету вебзастосунку

Представлений прототип демонструє адаптивний мобільний інтерфейс особистого кабінету користувача в межах вебзастосунку. Дизайн виконаний у мінімалістичному стилі з чітким поділом на функціональні блоки, що забезпечує зручність перегляду з екранів смартфонів. Інтерфейс оптимізований для вертикального скролу, що є природним способом взаємодії в мобільному середовищі.

У верхній частині екрану розташовано логотип платформи «Bahram», кнопку переходу до головної сторінки, іконку профілю, яка виконує роль навігаційного маркера поточної сторінки, а також меню у вигляді бургер-іконки. Така структура дозволяє швидко перемикатись між основними розділами, не перевантажуючи інтерфейс зайвими елементами.

Основна частина екрана представлена інформаційною карткою користувача. Вона містить аватар (іконка зображення), ім’я користувача, електронну адресу та кількість опублікованих робіт. Нижче розташовано кнопку «Створити», яка відкриває форму створення нової публікації. Візуальна ієрархія добре витримана: увага користувача фокусується на особистих даних і основній дії, що стимулює до активності в системі.

У нижній частині інтерфейсу розміщена вкладкова навігація з трьома піктограмами: галерея публікацій, обране та статистика. Перша вкладка (галерея) активна, про що свідчить синя лінія під іконкою. Це забезпечує інтуїтивне перемикання між типами контенту без потреби перезавантаження сторінки.

Завершує інтерфейс блок попереднього перегляду вмісту – велике зображення, яке, відображає всі завантажені публікації користувача. Такий підхід дозволяє швидко ознайомитись з результатами своєї активності на платформі та водночас зберігає естетичну складову інтерфейсу.

Загалом, прототип демонструє грамотне адаптивне рішення, яке зберігає функціональність десктопної версії, але у спрощеній і доступній для мобільних пристроїв формі.

2.3 Розробка методу оцінювання якості зображення

Оцінювання зображень є ключовим етапом для залучення користувачів до участі у процесі контентної експертизи, тому метод його реалізації має бути інтуїтивним, поетапним та функціональним. Користувачі мають змогу оцінювати різні аспекти зображень через вебзастосунок. Метод оцінювання забезпечує збір, збереження та оновлення оцінок, а також підтримує інтерактивний інтерфейс для поетапного введення балів. Це дозволяє ефективно аналізувати якість контенту, відображати статистику та підтримувати взаємодію користувачів із системою [16].

Метод оцінювання зображень включає такий функціонал:

1. Користувач відкриває модальне вікно оцінювання зображення в якому за допомогою динамічного рендерингу (через useMemo) відображаються окремі компоненти оцінювання.

2. Користувачу відображаються окремі компоненти оцінювання: `GeneralPostInformationStep`, `TitlePostInformationStep`, `TextPostInformationStep`, `TagsPostInformationStep`, `ImagePostInformationStep`, `ResultsPostInformationStep`.

2. Користувач послідовно проходить кілька кроків оцінювання, заповнюючи бали за різні категорії: загальна інформація, заголовок, текст, теги, зображення.

3. На кожному кроці через форму вводиться оцінка за відповідним полем.

4. Після завершення останнього кроку `results` користувач підтверджує свою оцінку кнопкою «Надіслати».

5. Запускається функція `handleSubmitForm`, яка закриває модальне вікно та викликає метод `postEvaluationVote (markId, marks)` для відправки оцінок на сервер.

6. Функція `postEvaluationVote (markId, marks)` виконує POST-запит на API `/marks/{id}`, передаючи введені оцінки.

7. Сервер приймає оцінки, зберігає або оновлює їх у базі даних, після чого повертає оновлений список оцінюваних учасників.

8. Отримані дані оновлюються у стані клієнта через `useEvaluateStore`, що забезпечує актуалізацію інтерфейсу.

9. У разі помилки користувач отримує відповідне повідомлення через alert.

10. Якщо зображення ще не додане або оцінене, компонент `ImagePostInformationStep` блокується через `isDisabled={!!marks.image}`, що забезпечує контроль послідовності оцінювання.

11. Також реалізовані методи для додавання фотографій до оцінювання (`postArticleForEvaluation`) та видалення їх із оцінювання (`removeArticleFromEvaluation`), що підтримує актуальність списку матеріалів для оцінки.

Технічна реалізація методу оцінювання базується на використанні станового менеджера `zustand` для централізованого зберігання та управління станом оцінок. Взаємодія з сервером здійснюється через axios із налаштованим інтерсептором. На клієнтському боці реалізовано React-компонент `MarathonVotePostModal`, який відповідає за поетапний інтерфейс оцінювання з використанням хуків `useState`,

`useCallback` та `useMemo`. Кожен крок оцінювання відображається окремим компонентом, що спрощує управління складною формою.

Серверна частина обробляє запити, зберігає оцінки у відповідній колекції бази даних і повертає актуальні дані. Важливо, що метод підтримує асинхронність, обробку помилок та оновлення стану у реальному часі, що підвищує користувацький досвід та забезпечує надійність системи оцінювання.

Завдяки цьому методу платформа отримує гнучкий інструмент для централізованої оцінки візуального та текстового контенту. Це дозволяє:

– підтримувати якість публікацій;

– запобігати публікації невідповідного контенту;

– покращувати враження користувачів;

– формувати базу перевірених матеріалів для подальшого використання в інформаційній діяльності;

– забезпечити прозорість та контрольованість процесу модерації.

Отже, даний метод оцінювання забезпечує повний цикл взаємодії: від введення даних на клієнті до збереження та оновлення результатів на сервері, що сприяє підвищенню якості контенту та залученню активних користувачів до оцінювання.

2.4 Розробка методу управління обміном статичних зображень

Обмін статичними зображеннями між користувачами є ключовою функціональністю платформи, що спрямована на підвищення взаємодії між учасниками спільноти. Методи реалізації обміну мають бути інтуїтивно зрозумілими, безпечними і прозорими, адже вони безпосередньо впливають на довіру до платформи та якість користувацького досвіду.

Функціонал обміну дозволяє користувачам обмінюватись зображеннями (представленими у вигляді постів), які вони публікують. Обмін ініціюється з інтерфейсу користувача, де обирається власне зображення та зображення партнера для взаємного обміну. Такий процес сприяє налагодженню комунікації, розвитку мережі учасників та ефективному управлінню контентом платформи.

Методи обміну зображеннями реалізує такий функціонал:

1. Авторизований користувач переходить до модального вікна обміну.

2. Відбувається завантаження списку зображень ініціатора через `fetchInitiatorPosts(id)` та партнера через `fetchPartnerPosts(id)` (з API `/articles/feed/:id`). Метод fetchInitiatorPosts відповідає за отримання списку постів ініціатора обміну. Метод fetchPartnerPosts відповідає за отримання списку постів партнера обміну. Це дозволяє користувачу обрати одну зі своїх статей для обміну з партнером.

3. Користувач обирає одне зображення зі списку своїх публікацій та одне зі списку партнера.

4. У стані `exchangePostSlug` зберігаються обрані ідентифікатори зображень.

5. При натисканні кнопки «Підтвердити» викликається метод `handleSubmitExchange`.

6. Метод `claimDeal` зберігає обрані зображення та формує запит обміну:

– користувач обирає по одному посту від себе та партнера;

– натискає кнопку «Підтвердити» у модальному вікні;

– викликається метод claimDeal з параметрами initiatorArticleSlug та partnerArticleSlug;

– метод надсилає POST-запит до API /articles/exchange з відповідними параметрами;

– у разі помилки виводиться повідомлення через alert.

7. У клієнтському запиті викликається POST-запит до API `/articles/exchange` з параметрами `ownArticleSlug` та `partnerArticleSlug`.

8. На сервері метод `exchangeArticle(user, partnerArticleSlug, ownArticleSlug)`:

– знаходить відповідні об'єкти зображень за slug;

– виконує обмін авторами між двома статтями;

– зберігає оновлення у базі даних;

– повертає оновлену статтю користувача.

9. Успішний обмін спричиняє оновлення інтерфейсу та перенаправлення користувача на сторінку профілю `/me`, де він може переглянути оновлений список

своїх постів. Це покращує користувацький досвід, дозволяючи одразу побачити результат обміну та переконатися у його успішності.

Технічна реалізація методу обміну включає взаємодію між фронтендом (React), клієнтським сховищем стану (Zustand) та серверним API (Node.js + Express). Компонент `ExchangePostModal` відповідає за відображення списку зображень, обробку вибору, логіку підтвердження та очищення. Обрані slug'и статей зберігаються у стані та використовуються для запиту.

На серверній частині метод `exchangeArticle` реалізовано як асинхронну функцію в сервісі, яка оперує об'єктами постів у базі даних. Обмін відбувається через зміну поля `author` у двох статтях і подальше збереження змін. Це гарантує прозорість, а також унеможливлює несанкціонований доступ до статей сторонніх користувачів.

Для захисту реалізовано механізм авторизації, що перевіряє користувача, котрий ініціює обмін. У випадку помилок реалізовано базову обробку через `try/catch` та інформування користувача про проблему через `alert`, що забезпечує прозорий зворотний зв'язок.

Таким чином, метод обміну забезпечує швидкий, інтуїтивний і безпечний механізм обміну статичними зображеннями, що формує довіру до платформи й забезпечує новий рівень взаємодії між користувачами.

2.5 Розробка моделей роботи системи

Розробка моделі функціонування вебзастосунку для візуалізації, аналізу та оцінювання статичних зображень вимагає побудови відповідної UML-діаграми активності. Така діаграма є інструментом, який дозволяє описати логіку взаємодії користувача із системою у вигляді послідовності дій. Вона слугує своєрідною картою, яка демонструє шлях користувача під час використання вебзастосунку: від першого входу до виконання конкретних функцій, таких як перегляд, оцінювання чи створення публікацій.

UML-діаграма активності (Unified Modeling Language Activity Diagram) – це графічне подання алгоритму або потоку робіт у системі. Вона дозволяє моделювати логіку поведінки системи чи окремого процесу, зосереджуючи увагу на переходах між станами внаслідок дій користувача або внутрішніх подій. Завдяки використанню символів, таких як дії, рішення, початкові й кінцеві стани, UML-діаграма активності допомагає зрозуміти, як саме працює система, які є сценарії її використання та де можуть виникати розгалуження або цикли [17].

На діаграмі діяльності (див. рисунок 2.4) зображено взаємодію користувача з вебзастосунком починаючи з моменту відкриття платформи. Користувач спершу потрапляє на сторінку, де система перевіряє, чи має він обліковий запис. Якщо ні – здійснюється реєстрація, інакше – авторизація. Після цього ще раз перевіряється, чи введено коректні дані, і якщо все в порядку, користувач потрапляє на головну сторінку.

З головної сторінки доступні три основні напрямки дій: стрічка публікацій, сторінка оцінювання та особистий кабінет. У стрічці публікацій користувач може шукати публікації за фільтрами, обирати їх, коментувати, натискати кнопку «Обмін», щоб обмінятись публікаціями з іншими користувачами, погоджувати або скасовувати обмін, а також переглядати отримані публікації. На сторінці оцінювання доступна можливість вибору публікацій для оцінювання, після чого можна оцінювати за різними критеріями й переглядати результати.

Особистий кабінет користувача дає змогу створювати публікації, виходити з облікового запису, переглядати свої пости, аналізувати статистику по ним, брати участь у контролі якості, переходити на сторінку оцінювання, а також редагувати чи видаляти власні публікації. Під час створення публікації користувач заповнює її заголовок, опис, завантажує зображення, вводить кількість тегів, дозволяє ексклюзивність публікації та зрештою підтверджує її створення.

У розробленій UML-діаграмі активності важливо зазначити, що вихід із системи можливий на будь-якому етапі взаємодії з вебзастосунком. Незалежно від того, чи користувач переглядає стрічку публікацій, оцінює зображення, створює нову публікацію або аналізує власну статистику, система завжди передбачає опцію

завершення сеансу. Це рішення реалізується для забезпечення гнучкості та зручності користувача, а також підвищення рівня безпеки, оскільки дає змогу негайно припинити роботу з обліковим записом у будь-який момент.

Таким чином, незалежно від того, на якій саме сторінці чи в якому процесі перебуває користувач, функціонал виходу із системи залишається доступним у будь-який момент.

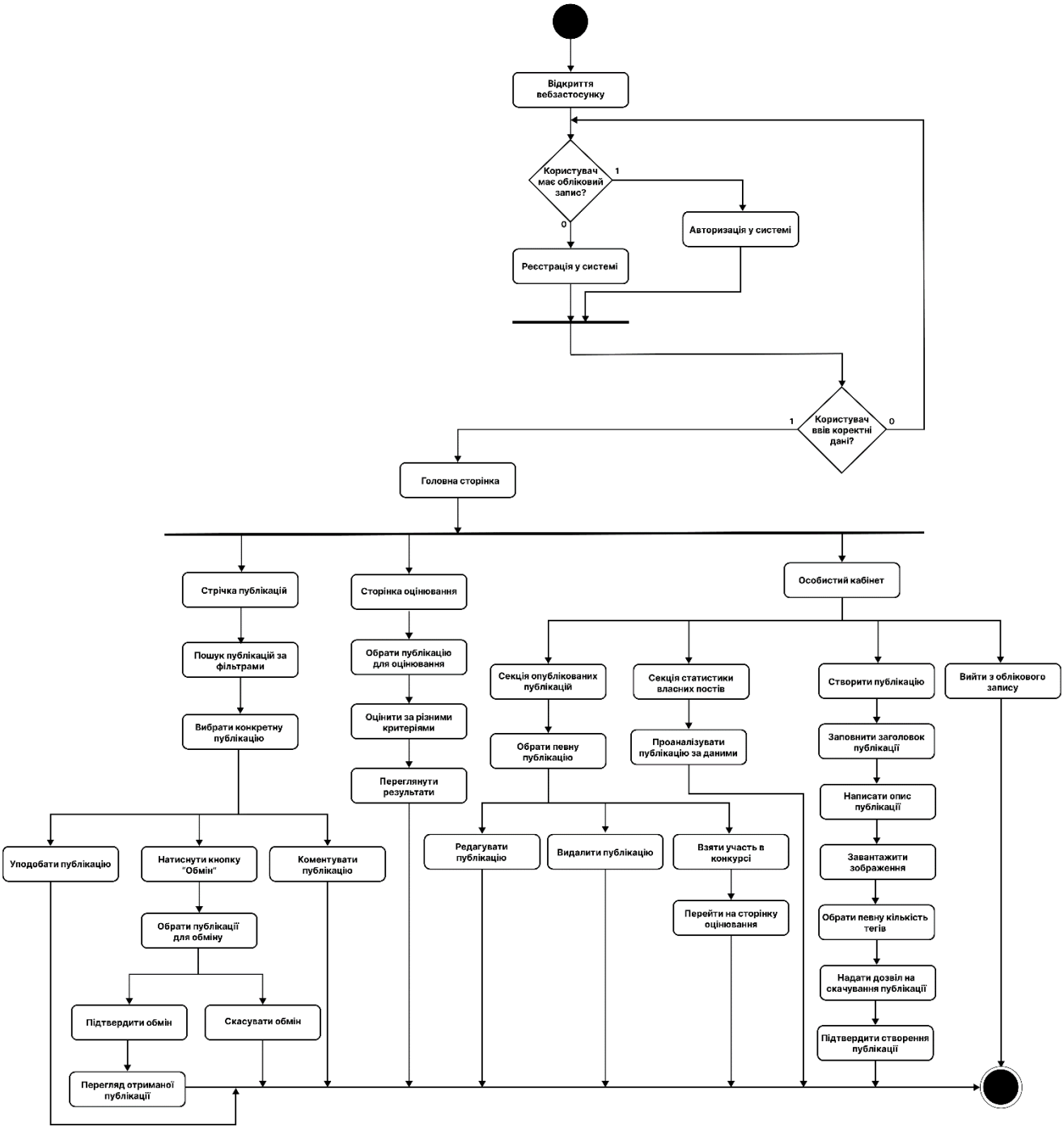

Рисунок 2.4 – Модель роботи системи у вигляді діаграми діяльності системи

Уся діаграма ілюструє логічну структуру дій, яка дозволяє повністю охопити сценарії взаємодії користувача із системою, забезпечуючи зрозумілу та узгоджену модель поведінки, яку можна використати як основу для реалізації функціоналу вебзастосунку.

Щоб краще зрозуміти логіку функціонування системи, її структуру та взаємозв'язки між основними об'єктами, було вирішено побудувати модуль роботи системи у вигляді діаграми класів UML (Unified Modeling Language)**.** Такий підхід дозволяє візуалізувати ключові сутності, які існують у системі, а також їхні атрибути, методи та типи зв'язків між ними. Діаграма класів є ефективним інструментом для проєктування архітектури програмного забезпечення ще до початку написання коду, що допомагає уникнути помилок на ранніх етапах розробки (див. рисунок 2.5).

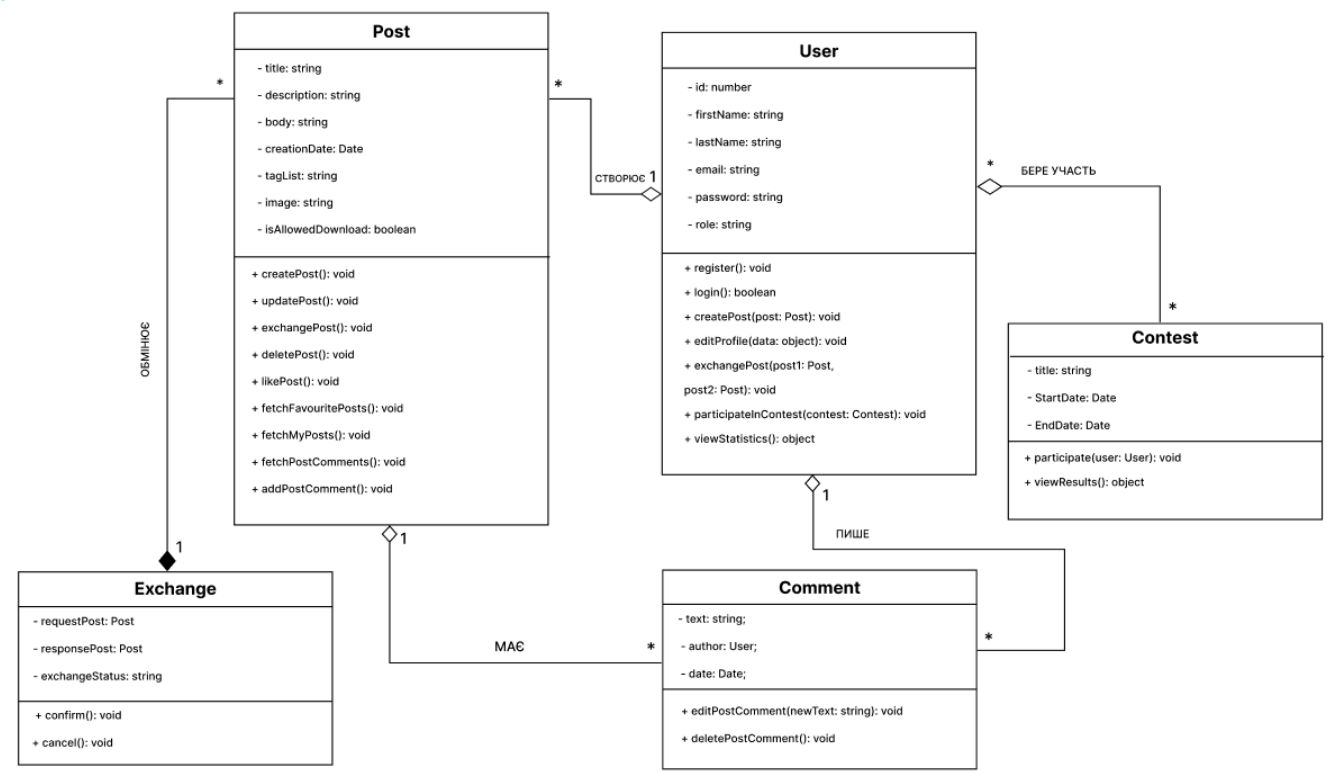

Рисунок 2.5 – Модель роботи системи у вигляді діагарми класів

У центрі діаграми розташований клас User, який є базовим суб'єктом системи. Користувач має стандартні атрибути, такі як id, firstName, lastName, email,

password та role, що дозволяє ідентифікувати його, керувати доступами і розмежувати функціонал. Методи register(), login() та editProfile() вказують на основні дії, доступні користувачеві. Зв’язки від цього класу демонструють, що саме користувач створює пости (створює), бере участь у конкурсах (бере участь), обмінює пости та пише коментарі. Типи зв’язків тут – агрегація, що правильно вказує на слабку залежність між об’єктами.

Клас Post представляє окрему одиницю контенту, яку користувач створює. Він має властивості, притаманні типовим публікаціям: title, description, body, creationDate, tagList, image тощо. Методи посту охоплюють CRUD-операції та взаємодію: створення, оновлення, видалення, лайки, коментування, а також обмін постами. Зв’язок з класом User виконано через агрегацію, що коректно вказує, що пост не може існувати без прив’язки до автора. Також пости можуть бути учасниками обміну (Exchange) – цей зв’язок реалізовано за допомогою композиції, оскільки Exchange не може існувати без обох постів – request і response [16].

Клас Exchange реалізує функціонал обміну постами. Він має два атрибути – requestPost і responsePost, а також статус обміну. Це показує, що в системі є механізм двосторонньої взаємодії між публікаціями. Зв’язок композиції правильно вказує, що обмін залежить від існування конкретних постів і не має сенсу без них. Метод confirm() і cancel() дає можливість реалізувати зміну стану цього об'єкта.

Клас Comment моделює коментарі, які користувачі можуть залишати під постами. У нього є текст, автор і дата – базовий набір атрибутів. Зв’язок із класами User і Post є мультиплікативним: один пост може мати багато коментарів, як і один користувач може написати багато коментарів. Зв’язки типу агрегація тут обґрунтовані, адже коментарі залежать від наявності постів і авторів, але можуть існувати незалежно в базі, наприклад, якщо автор був видалений.

Клас Contest описує конкурси, до яких можуть долучатися користувачі. Має атрибути title, startDate і endDate. Зв’язок із User позначений агрегацією і є багато до багатьох, що логічно – один користувач може брати участь у кількох конкурсах, як і один конкурс може мати багато учасників. Методи participate() та viewResults() логічно розширюють можливості взаємодії [16].

Узагальнюючи, модель є добре структурованою, логічно побудованою, з правильним використанням типів зв’язків: агрегації, композиції, і мультиплікаційних позначень. Кожен клас має чітко виражену відповідальність, а зв’язки між ними – правильне семантичне навантаження. Така діаграма дозволяє зрозуміти, як елементи системи взаємодіють, і слугує шаблоном для реалізації системи в коді.

2.6 Розробка алгоритмів роботи системи

Розробка алгоритмів відіграє вирішальну роль у формуванні логіки роботи програмної системи. Від того, як побудовано алгоритмічну основу, залежить здатність системи своєчасно та коректно реагувати на дії користувача, обробляти дані та координувати взаємодію окремих модулів. Добре спроєктовані алгоритми дозволяють досягти узгодженості в роботі компонентів, уникнути помилок у виконанні операцій та забезпечити стабільність функціонування системи в різних сценаріях її використання [19].

Для забезпечення ефективної взаємодії користувача з функціональністю особистого кабінету у вебсистемі візуалізації, аналізу й оцінювання статичних зображень реалізовано чіткий алгоритм дій, який охоплює увесь цикл: від автентифікації до управління власними публікаціями та статистикою. Основна мета – забезпечити інтуїтивний інтерфейс із широким набором можливостей, реалізованих на сучасному технологічному стеку (див. рисунок 2.6).

Після завантаження сторінки користувачу пропонується ввести логін. Система перевіряє його наявність у базі даних PostgreSQL [20]. У разі відсутності логіна запускається модуль реєстрації нового користувача, який реалізовано на серверній частині за допомогою Node.js із використанням авторизації на основі токенів. Якщо ж логін існує, користувач переходить до введення пароля. Після верифікації пароля або успішного входу – користувач потрапляє до особистого кабінету, що реалізований у середовищі React з використанням бібліотеки Zustang для управління станом сесії [21].

Усередині особистого кабінету користувач має можливість переглядати власні публікації з результатами аналізу зображень, які можуть містити інформацію про тип зображення, параметри обробки. Для взаємодії з публікаціями передбачені функції вибору, редагування, видалення або подання заявки на участь у конкурсах. Ці дії доступні через інтерфейс, стилізований з використанням SCSS [22], і синхронізовані із сервером через запити бібліотеки Axios.

Крім того, користувач має змогу переглядати статистику щодо переглядів, лайків, коментарів або завантажень власних публікацій. Для побудови графіків і динамічних візуалізацій використано бібліотеку Chart.js, що дозволяє наочно відображати тренди та аналітику взаємодії з опублікованим контентом [23].

Якщо ж користувач бажає створити нову публікацію, йому надається інтерфейс заповнення даних, який включає опис зображення, параметри завантаження фото, обробки та збереження результатів. Додатково доступне додавання тегів, що дозволяє автоматизовано класифікувати вміст для подальшого фільтрування та пошуку.

Перед публікацією користувач визначає, чи надавати дозвіл на завантаження зображення іншими користувачами, після чого приймає рішення щодо остаточного розміщення публікації. У разі надання дозволу, зображення стає доступним для скачування, що забезпечує гнучкість в управлінні контентом. Після підтвердження користувачем публікація автоматично розміщується на платформі, а всі зміни фіксуються в системі. Усі публікації автоматично зберігаються в профілі користувача та стають доступними для подальшого статистичного аналізу, що дозволяє відстежувати популярність контенту, його взаємодію з іншими користувачами та ефективність. Це дає змогу системі генерувати детальні звіти та графіки, які допомагають користувачам оцінювати ефективність їхніх публікацій і оптимізувати подальшу взаємодію з аудиторією.

На завершальному етапі користувач може завершити сесію, натиснувши кнопку «Вийти». Після цього система автоматично здійснює вихід із особистого кабінету, зберігаючи всі необхідні зміни та оновлення, які були зроблені під час сеансу. Користувач перенаправляється на головну сторінку або форму авторизації,

де може повторно увійти до свого акаунту. Вихід з особистого кабінету гарантує, що персональна інформація користувача буде захищена, а доступ до акаунту заблокований, поки не буде здійснено повторну авторизацію.

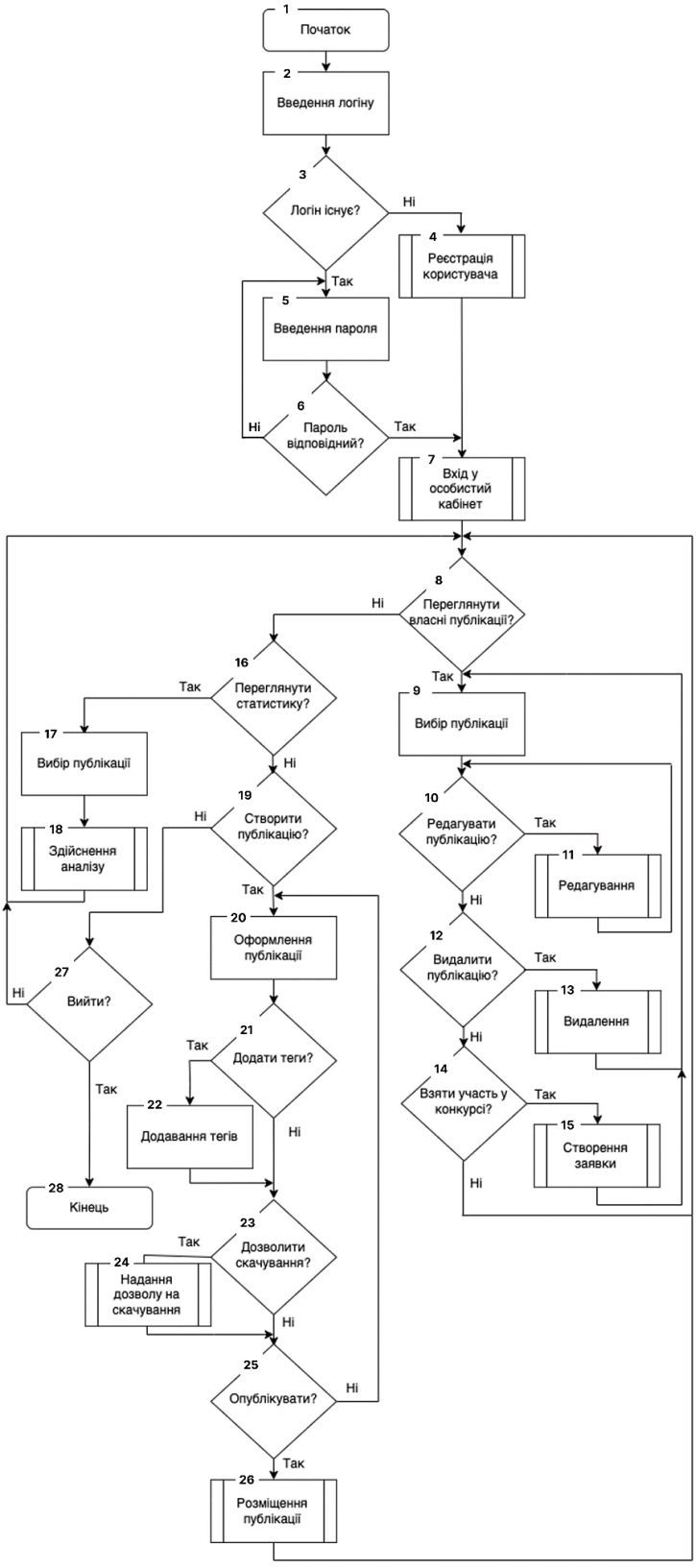

Рисунок 2.6 – Алгоритм взаємодії користувача з особистим кабінетом

Таким чином, розроблений алгоритм забезпечує повноцінну взаємодію користувача з особистим кабінетом, дозволяє керувати візуалізаційними публікаціями, переглядати аналітику, брати участь у конкурсах, а також гнучко налаштовувати доступ до опублікованого контенту.

З метою забезпечення зручної та ефективної взаємодії користувачів із платформою, призначеною для візуалізації, аналізу та оцінювання статичних зображень, було розроблено базовий алгоритм (див. рисунок 2.7), який охоплює основні дії: завантаження контенту, його перегляд, взаємодію з публікаціями, а також можливість залишати лайки, коментарі чи ділитися зображеннями. Алгоритм спрямований на формування зрозумілого та зручного механізму взаємодії між користувачами платформи.

На початковому етапі відбувається ініціація сесії користувача та завантаження стрічки публікацій. Цей процес реалізовано за допомогою API-запитів до серверної частини, побудованої на Node.js, з використанням бібліотеки axios на фронтенді, що забезпечує асинхронну обробку запитів і швидке завантаження контенту.

Наступним кроком є можливість задати фільтри, що дозволяє користувачу налаштувати перегляд відповідно до своїх інтересів. Для цього використано стан-контейнер Zustand, який забезпечує збереження та оновлення фільтраційних параметрів у глобальному стані додатку [21].

Після цього користувач переходить до перегляду та вибору конкретної публікації. У цьому місці реалізовано інтерактивний інтерфейс на базі React із адаптивним дизайном (SCSS), що дозволяє переглядати деталі публікації з мобільних і десктопних пристроїв. Після перегляду користувачеві пропонується можливість вподобати публікацію. Якщо обрано цю опцію, активується функція лайку, що надсилає POST-запит до серверної частини [22]. На рівні бази даних відбувається інкрементування лічильника вподобань для відповідної публікації.

У випадку, якщо користувач бажає залишити коментар, він переходить до поля введення, після чого система публікує коментар. Для збереження коментарів використано окрему таблицю в базі даних, пов’язану з відповідною публікацією за

зовнішнім ключем. Додатково реалізовано функціонал обміну публікаціями. У випадку позитивного вибору користувач обирає власну публікацію для обміну, після чого підтверджує дію. Якщо підтвердження успішне, система здійснює обмін публікаціями. Цей процес включає перевірку прав доступу до публікацій, унеможливлюючи несанкціоновану зміну інформації.

На завершальному етапі користувачу пропонується вийти з процесу взаємодії. У разі підтвердження виходу сесія завершуються, і користувача повертають до головного інтерфейсу або сторінки авторизації.

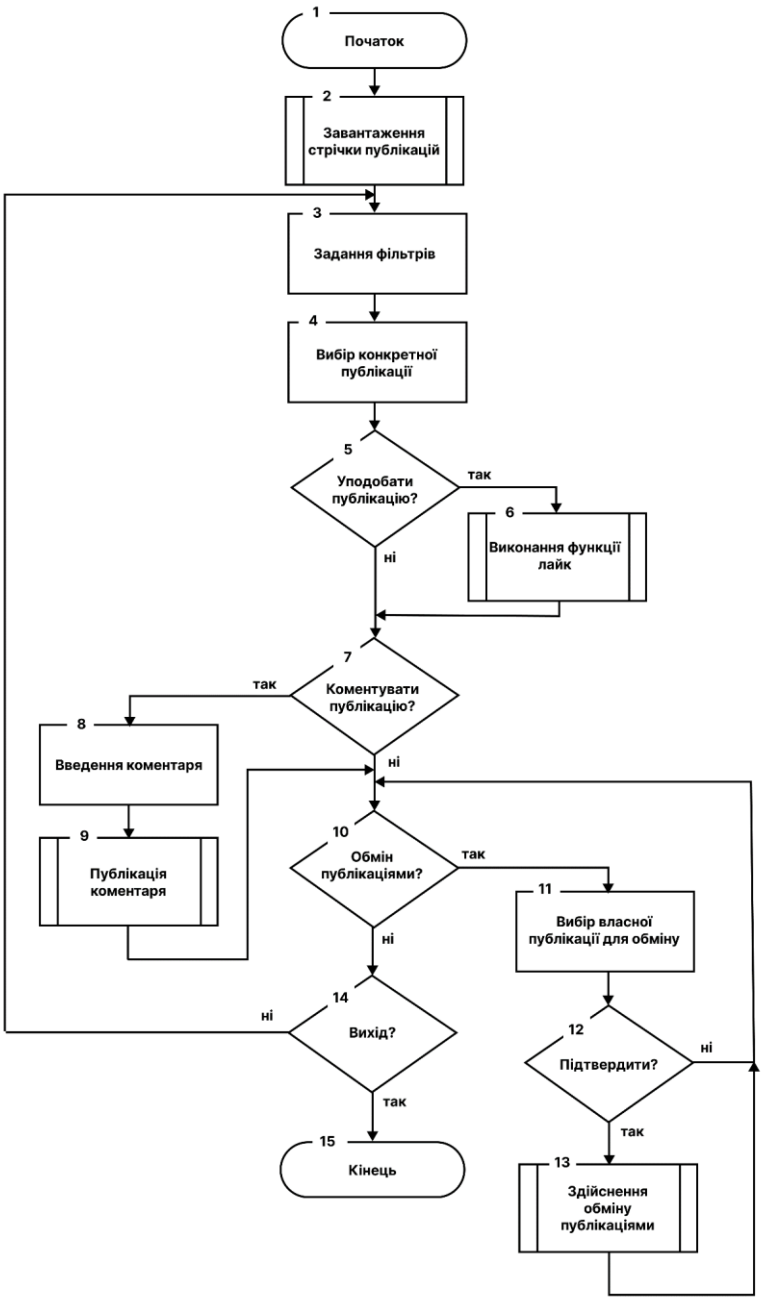

Рисунок 2.7 – Алгоритм взаємодії користувача з публікаціями

Отже, розроблений алгоритм гарантує інтуїтивну, безпечну та гнучку взаємодію користувачів із системою, охоплюючи основні сценарії роботи з публікаціями: вподобання, коментування та обмін. Користувачі можуть легко реагувати на контент, залишати свої відгуки у вигляді лайків і коментарів, що сприяє формуванню динамічної та зацікавленої спільноти. Додаткова можливість ділитися публікаціями з іншими користувачами сприяє поширенню контенту та залученню нових учасників до платформи. Усі функціональні можливості реалізовані з урахуванням зручності користування, а також із дотриманням вимог щодо захисту особистих даних та інформаційної безпеки.

Для забезпечення об’єктивного, послідовного та структурованого оцінювання вмісту публікацій у вебзастосунку було розроблено алгоритм оцінювання якості зображення, який дозволяє користувачам поетапно проаналізувати публікацію за низкою важливих критеріїв (див. рисунок 2.8).

Алгоритм починається з ініціації процесу оцінювання, який передбачає вибір конкретної публікації користувачем. Далі йде обов’язковий етап ознайомлення з роботою, що дозволяє сформувати загальне уявлення про зміст публікації перед тим, як переходити до оцінювання окремих її елементів.

Першим критерієм оцінювання є доречність заголовка. Якщо заголовок не відповідає змісту або є недоречним, публікація отримує мінімальну оцінку за цей критерій – 1 бал. Якщо заголовок вважається прийнятним, користувач вводить свою оцінку за шкалою від 2 до 10 балів. Наступним кроком є перевірка відповідності наповнення публікації її темі, що оцінюється за аналогією попереднього критерія. Далі перевіряється релевантність тегів. Наявність невідповідних або хибних тегів знижує якість навігації сайтом і ускладнює пошук, тому якщо теги недоречні.

Останнім етапом суб’єктивного оцінювання є враження користувача від самого зображення. У цьому випадку передбачено дві гілки оцінювання: якщо зображення не подобається, оцінювач може поставити від 1 до 5 балів; якщо воно викликає позитивне враження – від 6 до 10 балів. Такий підхід дозволяє збалансувати технічну і візуальну оцінку.

Після виставлення всіх оцінок система генерує результати оцінювання, які виводяться на екран. Користувач має можливість переглянути підсумкові оцінки та прийняти рішення, чи надсилати результати на сервер. У разі згоди – дані передаються, і користувач отримує повідомлення про успішну відправку. Цим етапом завершується робота алгоритму.

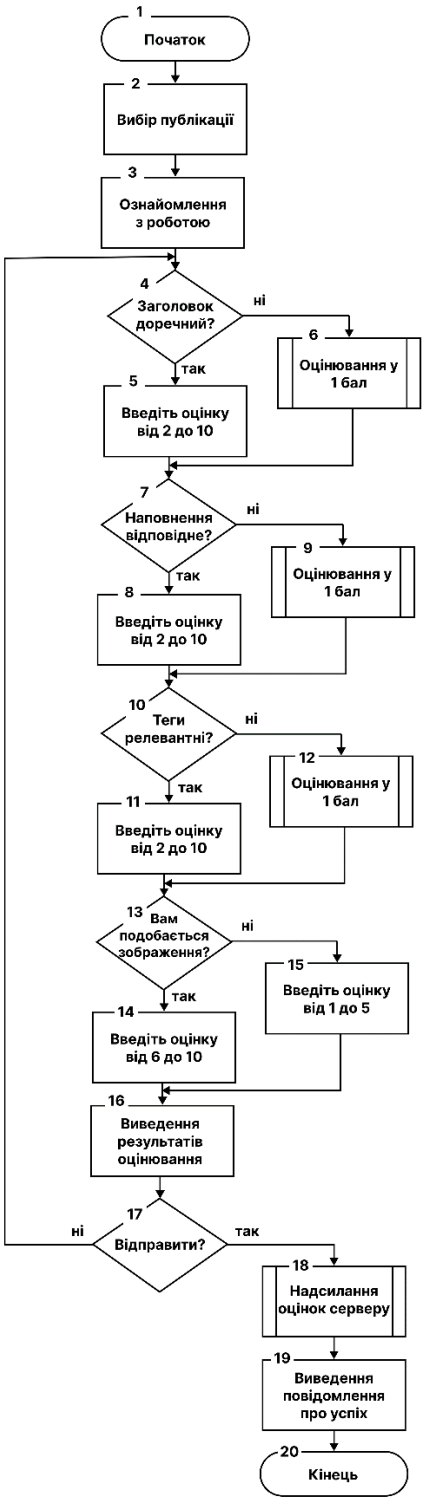

Рисунок 2.8 – Алгоритм оцінювання якості зображення

Таким чином, розроблений алгоритм дозволяє уніфіковано оцінювати публікації за критеріями змісту, оформлення та естетики, сприяючи покращенню якості контенту у вебзастосунку та формуючи довіру до оцінювання серед користувачів.

2.7 Висновки

Було здійснено ґрунтовний аналіз вхідних даних системи, за результатами якого сформовано модель функціонування вебплатформи та розроблено основні алгоритми, що забезпечують інтуїтивну взаємодію користувачів з інтерфейсом. У ході дослідження визначено, що ключовими об’єктами обробки є зображення та супровідні метадані (такі як авторство, дата завантаження, теги, описи), а також інформація про користувачів – їхні облікові дані, історія взаємодій, оцінки та вподобання. Однією з важливих особливостей реалізації є впровадження токенізованої авторизації, що гарантує безпечний доступ до персоналізованих функцій платформи.

У процесі проєктування було виконано розробку структури програмного застосунку, зокрема створено діаграму варіантів використання (use case diagram), яка відобразила можливості взаємодії користувачів із системою. Це дозволило визначити функціональні пріоритети та краще спланувати адаптивність інтерфейсу під різні сценарії. Додатково створено прототипи інтерфейсів із врахуванням адаптації для різних пристроїв (десктоп, планшет, мобільний телефон), що значно поліпшило зручність користування застосунком та підвищило його інклюзивність.

Модель системи представлено у вигляді UML-діаграми активності, яка демонструє типову взаємодію користувача з додатком, починаючи від авторизації та навігації, до участі в створенні, оцінюванні й перегляді публікацій. Окрім того, модель доповнено діаграмою класів, яка ілюструє структуру об’єктів, їхні атрибути, зв’язки та залежності. Це дозволяє чітко уявити архітектуру системи на рівні взаємопов’язаних сутностей.

Особливу увагу було приділено розробці методів роботи системи.

У свою чергу, розроблені алгоритми логічно й послідовно описують усі етапи взаємодії користувача з системою – від моменту завантаження зображення, його покрокового оцінювання, до фінального рішення про надсилання оцінки та завершення сесії. Завдяки цьому забезпечено цілісний сценарій користувацького досвіду, що відповідає сучасним вимогам до UX/UI.

У підсумку, проведені дослідження та реалізовані компоненти дали змогу створити функціонально насичену, масштабовану модель вебзастосунку, що задовольняє вимоги до сучасних інтерактивних платформ, орієнтованих на роботу з візуальним контентом та оцінюванням.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для реалізації програмних модулів вебсистеми, призначеної для візуалізації, аналізу та оцінювання статичних зображень, було підібрано сучасний стек технологій, що забезпечує високу продуктивність, якість та зручність подальшої розробки й супроводу системи. На фронтенд-частині (інтерфейсна частина застосунку) використано мову програмування JavaScript, бібліотеку React, потужний препроцесор SCSS, бібліотеку Chart.js для побудови графіків, стан-менеджер Zustand та бібліотеку для HTTP-запитів Axios. На бекенді застосовано Node.js, механізм авторизації з використанням токенів JWT та базу даних PostgreSQL. Розробка здійснювалася у зручному та функціональному середовищі Visual Studio Code**,** яке забезпечує підтримку численних інструментів та розширень для комфортної роботи з проєктом [24].

JavaScript було обрано як основну мову програмування, оскільки це стандарт для веброзробки, підтримуваний усіма сучасними браузерами. Її універсальність, широке розповсюдження та повна підтримка усіма сучасними браузерами забезпечують високу сумісність та безперебійну роботу інтерфейсу на різних платформах. JavaScript дозволяє створювати динамічні, інтерактивні сторінки, обробляти події в реальному часі, а також легко взаємодіяти з бекендом за допомогою HTTP-запитів. Крім того, завдяки розвитку таких фреймворків, як React, Vue, Angular, можливості JavaScript розширилися до побудови повноцінних SPA (Single Page Applications), що підвищує продуктивність та зручність користування [25].

Серед можливих альтернатив розглядалися мови, що також мають широке використання в веброзробці, такі як Python, Ruby та C#.

Python – це високорівнева мова програмування, яка відома своєю читабельністю та простотою синтаксису. Хоча Python не є нативною мовою для браузерного виконання, існують проєкти, такі як Transcrypt та Brython, які

дозволяють компілювати Python-код у JavaScript. Однак ці рішення мають обмежену продуктивність, обмежену інтеграцію з браузерним API, а також низький рівень підтримки у спільноті фронтенду. Тому Python залишається більш придатним для бекенд-розробки або наукових обчислень, ніж для розробки інтерфейсів [26].

Ruby – ще одна високорівнева мова, орієнтована на простоту та елегантність коду. Існує транслятор Opal**,** який дозволяє перетворювати Ruby-код у JavaScript, що відкриває можливості для використання Ruby у фронтенді [27]. Проте цей інструмент має меншу популярність, порівняно низьку продуктивність та обмежену кількість бібліотек, сумісних з браузером. Також, інтеграція з популярними фронтенд-фреймворками (такими як React або Vue) взагалі відсутня.

C# – це об'єктно-орієнтована мова програмування від Microsoft, яка зазвичай використовується у розробці десктопних, серверних і мобільних додатків. З виходом технології Blazor WebAssembly, стало можливим використовувати C# і для фронтенд-розробки: код компілюється у WebAssembly та виконується прямо в браузері. Це дозволяє програмістам використовувати одну мову (C#) як для клієнтської, так і серверної частини [28].

Перевагами такого підходу є висока продуктивність, повторне використання коду між фронтендом і бекендом, а також потужні можливості C#. Проте недоліками залишаються більший розмір завантаження сторінки, повільніше перше завантаження та менша екосистема у сфері вебінтерфейсів порівняно з JavaScript.

Отже, для обґрунтованого вибору мови програмування необхідно провести порівняння функціональних вимог мов програмування JavaScript, C#, Ruby та Python. Це важливо, бо правильний вибір мови програмування дозволяє знизити витрати часу і ресурсів на розробку та подальшу підтримку проєкту. Крім того, це забезпечує кращу якість, надійність і масштабованість програмного продукту відповідно до поставлених задач.

Порівняння функціональних характеристик мов програмування наведено в таблиці 3.1.

Таблиця 3.1 – Таблиця порівняння функціональних характеристик мов програмування

Критерії функціональних вимог	JavaScript	C#	Python	Ruby
Підтримка мікросервісів	1	1	1	0
Low-code / Rapid prototyping	1	0	1	1
Пакетний менеджер (npm, NuGet, pip, gem)	1	1	1	1
Безпека / шифрування	1	1	1	0
AI/ML/DS підтримка	0	0	1	0
High-load системи	1	1	0	0
WebAssembly / вбудованість у браузер	1	1	0	0
BigTech підтримка	1	1	1	0
Загальний коефіцієнт	7	6	6	2

Отже, JavaScript отримує найбільшу кількість балів, демонструючи найширший функціональний спектр для веброзробки та інтеграції у сучасні технології. На основі цих даних було створено графік порівняння мов програмування (JavaScript, C#, Python, Ruby) за поглибленими функціональними критеріями. Кожен стовпчик відображає, чи підтримує мова відповідний функціонал (див. рисунок 3.1).

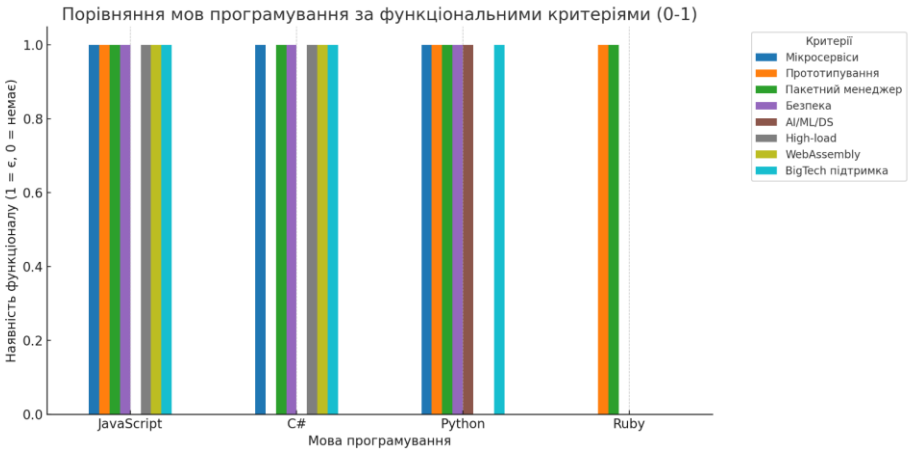

Рисунок 3.1 – Графік порівняння функціональних вимог

Після вибору мови програмування наступним важливим кроком є вибір середовища розробки (IDE або редактора коду). Від цього залежить не лише зручність написання коду, але й ефективність усієї розробки – включаючи налагодження, інтеграцію з системами контролю версій, автодоповнення коду та інші важливі функції. Якісне середовище допомагає програмісту зосередитися на логіці проєкту, мінімізуючи рутинні завдання.

Visual Studio Code (VS Code) – це безкоштовний, легкий, але надзвичайно потужний редактор коду від Microsoft. Він підтримує безліч мов програмування завдяки розширенням, має зручне автодоповнення, вбудований термінал, Git-інтеграцію та тисячі плагінів, які розширюють його можливості. Його відкритий код і активна спільнота роблять його одним із найпопулярніших редакторів серед розробників. Завдяки гнучкості, стабільності та великій кількості функцій Visual Studio Code ідеально підходить як для новачків, так і для професіоналів [29].

Sublime Text – це швидкий, легкий та мінімалістичний редактор коду, який підходить для тих, хто цінує швидкість запуску і простоту. Він має підтримку багатьох мов програмування, можливість додавання плагінів через Package Control та приємний інтерфейс. Однак деякі розширені функції доступні лише після купівлі ліцензії, а базова версія менш функціональна без додаткових налаштувань [29].

Atom – це редактор коду, розроблений GitHub, який відомий своєю відкритістю та кастомізацією. Має вбудовану підтримку Git та GitHub, численні теми і плагіни, а також зручне редагування файлів у реальному часі кількома користувачами. Проте останніми роками його розвиток сповільнився, і він працює повільніше на великих проєктах у порівнянні з іншими редакторами [30].

Visual Studio – це потужне інтегроване середовище розробки (IDE), створене компанією Microsoft, яке призначене для розробки програмного забезпечення на різних мовах програмування, зокрема C#, C++, Visual Basic, Python, JavaScript та інших [31]. Воно забезпечує широкий спектр інструментів для написання, налагодження, тестування та розгортання програм, включаючи підтримку .NET, роботу з базами даних, створення вебзастосунків, десктопних і мобільних додатків.

Порівняння функціональних характеристик середовищ наведено в таблиці 3.2.

Таблиця 3.2 – Таблиця порівняння функціональних характеристик середовищ програмування

Функціональна характеристика	VS Code	Sublime Text	Atom	Visual Studio
Підсвічування синтаксису	1	1	1	1
Автодоповнення коду	1	0	1	1
Вбудований термінал	1	0	1	1
Інтеграція з Git	1	0	1	1
Плагіни/розширення	1	1	1	1
Підтримка кількох мов	1	1	1	1
Налагодження (debugging)	1	0	0	1
Підтримка рефакторингу	1	0	0	1
Інтегорація хмарних технологій	1	0	0	1
Безкоштовність	1	1	1	1
Кросплатформеність	1	1	1	0
Загальний коефіцієнт	11	5	8	10

Згідно з оцінкою функціональних характеристик, Visual Studio Code отримує найвищу кількість балів, що свідчить про його універсальність, зручність і розширюваність. Саме тому було обрано Visual Studio Code для розробки вебзастосунку для візуалізації, аналізу та оцінювання статичних зображень.

У якості бібліотеки для побудови користувацького інтерфейсу обрано React. Він дозволяє створювати швидкі, гнучкі та масштабовані односторінкові застосунки завдяки використанню компонентного підходу. На відміну від альтернативних рішень, таких як Angular або Vue.js, React є менш обтяжливим за кількістю обов'язкових структурних вимог і має значно ширшу підтримку серед розробників комерційних проєктів.

Для стилізації інтерфейсу використано SCSS – потужний препроцесор для CSS, що розширює можливості стандартних каскадних таблиць стилів за рахунок підтримки змінних, вкладеності, міксинів та інших інструментів для оптимізації структури коду. У порівнянні з аналогами, такими як LESS або Stylus, SCSS має кращу інтеграцію з екосистемою React і більш популярний серед розробників, що полегшує командну розробку та супровід проєкту [22].

Для побудови графіків і візуалізації аналітичних даних обрано бібліотеку Chart.js. Вона дозволяє швидко інтегрувати графіки до вебзастосунку, має простий API та підтримує широкий набір типів діаграм. Серед альтернатив розглядалися D3.js та ApexCharts. D3.js є потужним інструментом для глибокої кастомізації візуалізацій, однак має складний API і вимагає великих витрат часу на розробку навіть базових графіків. ApexCharts має зручніший інтерфейс, однак Chart.js виграє завдяки простоті інтеграції, хорошій продуктивності на малих і середніх наборах даних та ширшій спільноті підтримки. Для потреб аналізу статичних зображень і їх оцінювання Chart.js є оптимальним вибором [23].

Для управління станом застосунку використано Zustand – легкий і зручний у використанні стан-менеджер для React. На відміну від таких рішень як Redux чи MobX, Zustand не вимагає складної конфігурації і надмірної кількості шаблонного коду, що дає змогу скоротити час на розробку та полегшити підтримку модулів стану [21].

Усі запити до бекенду здійснюються за допомогою бібліотеки Axios, яка спрощує виконання HTTP-запитів завдяки зручному інтерфейсу для налаштування параметрів запиту, обробки помилок і підтримки асинхронності.

Для бекенд-частини було обрано Node.js як серверне середовище виконання JavaScript-коду. Node.js дозволяє використовувати єдину мову програмування як на стороні клієнта, так і на сервері, що суттєво спрощує розробку і уніфікує кодову базу. На відміну від таких альтернатив як Django (Python) чи Spring Boot (Java), Node.js має неблокуючу модель обробки запитів, що забезпечує високу продуктивність при роботі з великою кількістю одночасних з'єднань [32].

Для авторизації користувачів застосовано механізм токенів на основі стандарту JWT (JSON Web Token). Цей підхід дозволяє безпечно передавати дані між клієнтом і сервером без необхідності зберігати стан сесії на сервері, що особливо важливо для масштабованих розподілених систем [33].

У якості системи керування базами даних було обрано PostgreSQL, що є потужною об'єктно-реляційною СКБД з відкритим кодом. PostgreSQL забезпечує підтримку складних транзакцій, розширене управління типами даних та можливість виконання аналітичних запитів. У порівнянні з MySQL PostgreSQL краще обробляє великі обсяги даних і складні запити, а на відміну від MongoDB підтримує складні зв'язки між сутностями, що є критично важливим для коректного моделювання даних у межах даного проєкту [20].

Таким чином, вибір зазначених засобів реалізації було здійснено на основі їх відповідності вимогам до системи, високої продуктивності, зручності розробки, надійності та масштабованості. Обраний стек технологій забезпечує ефективну розробку, якісну інтеграцію модулів і зручність подальшого супроводу вебсистеми.

3.2 Розробка модуля оцінювання статичних даних

У межах створення вебсистеми візуалізації, аналізу й оцінювання статичних зображень важливим компонентом є модуль оцінювання. Його основною функцією є забезпечення можливості користувачам об'єктивно оцінювати подані зображення за визначеними критеріями, що формує базу для подальшого аналізу якості контенту та прийняття рішень на основі отриманих оцінок.

Впровадження цього модуля є важливим з кількох причин. По-перше, він дозволяє здійснювати систематизований збір даних про сприйняття зображень великою кількістю користувачів, що забезпечує більш надійну та репрезентативну аналітику. По-друге, модуль створює умови для чесного конкурсного відбору, оскільки оцінювання відбувається за однаковими для всіх учасників критеріями. По-третє, використання чітких критеріїв оцінки допомагає мінімізувати суб'єктивність і сприяє більш прозорому визначенню результатів.

Вигляд розробленого головного фрагменту модуля проведення оцінювання якості зображень зображено на рисунку 3.2.

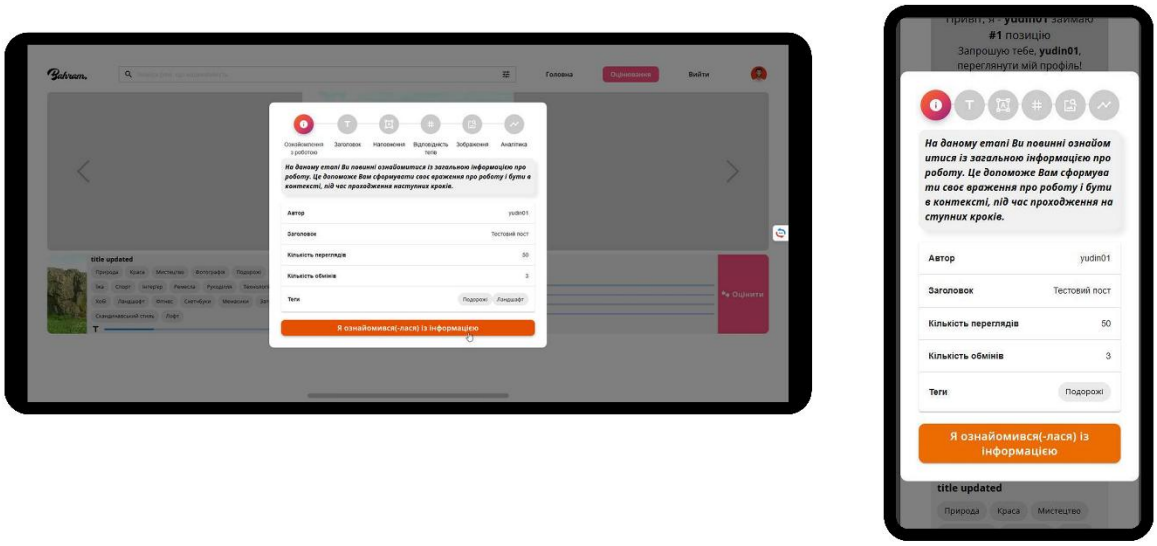

Рисунок 3.2 – Фрагмент модуля оцінювання якості зображення

Модуль розрахунку оцінок реалізовано за допомогою JavaScript у середовищі Node.js. Його основним завданням є обробка та підсумовування окремих оцінок, наданих користувачами за різними критеріями, та обчислення загальної оцінки з урахуванням вагових коефіцієнтів. Блок-схема алгоритму розрахунку оцінок зображена на рисунку 3.3.

У методі calculateMarkFields (marks: UpdateMarkDto) здійснюється оновлення проміжних результатів оцінювання за такими критеріями, як якість зображення (imageRating), релевантність заголовка (titleRating), змістовність контенту (contentRating) та відповідність тегів (tagsRating). Значення для кожного критерію накопичуються, додаючи оцінки нових користувачів.

Після оновлення окремих рейтингів обчислюється загальна оцінка totalRating. Формула враховує різну вагу критеріїв: оцінка за зображення має коефіцієнт 1, за заголовок – 0.5, за зміст – 0.5, а за теги – 0.3.

Завдяки такій структурі коду забезпечується прозоре, масштабоване та контрольоване оновлення оцінок, що є важливим для об'єктивного підбиття підсумків конкурсів та збору аналітичної інформації.

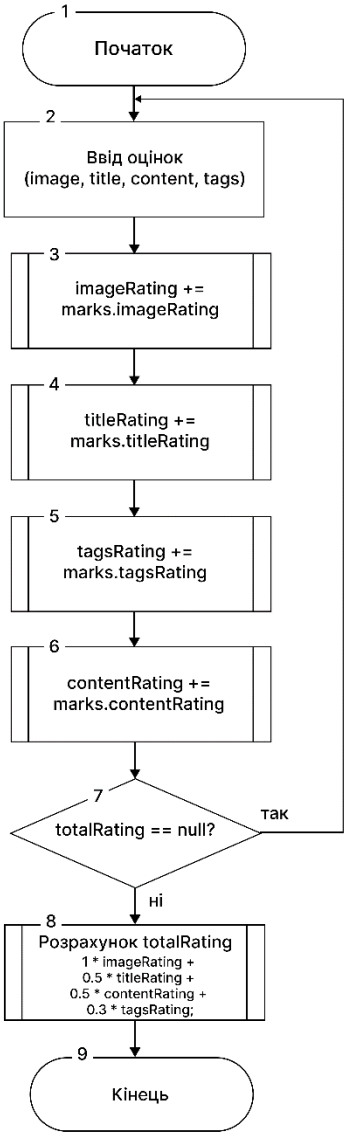

Рисунок 3.3 – Блок-схема алгоритму розрахунку оцінок

Для розробки модуля було реалізовано функціонал модального вікна для покрокового оцінювання конкурсних постів у межах конкурсу. Блок-схема алгоритму функціоналу модального вікна зображена на рисунку 3.4. Основна мета – надати користувачу можливість послідовно оцінити різні аспекти поста за встановленими критеріями. Після завершення оцінювання усі зібрані оцінки відправляються на сервер для подальшої обробки та збереження.

Рішення побудовано з використанням React у поєднанні з TypeScript для підвищення надійності типізації компонентів і функцій. Для підвищення продуктивності застосовано React-хуки: компонент MarathonVotePostModal

реалізовано на основі React із використанням TypeScript для типізації даних. Для оптимізації роботи з пам'яттю та зменшення кількості непотрібних перерендерів застосовано стандартні React-хуки useCallback (для мемоізації функцій) та useMemo (для мемоізації обчислюваного контенту залежно від активного кроку).

Передача оцінок на сервер здійснюється через функцію postEvaluationVote, що взаємодіє з серверною частиною, реалізованою на Node.js із використанням Express. Сервер приймає оцінки для постів і зберігає їх у базі даних.

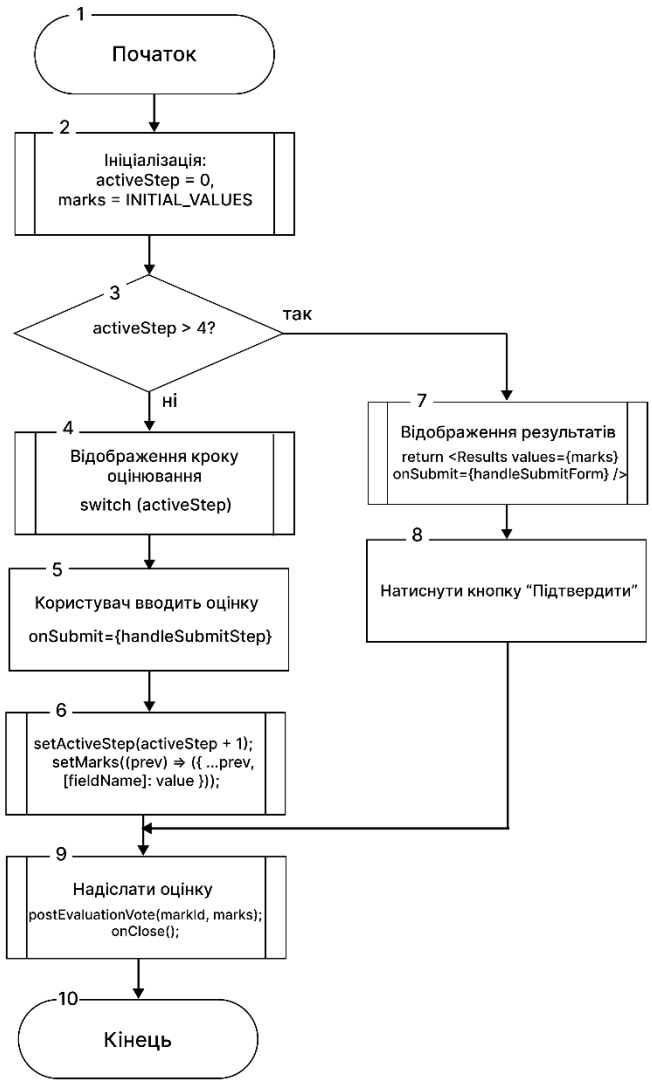

Рисунок 3.4 – Блок-схема алгоритму функціоналу модального вікна

Структурування інтерфейсу модального вікна забезпечується за допомогою власних CSS-стилів або підключення модульних стилів (імпорт styles). Модальне вікно відкрите постійно (open={true}) і має обробник закриття через onClose.

Таким чином, реалізовано покрокову форму оцінювання, де залежно від поточного стану (activeStep) рендериться відповідний крок оцінювання. Після завершення всіх кроків відбувається остаточна відправка даних на сервер.

Також у межах розробки модуля було реалізовано функціонал централізованого стану для керування оцінюванням конкурсних робіт у межах марафону. Рішення побудовано із застосуванням бібліотеки Zustand у поєднанні з TypeScript для забезпечення надійної типізації стану та методів оновлення даних. Zustand обрано завдяки його мінімалістичному API, що дозволяє ефективно працювати зі станом без надмірної складності.

Для забезпечення взаємодії із серверною частиною використано бібліотеку Axios, яка дозволяє здійснювати асинхронні HTTP-запити з обробкою помилок. Власна конфігурація axios.interceptor забезпечує централізовану обробку запитів і відповідей, що підвищує безпеку та уніфікацію мережевих запитів.

Код містить чотири основні методи для роботи з оцінюванням:

- fetchParticipants: отримує поточний список учасників для оцінювання;
- postEvaluationVote: надсилає оцінку для певного учасника на сервер;
- postArticleForEvaluation: додає статтю до списку для оцінювання;
- removeArticleFromEvaluation: видаляє статтю зі списку оцінювання.

Серверна частина для обробки запитів реалізована на Node.js із використанням Express.js, що дозволяє ефективно обробляти маршрути для керування об'єктами оцінювання. Дані оцінювання зберігаються у базі даних після проходження валідації на стороні сервера.

Типізація таких структур, як EvaluateArticleItemType та UpdateEvaluationArticleType, забезпечує попередню перевірку правильності даних ще на етапі розробки, що зменшує кількість потенційних помилок на продакшн-етапі.

Таким чином, поєднання Zustand, TypeScript та Axios дозволяє створити легкий, ефективний та розширюваний механізм роботи з оцінюванням конкурсних публікацій у реальному часі.

3.3 Розробка модуля управління обміном статичних даних

Модуль обміну зображеннями є важливою складовою системи взаємодії користувачів на платформі, адже він забезпечує можливість безпосереднього обміну об'єктами контенту між учасниками без необхідності сторонніх погоджень або додаткових запитів. Реалізація механізму обміну підвищує гнучкість сервісу, створює нові сценарії використання платформи та покращує загальний досвід користувача. Вигляд розробленого модального вікна модуля управління обміном зображень зображено на рисунку 3.5.

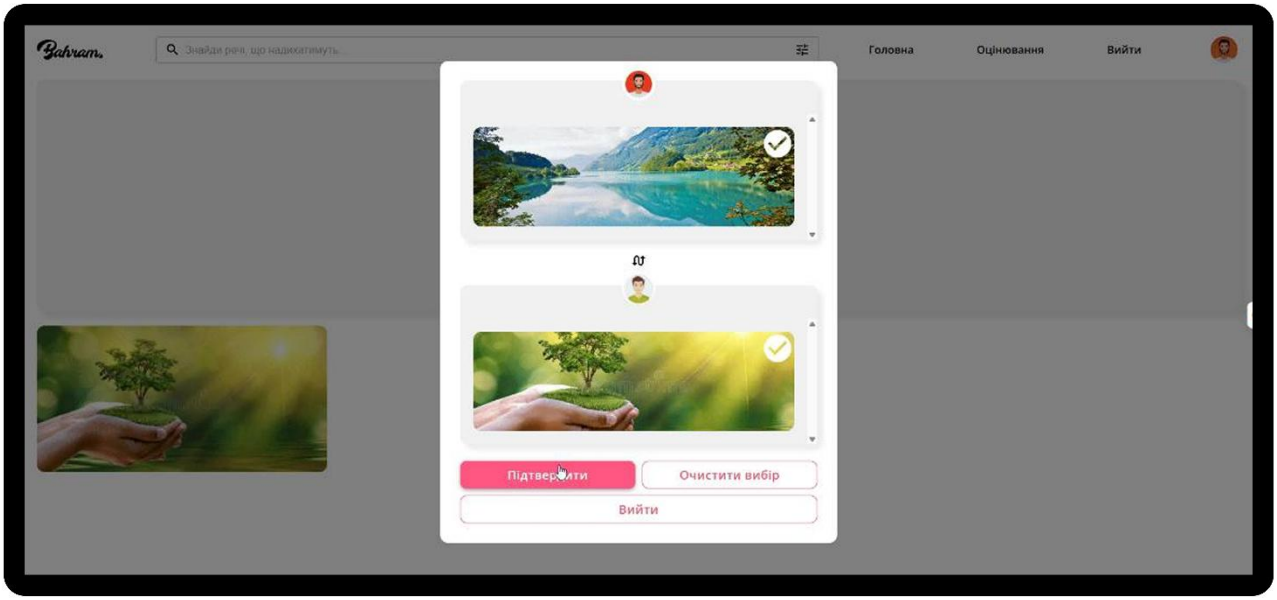

Рисунок 3.5 – Модальне вікно модуля управління обміном зображень

Функція exchangeArticle асинхронно виконує обмін авторства між двома публікаціями. Спочатку за допомогою методу findBySlug відбувається пошук обох публікацій за їх унікальними ідентифікаторами slug. Далі відбувається обмін авторами між цими публікаціями: власний об'єкт статті отримує автора партнера, а публікація партнера отримує нового власника – користувача, який ініціює обмін. Після зміни авторів обидві публікації зберігаються за допомогою методу save(). У фіналі функція повертає оновлену версію партнерської статті, яка персоналізується для поточного користувача.

Блок-схема алгоритму обміну публікаціями між користувачами зображена на рисунку 3.6.

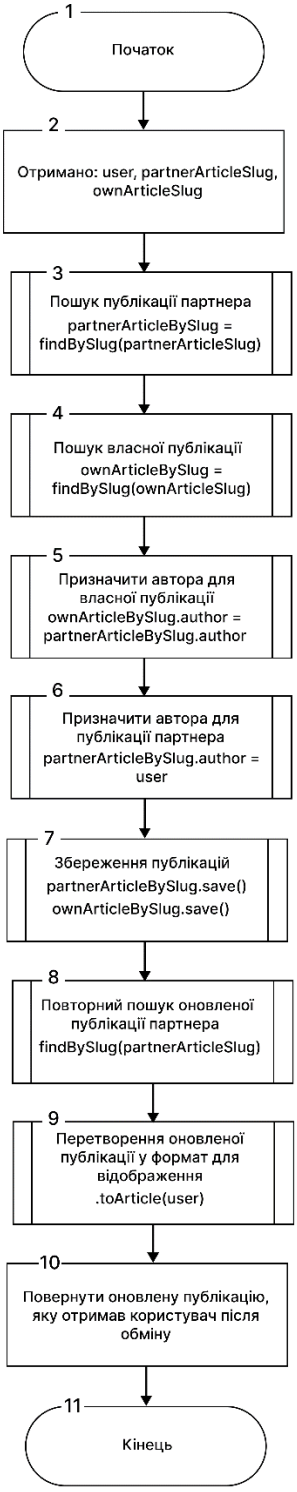

Рисунок 3.6 – Блок-схема алгоритму обміну публікаціями між користувачами

Для організації процесу обміну постами між двома користувачами в модальному вікні було використано компонент ExchangePostModal. У цьому

компоненті реалізовано відображення списків постів обох користувачів, вибір кожного з них, а також підтвердження або скасування дії. Користувач бачить у модальному вікні дві колонки: одна з його власними постами, інша – з постами партнера. Обрані пости позначаються відповідними слагами, які зберігаються в об'єкті exchangePostSlug.

Блок-схема алгоритму обміну публікаціями між користувачами в модальному вікні зображена на рисунку 3.7, відображаючи основні етапи перевірки вибору публікацій, виконання обміну та завершення операції.

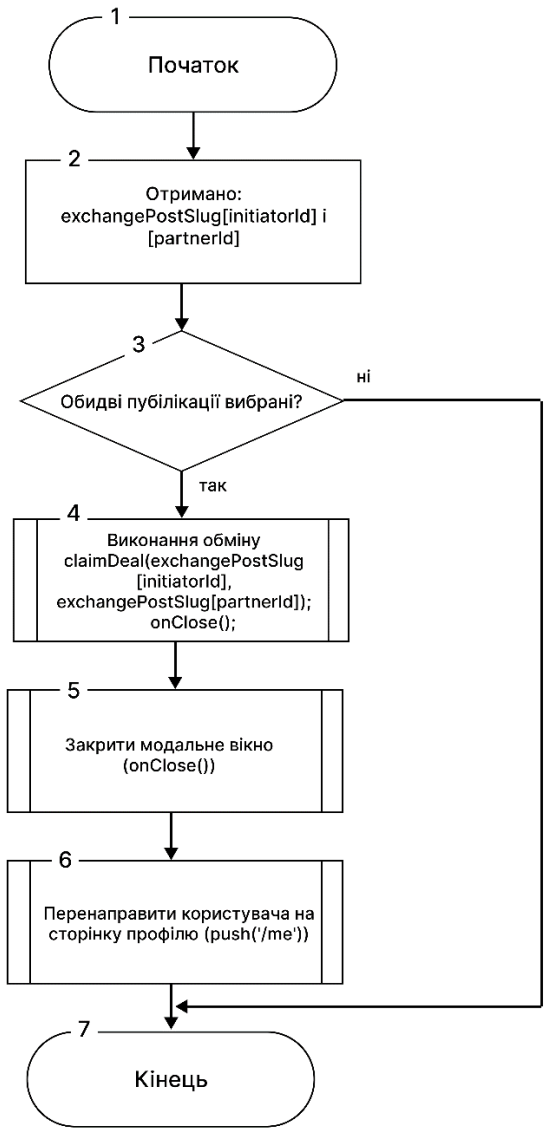

Рисунок 3.7 – Блок-схема алгоритму обміну публікаціями між користувачами в модальному вікні

Отже, основна логіка підтвердження обміну реалізована у функції handleSubmitExchange. У першу чергу в ній перевіряється, чи були вибрані обидва пости: якщо хоча б один із слагів відсутній, функція нічого не виконує й повертає null. Якщо ж обидва пости вибрані, викликається функція claimDeal, яка здійснює сам обмін постами на основі обраних слагів. Після цього вікно модального компонента закривається за допомогою onClose, і користувача автоматично перенаправляють на його сторінку через push('/me').

Інтерфейс модального вікна побудований на основі компонента Modal із кількома внутрішніми елементами: блоками списків постів, кнопками для підтвердження, очищення вибору та виходу [34]. Кнопка підтвердження (Підтвердити) активна лише тоді, коли обидва пости вибрані, що контролюється змінною isSubmitDisabled. Кнопка очищення (Очистити вибір) скидає вибрані пости, а кнопка Вийти просто закриває вікно.

Компонент `UserExchangePostList` для відображення та вибору постів користувача призначений для виведення списку публікацій, пов’язаних із конкретним автором. Він приймає три основні пропси: масив постів `posts` типу `ArticleType[]`, функцію `onClick`, яка викликається при виборі поста, та значення `chosenPostSlug`, що зберігає `slug` наразі вибраного поста або є `null`, якщо пост не вибрано. На початку компонент перевіряє, чи масив постів не порожній. Якщо постів немає, повертається `null`, тобто нічого не відображається. Із першого елемента масиву постів береться об’єкт автора, і за допомогою компонента `UserAvatar` виводиться його аватарка (див. рисунок 3.8).

Основна частина компонента – це відображення списку постів. Для кожного поста створюється окремий блок, який має стилізацію через клас `imageWrapper`. Якщо певний пост не є вибраним (тобто його `slug` не збігається з `chosenPostSlug`), йому додатково присвоюється клас `disabled`, що робить його менш помітним або неактивним. При кліку на пост викликається функція `onClick` із переданим `slug` цього поста.

Якщо пост вибрано (його `slug` збігається з `chosenPostSlug`), поверх зображення виводиться іконка `CheckCircleIcon`, що вказує на вибір. Зображення самого поста виводиться через компонент `Image`.

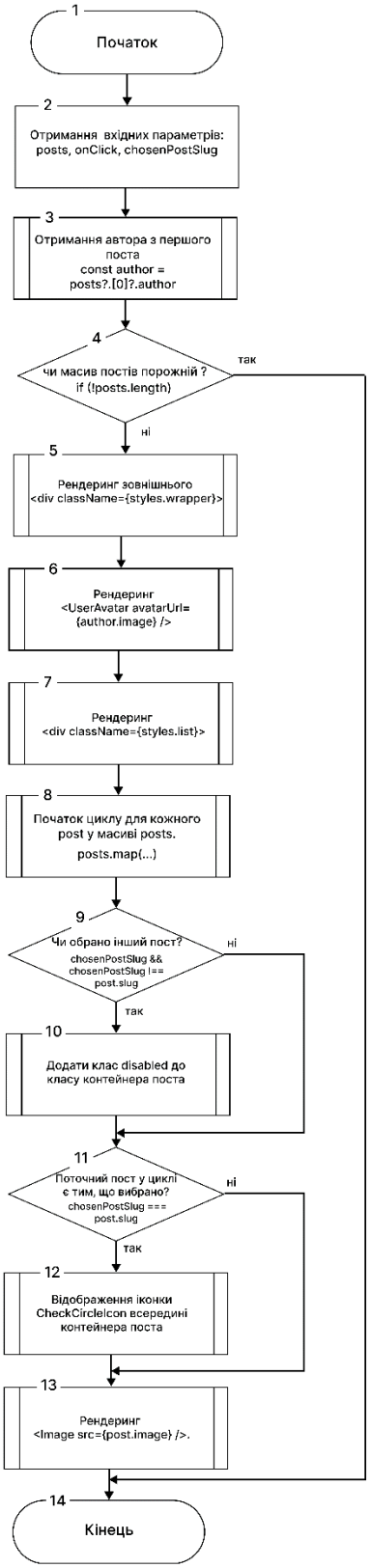

Рисунок 3.8 – Блок-схема алгоритму для відображення та вибору постів

Таким чином, `UserExchangePostList` виконує роль інтерфейсу для вибору одного з кількох постів, з наочним візуальним позначенням вибраного елемента та можливістю змінити вибір.

У свою чергу, для забезпечення динамічного керування процесом обміну зображеннями між користувачами в реальному часі важливо створити ефективну систему зберігання та управління станом даних на клієнтській стороні. Саме цю роль виконує модуль керування станом, побудований на базі бібліотеки Zustand – сучасного інструмента для управління глобальним станом у React-додатках, який забезпечує високу продуктивність і простоту інтеграції [21].

Загалом модуль успішно вирішує завдання якісної організації процесу обміну та підвищує рівень взаємодії між користувачами.

3.4 Розробка модуля для збирання, обробки та візуалізації статистичних даних

Модуль дозволяє в режимі реального часу відстежувати активність користувачів щодо кожної публікації: кількість переглядів, лайків, коментарів та інших взаємодій. Це не лише покращує користувацький досвід, але й допомагає авторам публікацій краще розуміти вподобання своєї аудиторії, а адміністраторам – контролювати якість і популярність контенту.

Процес відбувається таким чином: після того як користувач створює публікацію, система автоматично починає збір статистичних даних щодо неї. Усі взаємодії – перегляди, оцінки, коментарі – фіксуються та обробляються у фоновому режимі. Користувачеві не потрібно вручну активувати жодну функцію – відстеження аналітики відбувається автоматично з моменту публікації поста.

У подальшому, зайшовши до особистого кабінету, користувач може натиснути на спеціальну іконку «Статистика», яка розміщується поряд з його публікаціями. Ця функція відкриває аналітичний модуль, де автор отримує можливість переглядати детальну статистику по кожному зі своїх постів. Діаграми, що візуалізують кількість коментарів, рейтинг та перегляди, дозволяють автору швидко оцінити ефективність і популярність кожної публікації, а також зрозуміти,

які теми викликають найбільший інтерес у його аудиторії. Інтерфейс розробленого модуля зображено на рисунку 3.9.

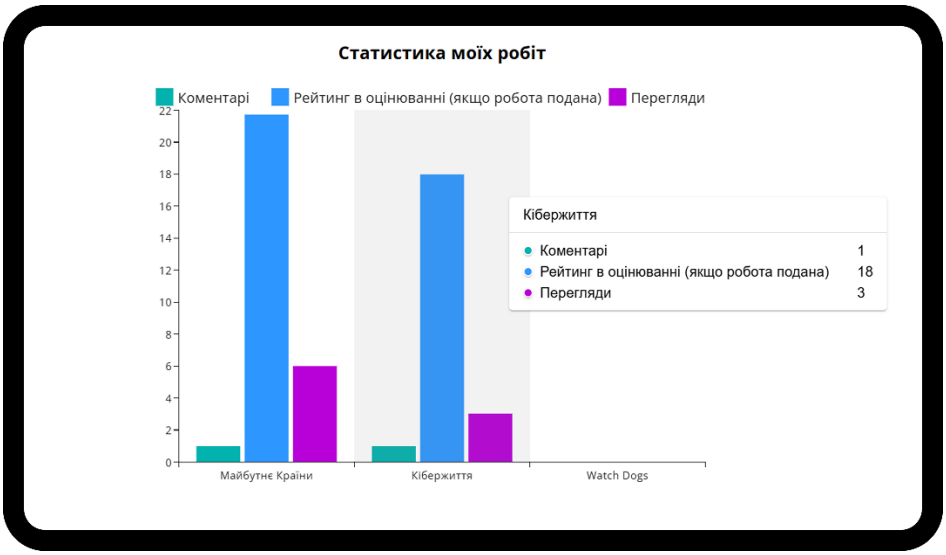

Рисунок 3.9 – Інтерфейс модуля для збирання, обробки та візуалізації статистичних даних

Компонент модуля Analytics є функціональним React-компонентом, який відповідає за відображення статистики публікацій користувача у вигляді стовпчикової діаграми (BarChart). Він отримує пропси posts, що є масивом об'єктів типу ArticleType або null. Блок-схема алгоритму візуалізації статистичних даних зображена на рисунку 3.10.

Усередині компонента за допомогою хука useMemo відбувається обчислення статистичних даних, необхідних для побудови графіків. Цей хук гарантує, що обчислення відбувається лише тоді, коли змінюється значення posts, що оптимізує продуктивність. Якщо posts не існує або є порожнім, функція повертає null, і жодна діаграма не виводиться [35].

Якщо дані є, формується структура postsStatistic – масив об'єктів із двома полями: data та label. У полі data зберігаються масиви значень для кожного виду метрики (кількість коментарів, загальний рейтинг та перегляди постів). Водночас, поле label відповідає за назву кожної серії даних, яка буде підписана на графіку. Також формується масив xAxises, який містить значення для осі X – назви

публікацій (post.title). Тип шкали встановлюється як «band», що підходить для категоріальних даних.

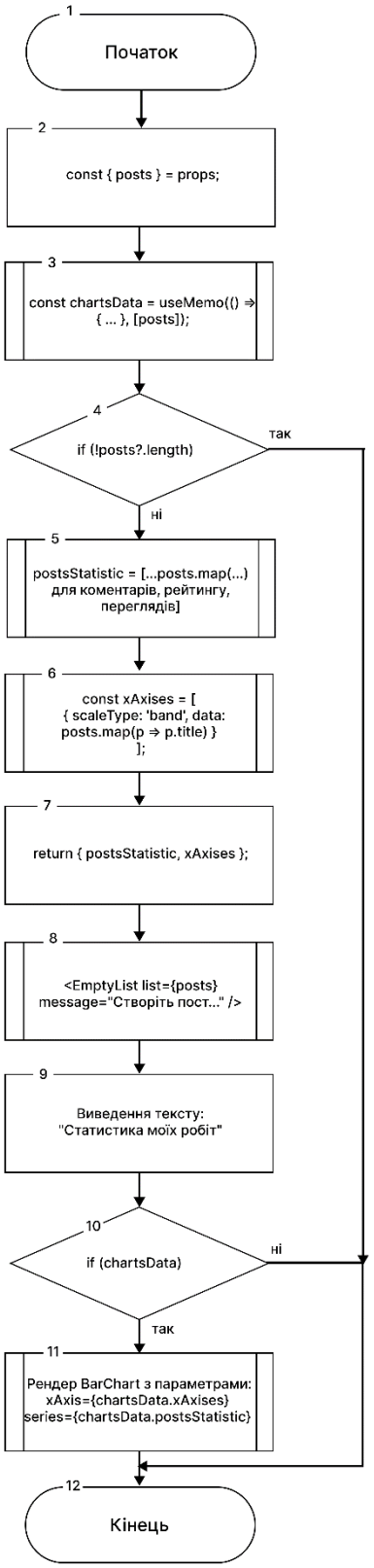

Рисунок 3.10 – Блок-схема алгоритму візуалізації статистичних даних

Основна частина JSX-розмітки обгорнута у компонент EmptyList, який перевіряє наявність списку постів. Якщо постів немає – відображається повідомлення з пропозицією створити публікацію. Якщо ж дані наявні, тоді відображається заголовок «Статистика моїх робіт» і сам графік (BarChart), в якому використовуються підготовлені осі та серії статистичних даних.

У підсумку, цей компонент забезпечує динамічну, інтерактивну та наочну візуалізацію статистики публікацій користувача, дозволяючи легко порівняти показники популярності й оцінювання кожної роботи.

3.5 Розробка інтерфейсу програмного застосунку

У процесі розробки інтерфейсу програмного застосунку було прийнято рішення поділити його на три логічно окремі сторінки, кожна з яких виконує певну функціональну роль. Такий підхід дозволяє спростити навігацію для користувача, зробити інтерфейс зрозумілішим та забезпечити кращий користувацький досвід. Основними сторінками інтерфейсу є:

1. Головна сторінка – слугує центральною точкою взаємодії користувача із застосунком, містить основні пости, статистику, загальну інформацію.

2. Сторінка оцінювання – надає можливість оцінювати публікації інших користувачів відповідно до певних критеріїв;

3. Особистий кабінет – дозволяє користувачам переглядати власні публікації та додавати нові, стежити за статистикою активності, переглядати збережені публікації, здійснювати обмін, брати участь в оцінюванні та змінювати дані профілю тощо.

Важливою частиною розробки стала адаптивність інтерфейсу. У сучасних умовах користувачі взаємодіють із веб-застосунками не лише через стаціонарні комп’ютери чи ноутбуки, а й за допомогою мобільних пристроїв – смартфонів і планшетів. Тому інтерфейс було реалізовано з дотриманням принципів адаптивного дизайну (responsive design) [15].

Це означає, що всі елементи інтерфейсу (меню, блоки з контентом, кнопки, графіки, шрифти тощо) автоматично підлаштовуються під розмір екрана,

забезпечуючи коректне відображення і зручну навігацію як на великому дисплеї ноутбука, так і на екрані мобільного телефона.

Для реалізації адаптивності були використані сучасні технології розмітки та стилізації (зокрема, CSS Flexbox/Grid, медіа-запити, мобільні breakpoints), а також компоненти, що динамічно змінюють свою поведінку залежно від ширини екрана.

Для забезпечення зручного та інтуїтивно зрозумілого переходу між основними сторінками застосунку було реалізовано систему навігації, що доступна користувачу на кожному з екранів (див. рисунок 3.11). Переміщення між сторінками – Головна**,** Оцінювання та Особистий кабінет здійснюється за допомогою навігаційного меню, розміщеного у верхній частині інтерфейсу (desktop-версія) або у вигляді мобільного бургер-меню (на смартфонах та планшетах).

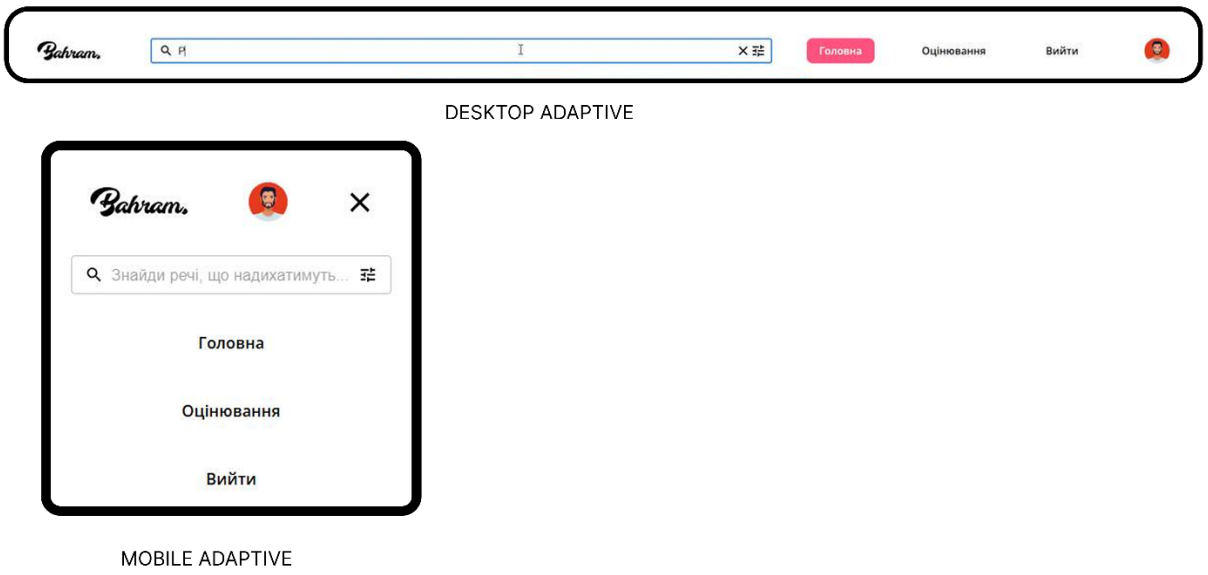

Рисунок 3.11 – Панель навігації застосунку

Головна сторінка вебплатформи Barham створює позитивне перше враження завдяки яскравому і гармонійному дизайну. У верхній частині розміщене зручне поле для пошуку, яке дозволяє швидко знаходити потрібні зображення за ключовими словами. Це спрощує навігацію та підвищує ефективність використання сайту. Праворуч у верхньому меню, є три основні кнопки: «Головна»

(виділена рожевим, активна), «Оцінювання», що веде до сторінки з можливістю оцінювати контент, кнопка «Вийти», яка дозволяє користувачеві завершити сесію та іконка особистого кабінету.

Центральне місце займає слайдер з якісними, яскравими фотографіями природи, що одразу привертає увагу та задає емоційний настрій. Такий візуальний контент викликає естетичне задоволення і мотивує продовжити перегляд.

Під слайдером із зображеннями в центрі розміщено блок з інформацією про поточного користувача. Напис «Привіт, я – yudinsasha займаю #2 позицію» інформує про місце в рейтингу. Далі наведене ім’я іншого користувача yudin01, якого запрошують переглянути. Головна сторінка зображена на рисунку 3.12.

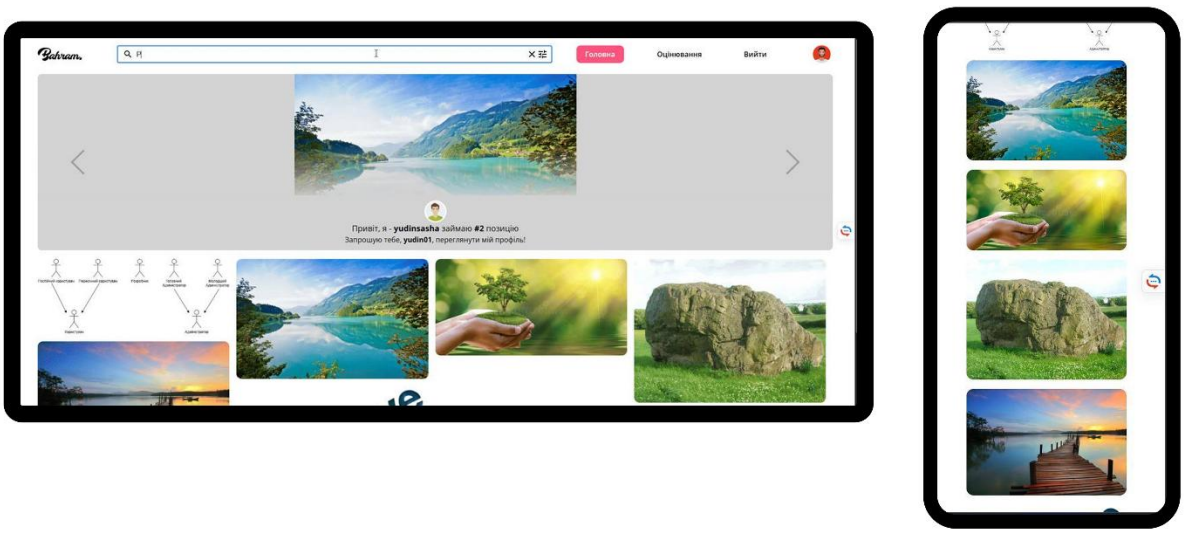

Рисунок 3.12 – Головна сторінка застосунку

Особистий кабінет на платформі Barham оформлений у лаконічному та інтуїтивно зрозумілому стилі. У верхній частині розташоване стандартне меню сайту з полем пошуку, кнопкою фільтрів, а також посиланнями на розділи, що забезпечує швидкий доступ до основних функцій платформи.

У центрі особистого кабінету відображається профіль користувача. Виводиться аватар із зображенням, ім’я користувача YUDIN01, електронна адреса yudin01@gmail.com і кількість створених робіт – 1. Нижче розташована яскрава кнопка «Створити», яка дозволяє створити нову публікацію.

У нижній частині особистого кабінету розміщено три вкладки, які забезпечують зручну навігацію між різними розділами взаємодії з контентом. Перша вкладка активна за замовчуванням і підсвічена синім кольором, що вказує на її вибраний стан. Цей розділ є галереєю користувача, де відображаються всі власні пости – тобто роботи, які були завантажені або створені. Тут користувач може взаємодіяти з ними: переглядати, редагувати, видаляти або брати участь у процесі оцінювання.

Друга вкладка відповідає за збережені публікації. Це ті роботи інших користувачів, які були позначені як цікаві чи корисні. Цей розділ дозволяє швидко повертатися до улюблених постів без потреби знову їх шукати на головній сторінці чи серед усіх матеріалів.

Третя вкладка містить статистику щодо власних публікацій. Тут користувач може переглядати дані про активність навколо своїх робіт: кількість переглядів, коментарів та інші аналітичні показники, які дозволяють оцінити ефективність і популярність контенту.

Завдяки такій структурі вкладок особистий кабінет стає зручним інструментом для управління власною активністю на платформі, аналізу результатів і збереження важливого контенту (див. рисунок 3.13).

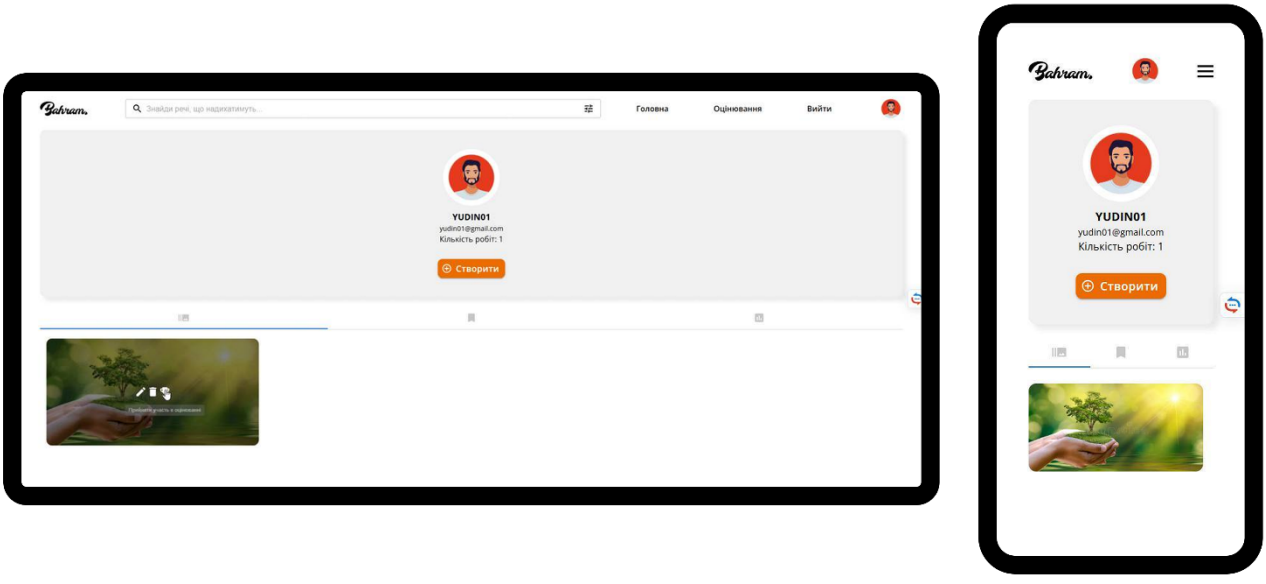

Рисунок 3.13 – Сторінка особистого кабінету застосунку

Сторінка «Оцінювання» призначена для взаємодії з публікаціями користувачів через систему оцінювання за кількома критеріями. У верхній частині розміщене велике зображення публікації, яку потрібно оцінити, з інтерактивними смайликами, що дозволяють швидко висловити загальне враження – позитивне чи негативне. Нижче міститься інформація про автора публікації та його позицію в рейтингу.

Під основним блоком розташовані картки постів із назвами, тегами та коротким описом. Біля кожної публікації є окрема кнопка «Оцінити», виділена яскравим кольором, яка відкриває можливість поставити оцінки за різними параметрами. Для кожного з цих елементів передбачені шкали для оцінювання.

Зліва і справа від зображення є стрілки для перегортання публікацій, що дозволяє легко переглядати інші роботи. Інтерфейс зручний і інтуїтивно зрозумілий, а велика кількість тегів допомагає швидко зрозуміти, до якої тематики належить конкретна публікація (див. рисунок 3.14).

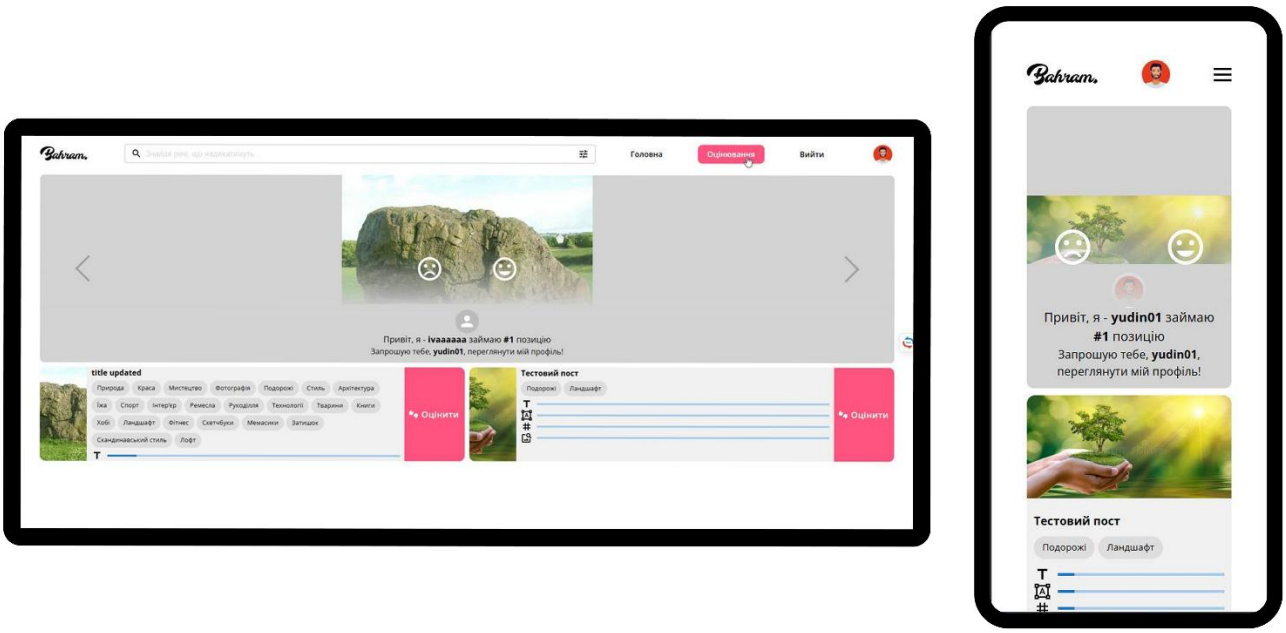

Рисунок 3.14 – Сторінка «Оцінювання» вебзастосунку

Загалом інтерфейс побудований таким чином, щоб бути доступним як для нових користувачів, так і для досвідчених. Кожен розділ логічно пов’язаний з іншим, що забезпечує цілісність користувацького шляху та ефективну навігацію.

Вебзастосунок поєднує в собі функціональність соціальної платформи, інструменту оцінювання контенту та системи мотивації, що робить його перспективним для подальшого розвитку.

3.6 Висновки

У процесі розробки програмного забезпечення вебсистеми було комплексно реалізовано всі необхідні компоненти, що забезпечують її ефективне функціонування, зручність використання та продуктивність. Завдяки ретельно проведеному варіантному аналізу було обґрунтовано вибір сучасного технологічного стека, який включає React, SCSS, Node.js, PostgreSQL та інші інструменти, що забезпечують високу продуктивність і гнучкість системи.

Було створено модулі для оцінювання зображень, обміну контентом, збору та візуалізації статистичних даних, кожен із яких виконує конкретну функціональну роль і доповнює загальну архітектуру застосунку. Модуль оцінювання дозволяє здійснювати багатокритеріальний аналіз із урахуванням вагових коефіцієнтів, що забезпечує об’єктивність та прозорість результатів. Механізм обміну зображеннями підвищує взаємодію між користувачами, а система статистики надає корисну аналітику авторам публікацій і адміністраторам платформи.

Інтерфейс вебзастосунку реалізовано з дотриманням принципів адаптивного дизайну, що гарантує його доступність на різних типах пристроїв. Логічна структура інтерфейсу – поділ на головну сторінку, сторінку оцінювання та особистий кабінет – сприяє простій навігації, підвищенню зручності користування та інтуїтивному розумінню функцій системи.

Таким чином, програмне забезпечення системи є функціональним, технологічно досконалим та орієнтованим на користувача, що створює міцну основу для подальшого розвитку платформи.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Якість програмного забезпечення безпосередньо залежить від ефективності процесу його тестування. Саме завдяки тестуванню розробники отримують можливість виявити критичні помилки на ранніх етапах, запобігти збоям у роботі системи та забезпечити надійність і відповідність продукту вимогам користувачів і бізнесу. У сучасних реаліях, коли програмні рішення стають дедалі складнішими та багатофункціональними, тестування відіграє ключову роль у підтримці конкурентоспроможності та довіри клієнтів.

Тестування програмного забезпечення – це систематичний процес перевірки програмного продукту, спрямований на виявлення дефектів і оцінку відповідності функціональних і нефункціональних характеристик заданим вимогам. Воно не лише дозволяє знизити ризики виходу на ринок недосконалого продукту, а й оптимізує витрати на подальшу підтримку та розвиток системи [36].

Існує велика різноманітність методів і видів тестування, кожен з яких має своє призначення і дозволяє оцінити різні аспекти якості ПЗ. Найбільш поширеним є функціональне тестування, яке перевіряє відповідність роботи програми очікуваним функціям та логіці. Водночас нефункціональне тестування спрямоване на оцінку таких характеристик, як продуктивність, безпека, зручність користування і сумісність з іншими системами.

На більш детальному рівні тестування розпочинається з модульного (unit testing), коли окремі компоненти коду перевіряються розробниками для своєчасного виявлення помилок. Далі проводиться інтеграційне тестування, що досліджує взаємодію між різними модулями. Системне тестування охоплює перевірку програмного продукту як цілого, а приймальне – підтверджує готовність системи до експлуатації замовником [37].

Окрему увагу приділяють регресійному тестуванню, яке гарантує, що нові зміни в коді не вплинули негативно на вже реалізований функціонал. Безпекове тестування дозволяє виявити уразливості, які можуть загрожувати цілісності та

конфіденційності даних. Тестування продуктивності і навантаження демонструє, як система працює в умовах реального або максимального навантаження.

Таким чином, комплексне застосування різних методів тестування дозволяє забезпечити всебічну оцінку якості програмного забезпечення і мінімізувати ризики, пов’язані з його використанням у реальному середовищі. Це робить тестування одним із найважливіших етапів у процесі розробки сучасних програмних систем.

4.2 Тестування розробленого програмного застосунку

На початковому етапі тестування програмного забезпечення було перевірено, як застосунок реагує на введення неправильних даних під час авторизації. Окрему увагу приділено ситуаціям, коли користувач залишає обов'язкові поля незаповненими (див. рисунок 4.1). У таких випадках система миттєво виводить повідомлення про необхідність заповнення відповідних полів, що запобігає надсиланню неповних даних. Це свідчить про наявність механізмів валідації даних на стороні клієнта та сервера, що суттєво підвищує безпеку і стабільність роботи вебсистеми.

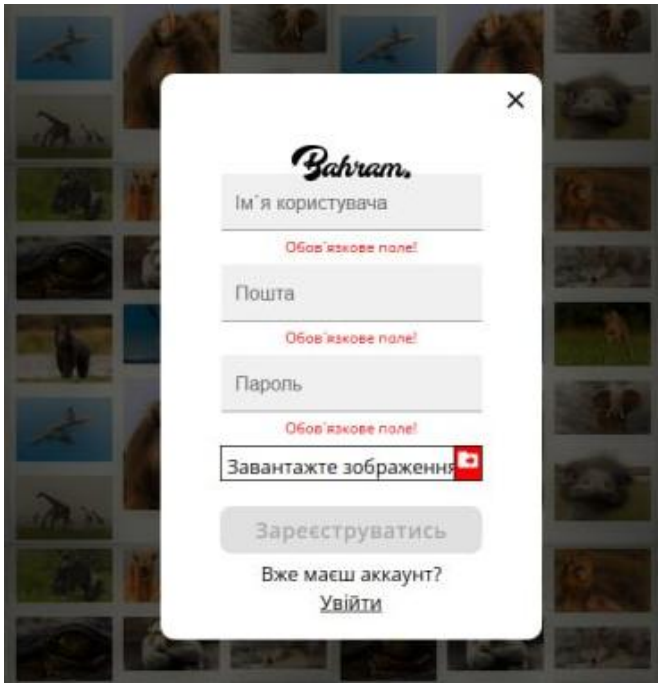

Рисунок 4.1 – Повідомлення про потребу заповнення обов'язкових полів

Додатково тестувалася обробка введення даних у неправильному форматі. Якщо користувач вводить неприпустимі символи або дані, що не відповідають очікуваним вимогам (наприклад, використання спеціальних знаків у полі «Ім'я» або неправильний формат електронної пошти), система коректно виявляє такі помилки та інформує про це. Також було перевірено, що у разі введення надто коротких даних система повідомляє користувача про мінімальну кількість символів, необхідну для заповнення конкретного поля. Якщо дані заповнені правильно, система дозволяє продовжити реєстрацію без зауважень (див. рисунок 4.2).

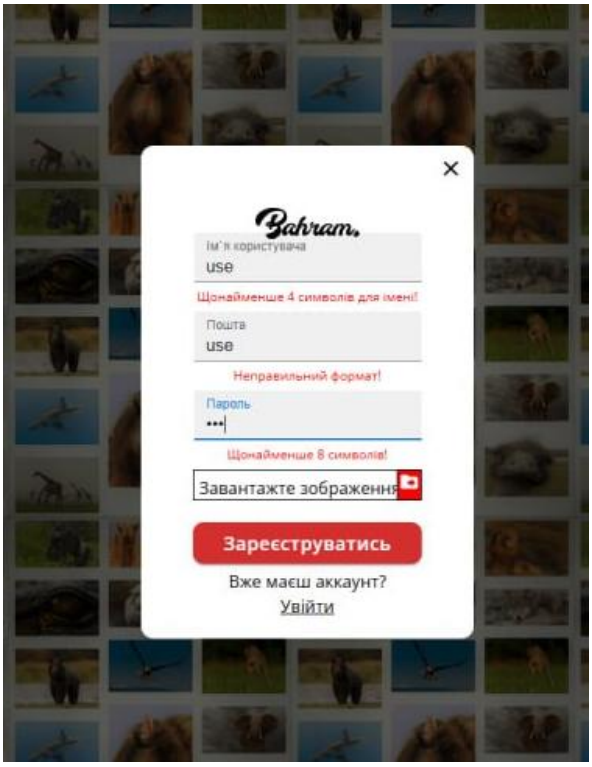

Рисунок 4.2 – Повідомлення про введення даних у невірному форматі

Далі перевірялося, як система реагує у випадку, якщо користувач заповнює всі обов'язкові поля коректними даними. Під час тестування було встановлено, що при правильному заповненні форми реєстрації або авторизації система не виводить жодних повідомлень про помилки. Усі введені дані успішно проходять валідацію, після чого користувач автоматично переходить до наступного етапу – особистий кабінет (див. рисунок 4.3).

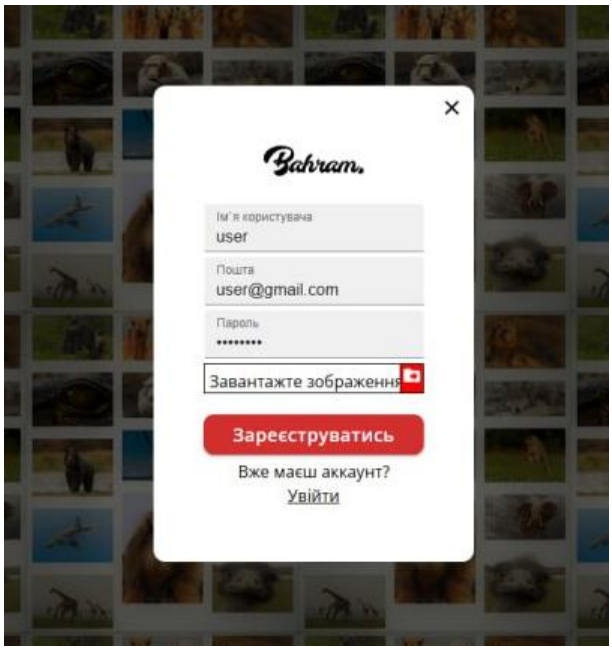

Рисунок 4.3 – Відсутність повідомлень про помилки при правильному заповненні

Після того, як користувач успішно увійшов у систему, було важливо протестувати основні функції взаємодії з публікаціями. Одним із ключових етапів було тестування функціоналу додавання публікації в розділ «Збережене». Після натискання кнопки, що відповідає за додавання в «Обране» публікація успішно додається до списку збережених публікацій, і система підтверджує операцію без помилок, відображаючи відповідне повідомлення про успішне додавання, а також оновлює список збережених публікацій в реальному часі (див. рисунок 4.4).

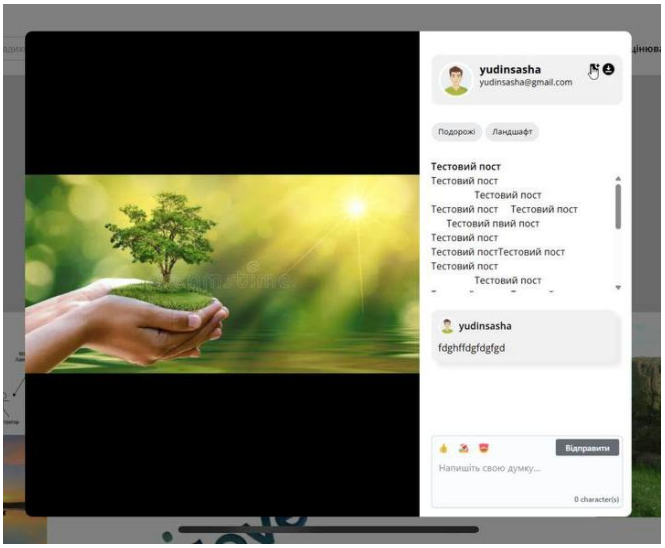

Рисунок 4.4 – Процес додавання публікації в «Збережене»

Наступним кроком було перевірено, чи коректно відображається збережена публікація в особистому кабінеті користувача. Після переходу на вкладку «Збережене», пост, який було додано, відображається у цьому розділі без затримок, що підтверджує правильність роботи функції збереження. Це дозволяє користувачам швидко отримати доступ до важливих публікацій (див. рисунок 4.5).

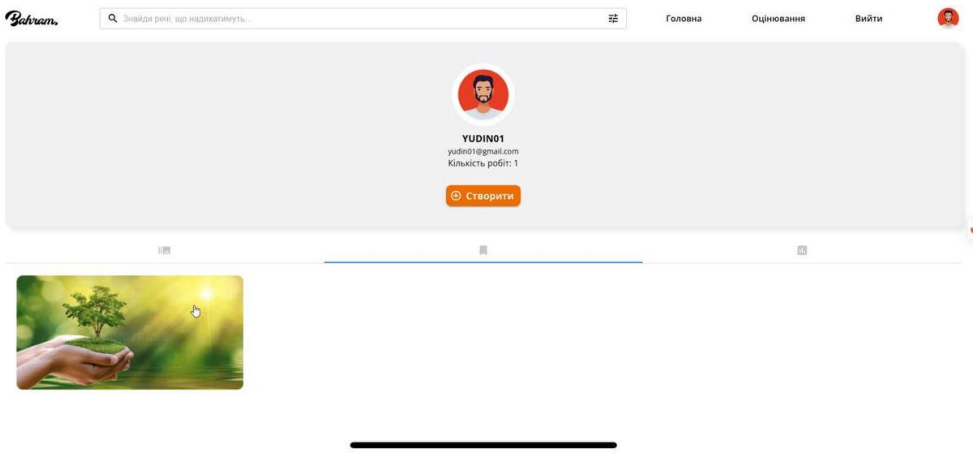

Рисунок 4.5 – Успішне додання публікації

Далі проводилось тестування функціоналу видалення публікацій з розділу «Збережене». Для цього натиснуто кнопку «Видалити з обраного», після чого публікація успішно видаляється із цього списку. Після видалення система коректно оновлює інформацію, і в особистому кабінеті відображається повідомлення про відсутність збережених публікацій, що підтверджує правильну роботу функції видалення (див. рисунок 4.6).

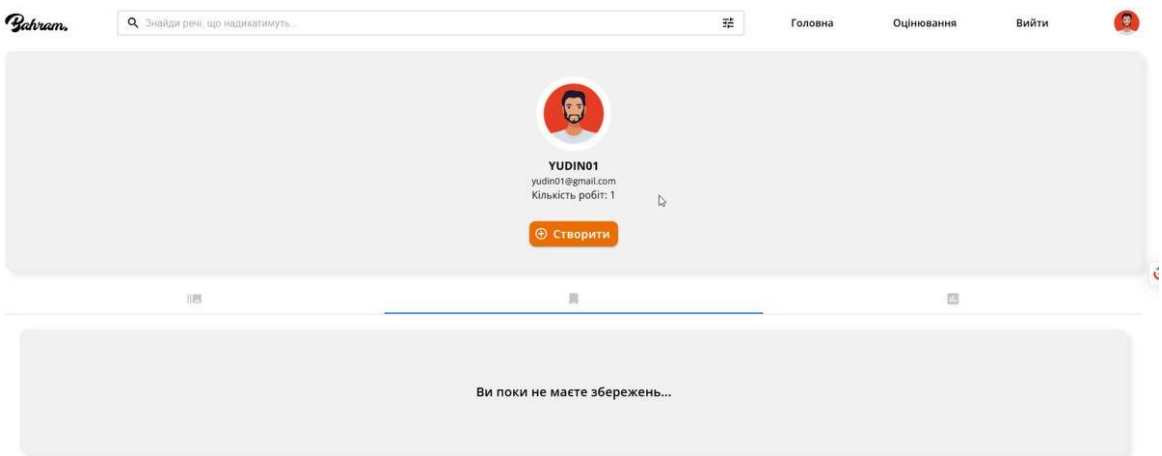

Рисунок 4.6 – Успішне видалення публікації

У деяких публікаціях встановлені обмеження щодо конфіденційності, і система вчасно попереджає користувача про це. Для таких постів, що мають підвищений рівень конфіденційності, було обмежено можливість взаємодії, зокрема, скачування контенту або залишення коментарів. У разі спроби здійснити ці дії, система виводить повідомлення, яке інформує, що доступ до публікації обмежений, а відповідні кнопки для завантаження чи коментування є недоступними.

Це забезпечує дотримання стандартів конфіденційності та безпеки, що особливо важливо для захисту приватної інформації. Крім того, система гарантує, що обмеження конфіденційності застосовуються на всіх етапах взаємодії з публікацією (див. рисунок 4.7).

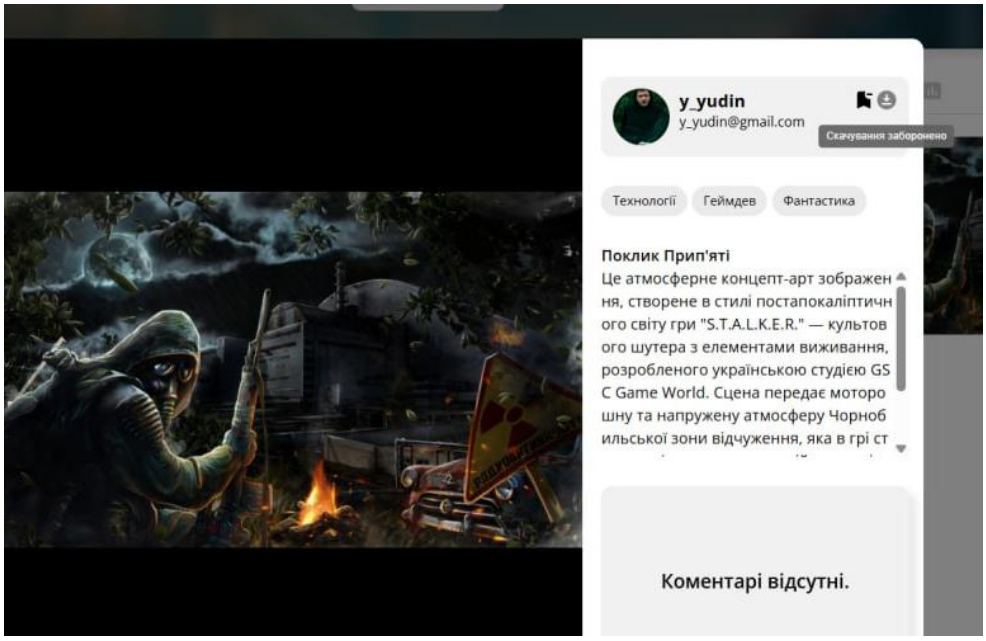

Рисунок 4.7 – Публікація з обмеженням щодо конфіденційності

Однак для публікацій з відкритим доступом або з низьким рівнем конфіденційності система дозволяє здійснювати повну взаємодію з контентом. У таких випадках кнопка для завантаження доступна, а також можливо залишати коментарі, що дозволяє користувачам обговорювати публікації або зберігати контент для подальшого використання (див. рисунок 4.8).

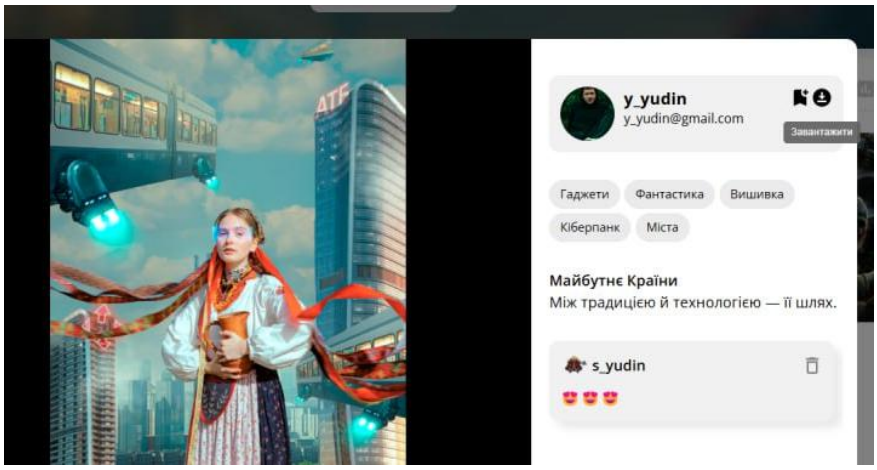

Рисунок 4.8 – Публікація без обмежень щодо конфіденційності

Під час тестування функціоналу створення поста було перевірено, чи система коректно обробляє ситуації, коли не всі поля заповнені. Якщо користувач намагається опублікувати пост, не заповнивши обов'язкові поля, система виводить відповідне повідомлення про помилку, інформуючи, що ці поля є обов'язковими для заповнення (див. рисунок 4.9).

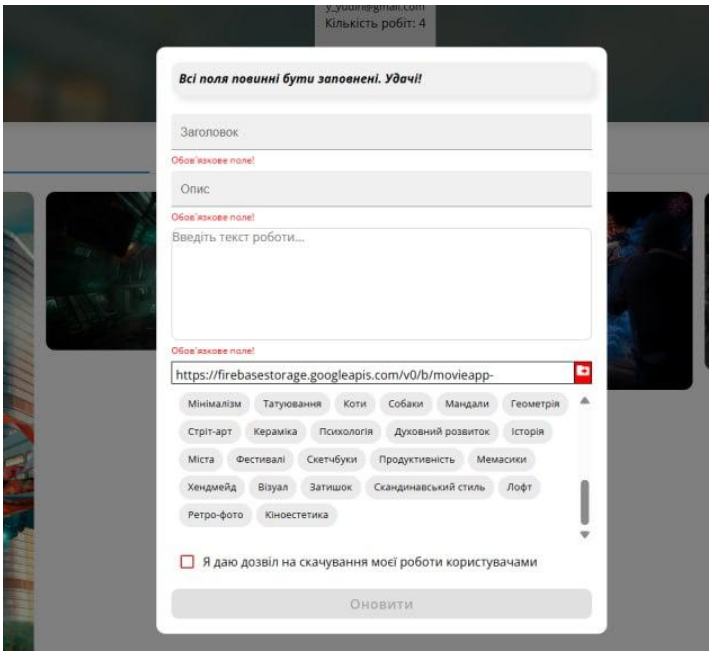

Рисунок 4.9 – Повідомлення про необхідність заповнення обов'язкових полів

У випадку, коли всі обов'язкові поля були коректно заповнені, система не виводить помилок, і публікація успішно створюється. Після натискання кнопки

«Опублікувати» публікація додається до списку доступних постів без будь-яких додаткових повідомлень про помилки (див. рисунок 4.10).

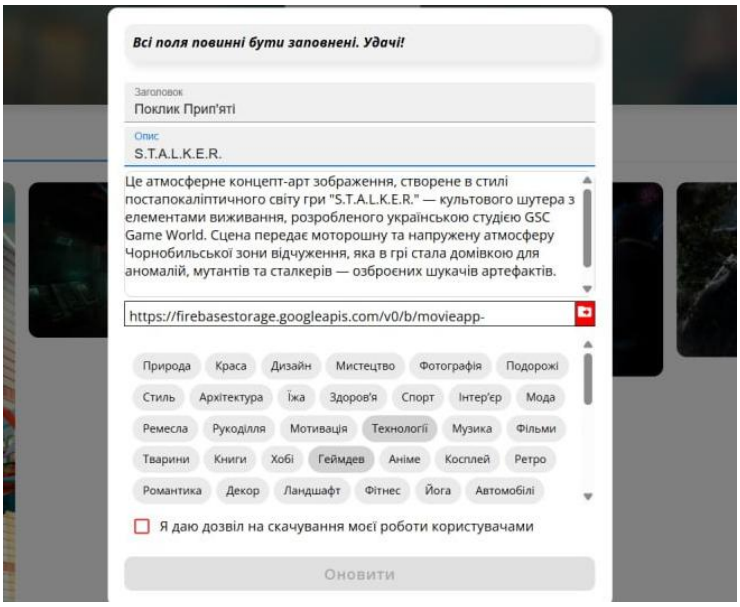

Рисунок 4.9 – Всі поля успішно заповнені

Модуль, що відповідає за участь у оцінюванні фото, є важливою частиною системи, оскільки забезпечує точність та контроль над процесом оцінки контенту. Щоб запобігти випадковому натисканню кнопки для оцінки фото, система виводить підтверджувальне вікно, яке запитує користувача, чи дійсно він хоче взяти участь у оцінюванні (див. рисунок 4.10).

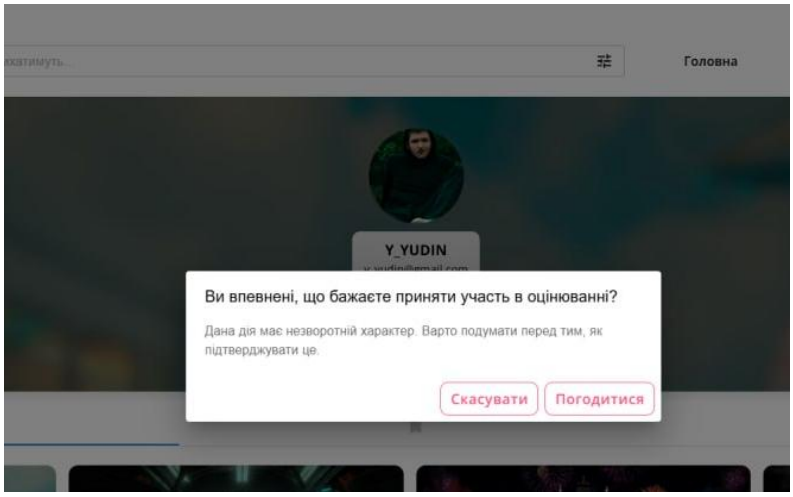

Рисунок 4.10 – Підтвердження участі в оцінюванні фото

Після того як користувач підтвердив свою участь в оцінюванні, його фото автоматично додається до списку зображень, доступних для оцінювання. Після переходу на відповідну сторінку система коректно відображає це фото серед інших, що беруть участь у процесі (див. рисунок 4.11).

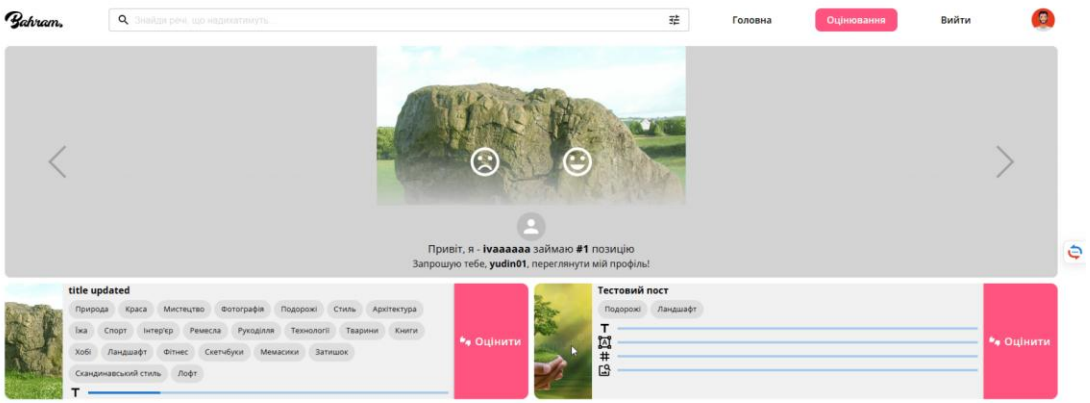

Рисунок 4.11 – Сторінка оцінювання фото

Після того як користувач натискає кнопку «Оцінити», з’являється модальне вікно, у якому відображено різні категорії для оцінювання (див. рисунок 4.12). Зокрема, система пропонує виставити оцінки за такими критеріями: заголовок, наповнення, відповідність тегів, зображення та аналітика**.** Кожна категорія має власну шкалу оцінювання, що дозволяє сформувати більш об’єктивну та детальну оцінку фото.

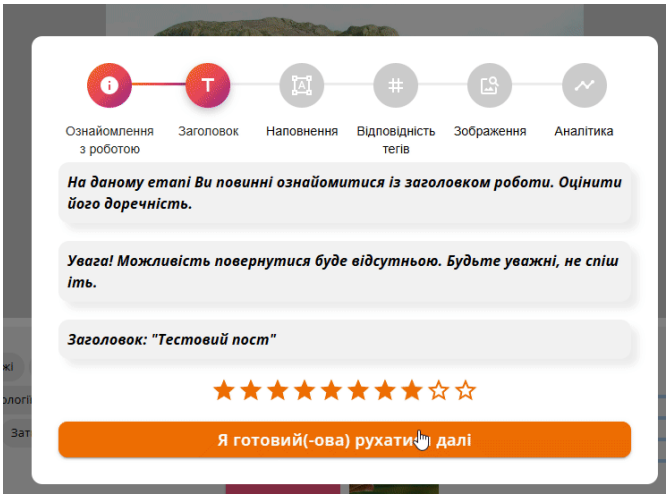

Рисунок 4.12 – Категорії оцінювання фото

Функціональність пошуку в системі є важливою для швидкого знаходження потрібних публікацій серед великої кількості контенту. Пошук дозволяє користувачам вводити ключові слова або фрази, що відповідають темі поста, і знаходити відповідні результати. У разі, якщо введений запит співпадає з назвою, тегами або описом публікацій, система виводить список результатів, що відповідають пошуковому запиту (див. рисунок 4.13).

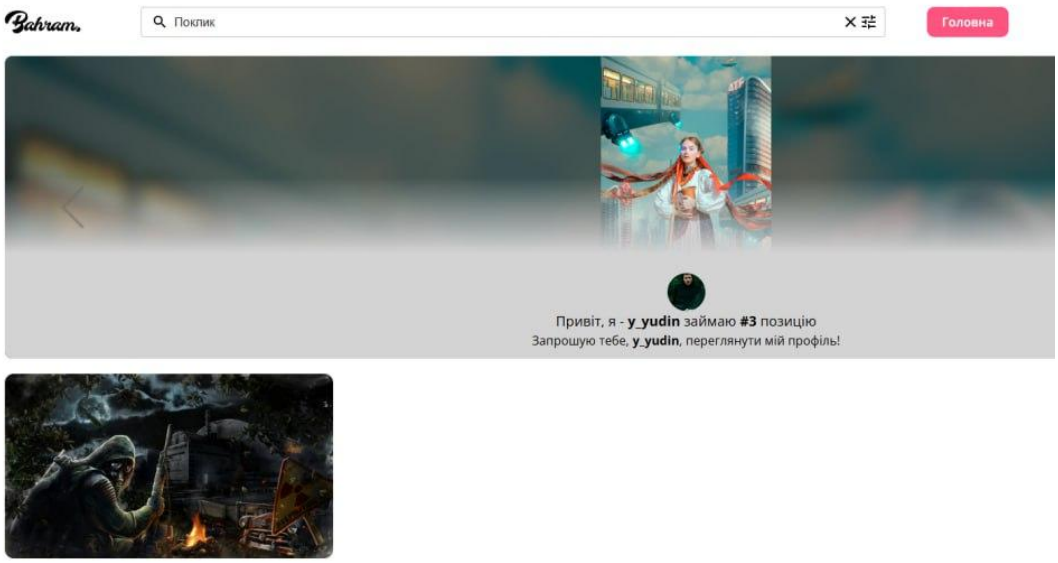

Рисунок 4.13 – Успішний результат пошуку по публікаціях

У разі, якщо за введеним запитом не знайдено жодних результатів, система коректно повідомляє користувача, що жодні публікації не відповідають цьому запиту (див. рисунок 4.14).

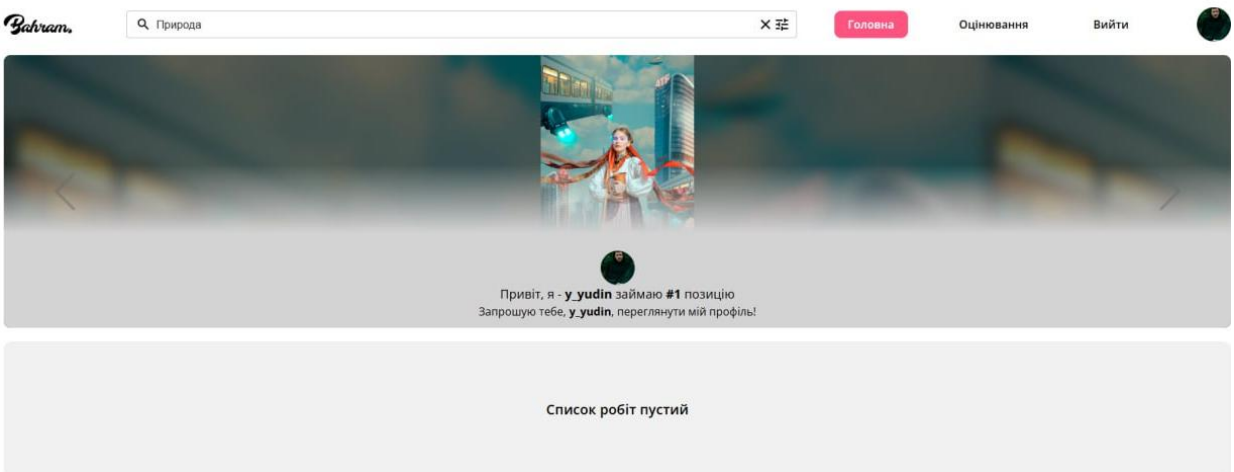

Рисунок 4.14 – Повідомлення про відсутність результатів пошуку

Система надає можливість користувачам залишати відгуки до публікацій, що є корисною та зручною функцією для зворотного зв’язку. Це дозволяє підтримувати комунікацію між користувачами, сприяє обговоренню контенту та підвищує залученість аудиторії. Процес написання коментаря зображено на рсиунку 4.15.

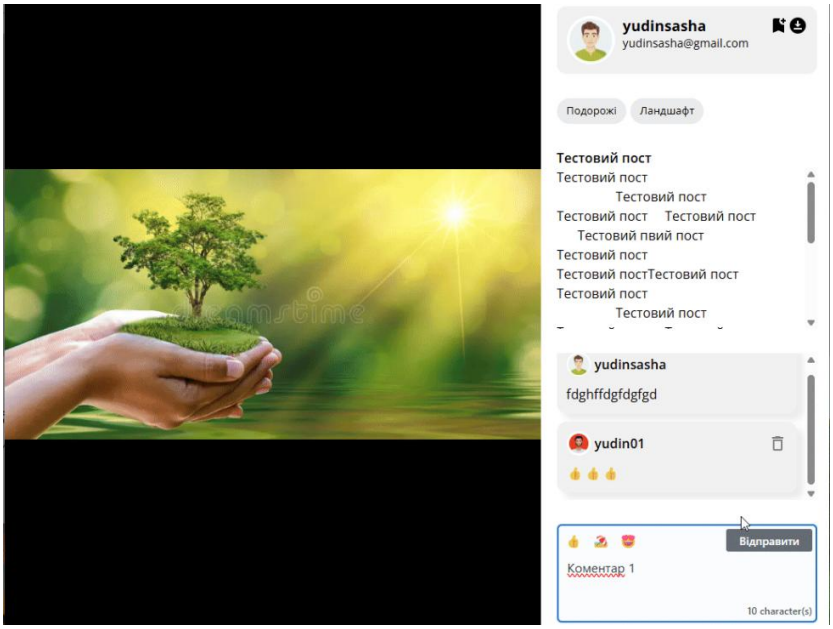

Рисунок 4.15 – Процес написання коментаря «Коментар 1»

У рамках перевірки функціональності цієї можливості було вирішено протестувати, чи коректно працює функція надсилання та публікації коментаря. Після публікації відгуку перевірено, чи зберігається коментар у системі. Результат тестування зображено на рисунку 4.16.

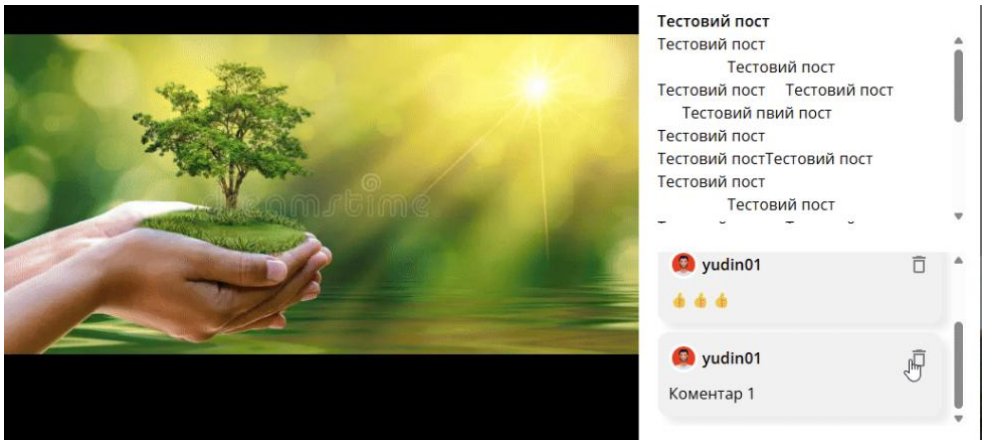

Рисунок 4.16 – Коментар успішно опублікований

Також важливо перевірити чи зникає з інтерфейсу коментар після його видалення. Результати тестування показали, що система успішно обробляє цю дію: після видалення коментарі зникають як із бази даних, так і з інтерфейсу, без залишкових помилок або повідомлень (див. рисунок 4.17).

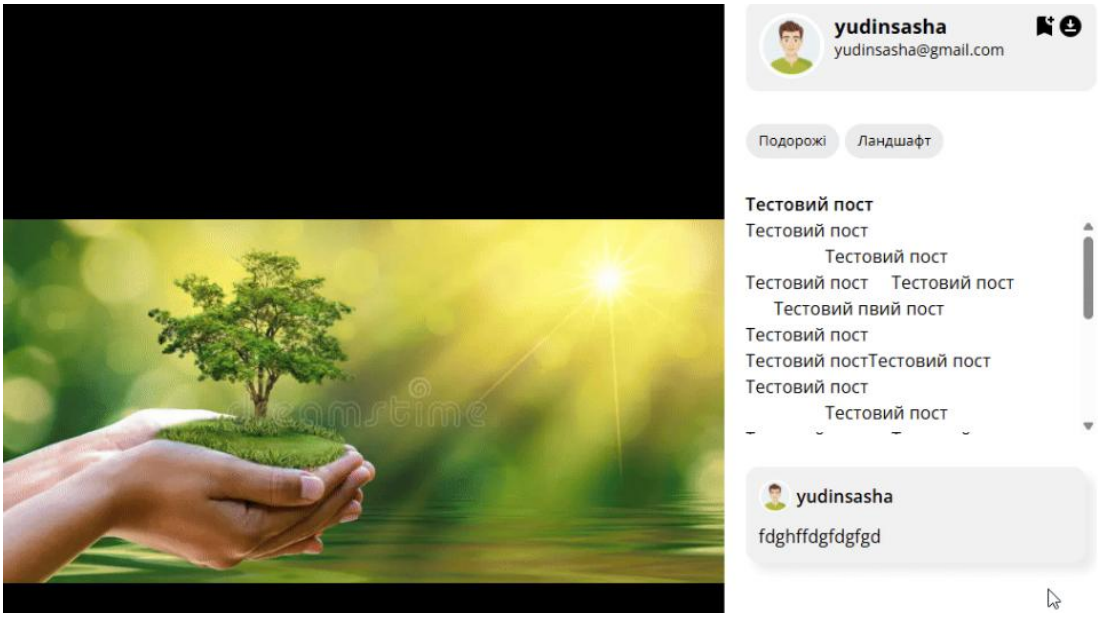

Рисунок 4.17 – Коментар успішно видалений

Важливим етапом у роботі системи є процес створення публікації та її додавання на сторінку користувача. Ця функція відіграє ключову роль у взаємодії користувача з платформою, оскільки дозволяє ділитися власним контентом і формувати інформаційний потік.

Створення публікації відбувається за допомогою спеціальної кнопки, розміщеної на інтерфейсі створення поста. Після натискання на неї користувач отримує можливість обрати файл із зображенням, можливо з будь-якого пристрою – смартфона, планшета чи комп’ютера. Обрані фото завантажуються до системи, після чого користувач має змогу переглянути їх попередній вигляд, додати опис, та підтвердити свій намір опублікувати матеріал (див. рисунок 4.18). Завершальним кроком є натискання кнопки «Створити», яка ініціює процес публікації контенту на сторінці.

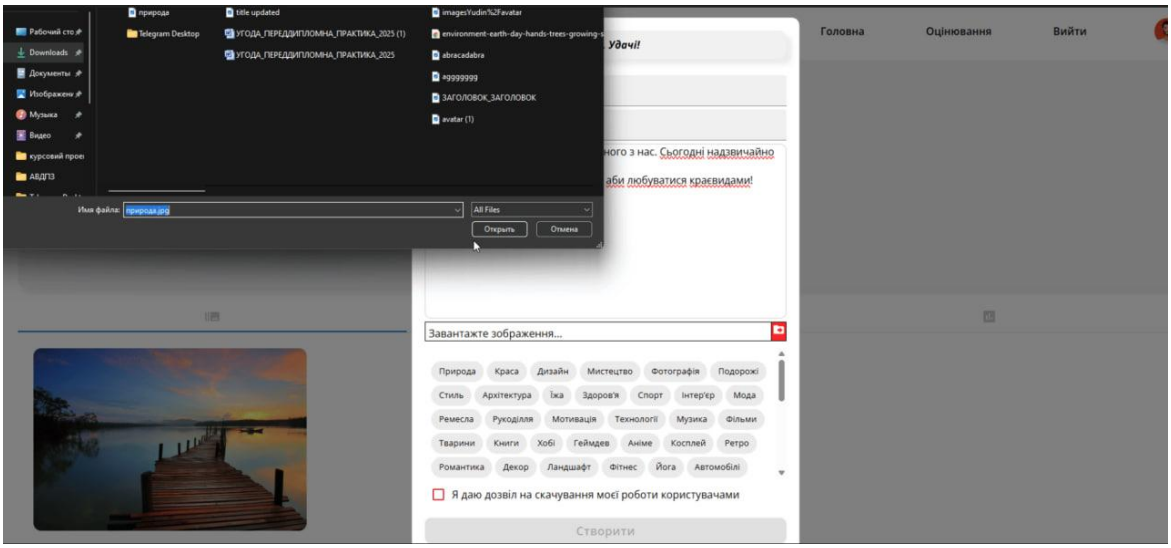

Рисунок 4.18 – Процес вибору фото з особистого файлового провідника

Після створення публікації було перевірено, чи зображення коректно відображається в галереї користувача у власному кабінеті. Здійснено перехід на відповідну сторінку профілю, де має відображатися щойно опубліковане фото. Результати тестування показали, що система успішно зберігає та виводить зображення в загальній стрічці користувача, забезпечуючи його доступність для перегляду, редагування або видалення (див. рисунок 4.19).

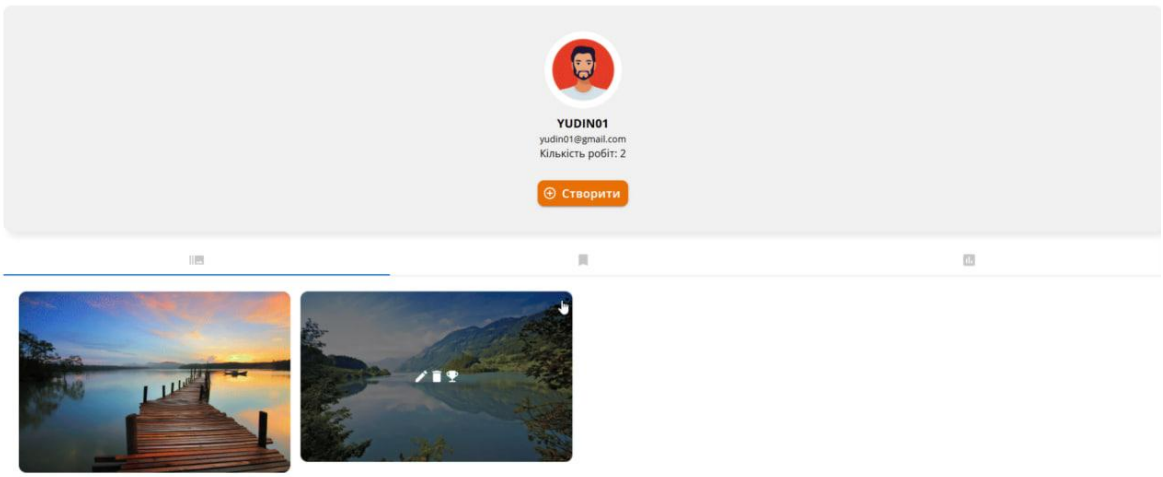

Рисунок 4.19 – Успішна публікація фото

Функція обміну є надзвичайно важливою у структурі вебзастосунку, оскільки вона забезпечує інтерактивність між користувачами та створює динамічне

середовище взаємодії. Завдяки цій функції користувачі можуть обмінюватися своїм контентом, що підвищує зацікавленість і залучення до платформи.

У рамках тестування було проаналізовано, які публікації на даний момент має користувач, зокрема переглянуто зображення, що доступні для обміну в особистому кабінеті (див. рисунок 4.20).

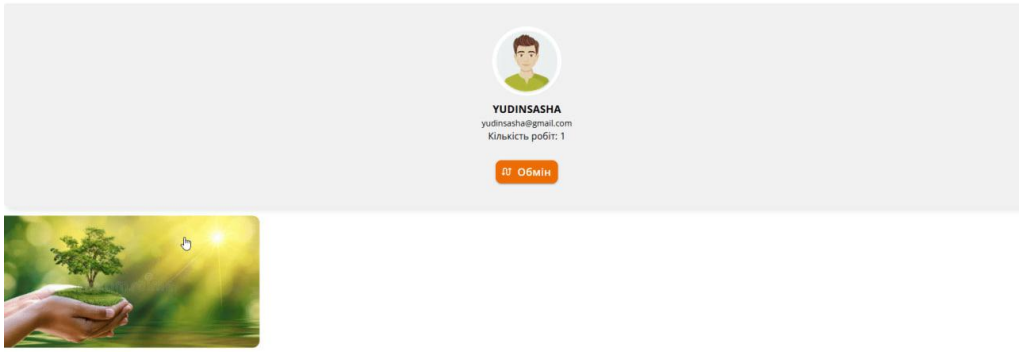

Рисунок 4.20 – Галерея власних публікацій

Надалі було розпочато безпосередній процес обміну фотографіями. Він передбачає вибір власного фото, яке користувач хоче запропонувати для обміну, та вибір фото іншого користувача зі списку доступних публікацій. Після цього натискається кнопка «Обмінятись» (див. рисунок 4.21). Після підтвердження обміну система автоматично оновлює сторінку, і в обох користувачів відбувається обмін зображеннями – тобто, кожен отримує нове фото, а своє віддає остаточно.

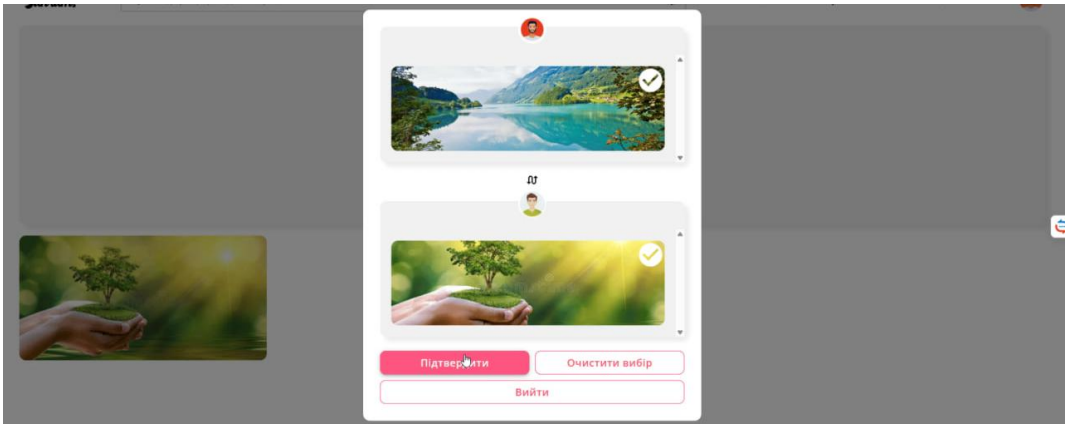

Рисунок 4.21 – Процес обміну публікаціями

Після завершення обміну в особистому кабінеті користувача з’явилося нове зображення – саме те, яке було вибрано в іншого користувача, натомість його власне фото зникло з галереї. Це підтверджує правильність виконання логіки обміну та стабільну роботу відповідного функціоналу без помилок або дублювання контенту (див. рисунок 4.22).

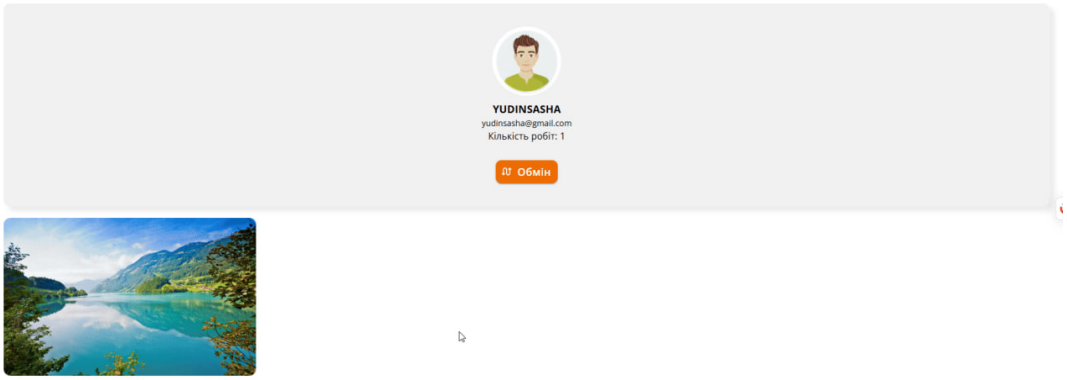

Рисунок 4.22 – Підтвердження успішного обміну

Також завершальним етапом тестування було перевірено правильність відображення статистики щодо публікацій (див. рисунок 4.23). Зокрема, було проаналізовано, чи коректно оновлюються дані про кількість оцінок, середній бал, кількість переглядів тощо. Всі перевірені показники відображалися відповідно до очікувань, що підтвердило правильність роботи системи збору та виведення статистичних даних.

Рисунок 4.23 – Відображення статистики публікацій

Проведене тестування підтвердило коректну роботу основного функціоналу вебзастосунку, зокрема авторизації, публікації зображень, функціоналу коментування та зберігання, їх обміну, оцінювання та відображення статистики. Усі перевірені процеси працювали стабільно й відповідали логіці системи. Це свідчить про високий рівень узгодженості компонентів застосунку та сприяє підвищенню його якості, надійності й загальної продуктивності.

4.3 Розробка інструкцій користувача

Створення інструкції користувача є невіддільною складовою процесу розробки будь-якої програмної системи або вебплатформи. Така інструкція виконує важливу роль у забезпеченні користувачів необхідними відомостями для зручного та правильного використання функціоналу продукту. Це, у свою чергу, підвищує рівень задоволеності користувачів і зменшує ймовірність помилок під час взаємодії з інтерфейсом системи [38].

Документація для користувача формулює послідовні, чіткі дії, які дозволяють легко опанувати роботу із системою навіть тим, хто не має спеціальної підготовки. Особливо це важливо у випадках впровадження нових або технічно складних продуктів, де без інструкцій користувачі можуть стикнутися з труднощами в навігації або виконанні базових операцій.

Робота з вебзастосунком починається з підготовки його до продакшн-середовища. Для цього запускається команда збірки (npm run build), яка компілює вихідний код і генерує оптимізовану версію застосунку. У результаті формується папка (build), що містить статичні файли HTML, CSS, JavaScript та інші ресурси, необхідні для роботи застосунку в браузері. Далі вміст цієї папки необхідно задеплоїти на хостинг-платформу. Після успішного розгортання застосунок стає доступним за вказаною URL-адресою і готовий до використання кінцевими користувачами.

Перед встановленням важливо впевнитися, що пристрій відповідає мінімальним технічним вимогам програми. Це дозволить уникнути збоїв або некоректної роботи. Всі необхідні параметри системної сумісності викладено в

таблицях 4.1, що дозволяє користувачам легко зорієнтуватися в потребах застосунку.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурації	Рекомендовано	Мінімально
Процесор	Intel Core i5 10-gen / AMD Ryzen 5	Intel Core i3 6-gen / AMD Ryzen 3
Оперативна пам'ять	6 ГБ RAM	3 ГБ RAM
Вільний простір на диску	2ГБ	500МБ
Операційна система	Windows 10 / 11, Linux, macOS	Windows 7
Дисплей	Роздільна здатність не менше 1080x2340 пікселів (Full HD+)	Роздільна здатність не менше 720x1520 пікселів (HD+)

Для досягнення найкращої продуктивності та стабільності роботи системи рекомендується використовувати апаратне забезпечення, яке відповідає рекомендованим вимогам. Однак для базового використання продукту, де не передбачено високих вимог до швидкості або навантаження, можна обмежитися мінімальними характеристиками [38].

Після входу в систему користувач потрапляє до особистого кабінету, де є доступ до основних функцій управління публікаціями.

Взаємодія користувача з постами є основною функцією вебдодатку. Користувач може додавати новий пост натиснувши кнопку «Створити», заповнивши всі обов'язкові поля (заголовок, опис, теги, фото) та натискаючи кнопку «Опублікувати» (див. рисунок 4.24). Система перевіряє, чи всі поля заповнені правильно. Якщо деякі обов'язкові поля не заповнені, система виведе

попередження, і користувач повинен виправити помилку, перш ніж публікація буде завершена.

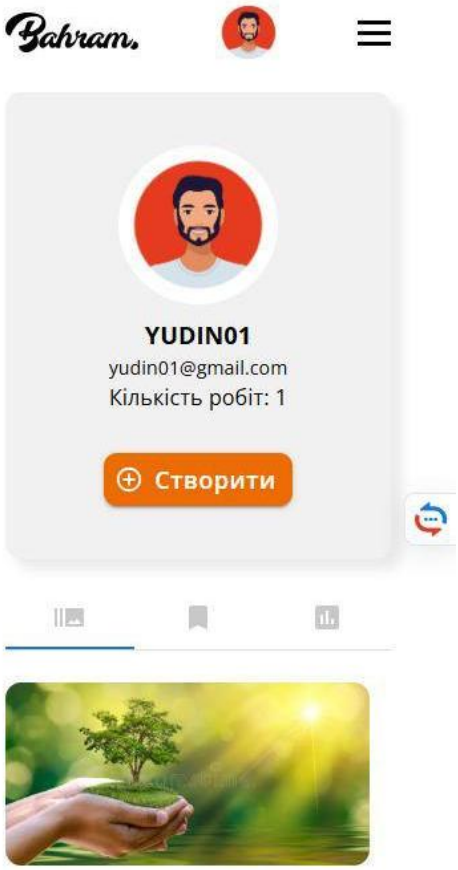

Рисунок 4.24 – Відображення кнопки «Створити»

Після публікації поста користувач може його редагувати або видалити. Для цього потрібно навести на публікацію, обрати та натиснути на відповідну кнопку («Редагувати» або «Видалити») поруч з публікацією (див. рисунок 4.25).

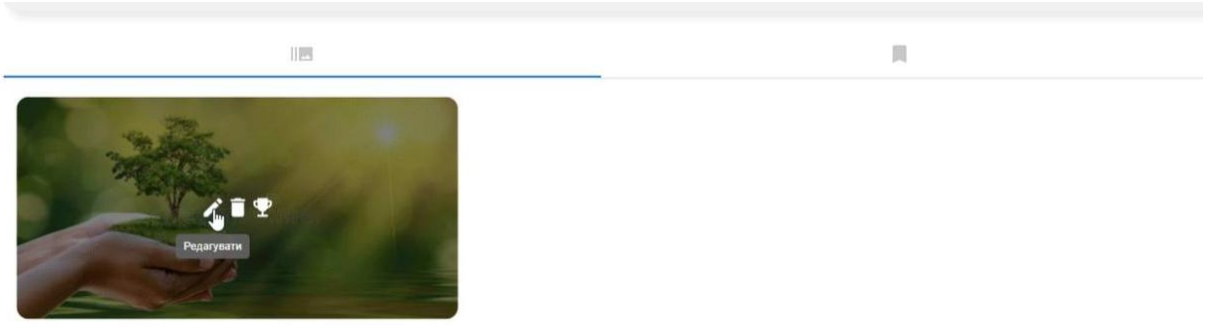

Рисунок 4.25 – Взаємодія з постом після публікації

Для обміну публікаціями з іншими користувачами необхідно натиснути кнопку «Обмін» в особистому кабінеті, після потрібно обрати свою публікацію для обміну та іншого користувача. Після натискання користувач підтверджує свою дію. Коли обмін підтверджено, публікація буде змінена: на місце попереднього поста з'явиться публікація іншого користувача, а публікація, що обмінялась, буде видалена з особистої сторінки (див. рисунок 4.26).

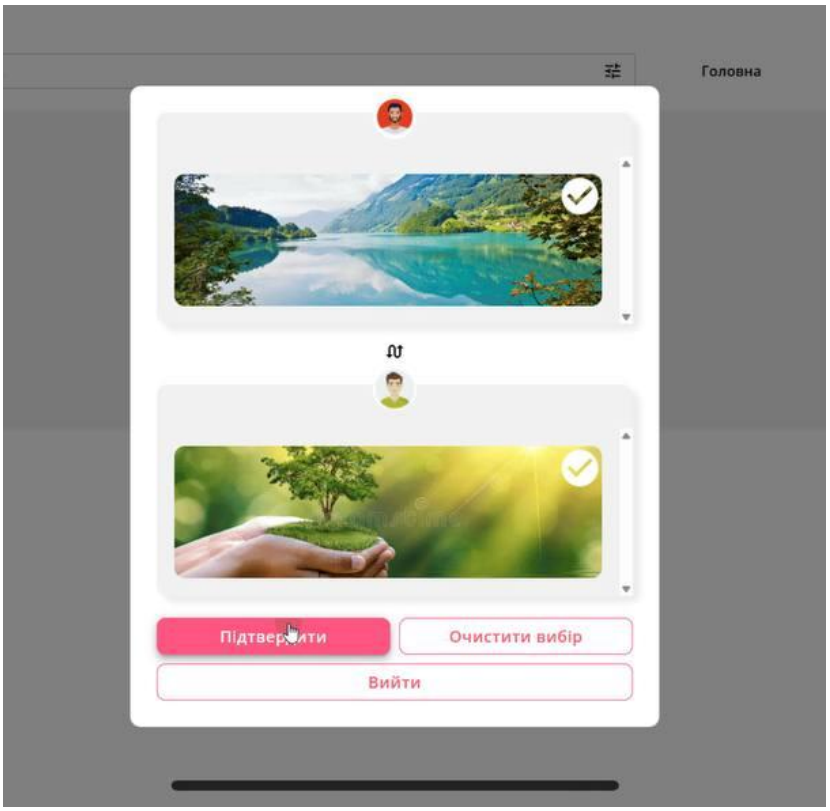

Рисунок 4.26 – Процес обміну публікаціями між користувачами

Також користувач може взяти участь в оцінюванні публікацій інших користувачів. Для цього потрібно навести курсор на власне фото, обраючи кнопку «Взяти участь в оцінюванні» та підтвердити участь. Після чого з'являється можливість оцінити фото іншого користувача, який теж бере участь за різними критеріями: назва, опис, теги, візуалізація тощо.

Оцінювання здійснюється шляхом присвоєння балів за допомогою зірочок: користувач ставить від 0 до 10 зірок для кожного з критеріїв. Після завершення оцінювання система автоматично обчислює середній бал за кожним критерієм та

відображає результат публікації. Таким чином, користувачі можуть побачити, як їхні публікації оцінюються іншими учасниками, а також отримати зворотний зв'язок для покращення своїх матеріалів (див. рисунок 4.27).

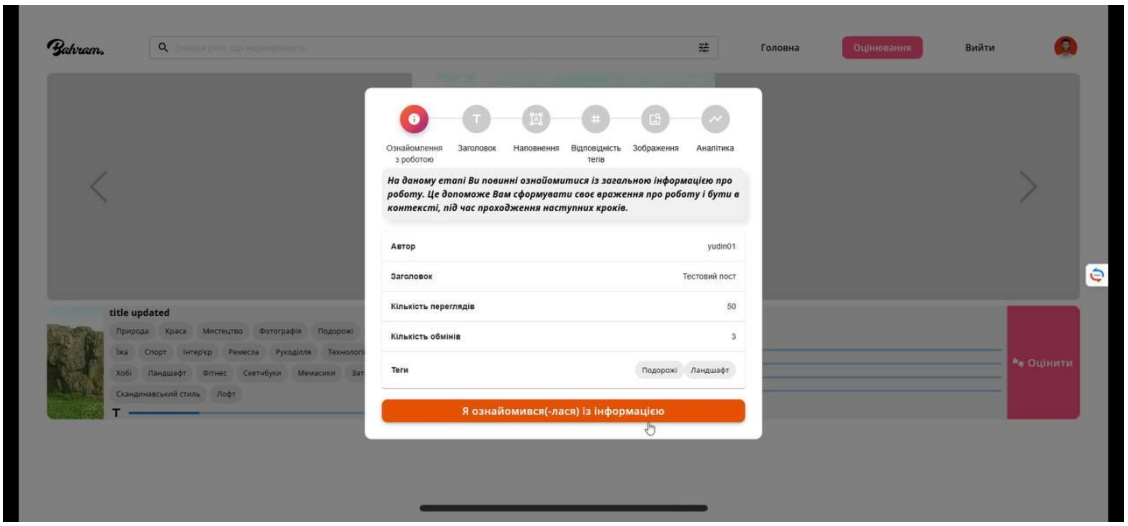

Рисунок 4.27 – Процес участі в оцінюванні

На будь-якому етапі користувач може вийти з системи. Для цього необхідно натискати кнопку «Вийти» в меню профілю. Після цього користувач буде перенаправлений на екран входу (див. рисунок 4.28).

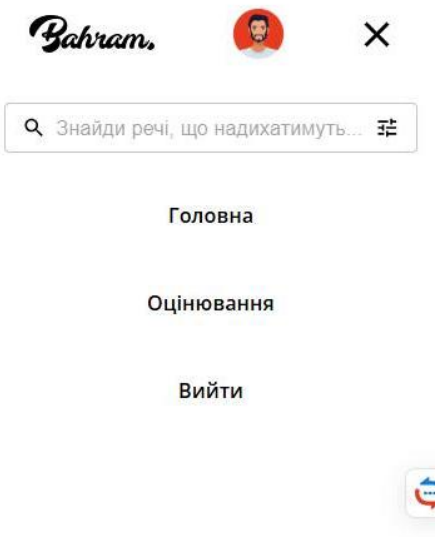

Рисунок 4.28 – Навігаційне меню вебсистеми

Рекомендується завжди виходити з системи після завершення роботи, особливо на загальнодоступних пристроях, щоб запобігти несанкціонованому доступу та захистити персональні дані користувача. Завдяки інтуїтивному інтерфейсу та зрозумілому функціоналу, даний застосунок є доступним для широкого кола користувачів. Навіть особи без технічного досвіду можуть швидко освоїти його можливості й ефективно ним користуватись.

4.4 Висновки

У результаті проведеного тестування було всебічно перевірено ключові функції вебзастосунку. Кожен з протестованих етапів продемонстрував стабільну та логічну роботу системи без виявлених критичних помилок. Система коректно реагує на дії користувача, своєчасно оновлює дані, забезпечує зручний інтерфейс взаємодії та дозволяє ефективно виконувати всі основні дії, передбачені логікою роботи застосунку.

Особливо важливо відзначити, що реалізація таких функцій, як обмін фотографіями, модуль оцінювання та автоматичне формування статистики, значною мірою підвищує рівень якості програмного забезпечення. Вони створюють інтерактивне середовище, покращують користувацький досвід і забезпечують продуктивну роботу всієї системи.

Інструкція користувача була розроблена для полегшення процесу взаємодії з вебсистемою. Вона включає покрокові вказівки для виконання основних операцій, таких як реєстрація, входження в систему, додавання та редагування публікацій, а також збереження контенту. Інструкція також описує, як працювати з функцією пошуку та фільтрації публікацій, а також як налаштувати приватність окремих публікацій.

Таким чином, проведене тестування не лише підтвердило правильність роботи основного функціоналу, але й продемонструвало високий рівень узгодженості між елементами інтерфейсу та логікою обробки даних, що у підсумку сприяє підвищенню загальної якості та продуктивності вебзастосунку.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено методи та засоби реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень. Реалізація проєкту відповідає методичним вказівкам [40]. Для реалізації рішення використовувалися сучасні вебтехнології, зокрема JavaScript і React для фронтенду, а також Node.js для серверної частини.

Під час виконання роботи було проведено аналіз існуючих рішень у сфері обробки та оцінювання зображень. Було розглянуто популярні сервіси, визначено їхні обмеження та здійснено порівняння з розробленою вебсистемою. Це дозволило чітко сформулювати цілі та завдання.

У процесі реалізації обґрунтовано вибір технологій: React для створення швидкого, адаптивного та інтерактивного інтерфейсу користувача, SCSS для стилізації компонентів, Chart.js для побудови графіків, Zustand як легкий і ефективний менеджер стану, а також Axios для реалізації HTTP-запитів. Для бекенд-частини використано Node.js із системою авторизації на базі токенів, а для зберігання даних – реляційну базу даних PostgreSQL. Обраний стек технологій забезпечив високу продуктивність, масштабованість і гнучкість розробленої системи.

У межах бакалаврської кваліфікаційної роботи було виконано наступне:

– здійснено аналіз стану питання та методів розв’язання задачі;

– спроектовано модель даних та структуру вебзастосунку;

– розроблено метод оцінювання якості зображення;

– розроблено метод управління обміном статичних зображень;

– розроблено алгоритм взаємодії користувача з особистим кабінетом;

– розроблено алгоритм взаємодії користувача з публікаціями;

– розроблено алгоритм оцінювання якості зображення;

– розроблено програмне забезпечення модуля оцінювання статичних зображень;

– розроблено програмне забезпечення модуля управління обміном статичних даних;

– розроблено програмне забезпечення модуля для збирання, обробки та візуалізації статистичних даних;

– проведено тестування програмного застосунку;

– розроблено інструкцію користувача та описати системні вимоги програми для візуалізації, аналізу й оцінювання статичних зображень.

У результаті виконаної було створено повнофункціональний вебзастосунок, що дозволяє оцінювати, обмінювати та візуалізувати статичні зображення. Реалізація алгоритмів взаємодії з користувачем, оцінювання якості зображень та модулів статистики підвищила ефективність і точність аналізу. Це суттєво покращило загальну якість програмного забезпечення, зробило його зручним у використанні та готовим до практичного застосування.

Перед завершенням роботи було проведено комплексне тестування функціоналу системи. Перевірялися всі основні сценарії взаємодії користувача, зокрема робота із заповненими й незаповненими формами, обробка некоректно введених даних, коректність авторизації, завантаження та оцінювання зображень. Система продемонструвала стійку роботу: усі основні процеси, включно з реєстрацією користувачів, взаємодією із зображеннями та управлінням профілем, функціонували стабільно, без виявлених критичних помилок або збоїв. Тестування підтвердило надійність обробки даних і відповідність роботи інтерфейсу очікуваній логіці взаємодії.

Для підвищення зручності використання вебплатформи також було підготовлено детальне керівництво користувача. У ньому описано процес реєстрації, авторизації, завантаження та оцінювання зображень, а також наведено інструкції щодо управління особистими даними й перегляду статистики в системі.

Результати тестування підтвердили високу якість реалізації функцій, адаптивність інтерфейсу під різні пристрої та зручність користувацької взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войтко В.В. Розробка методів та інструментів вебсистеми візуалізації, аналізу та оцінювання статичних зображень / В.В. Войтко, С В. Бевз, С.М. Бурбело, О.С. Юдін // Тези доповідей XV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології» м. Житомир, 28-29 березня 2025 р. – Житомир: Житомирська політехніка, 2025. – С. 266-267).

2. Войтко В.В. Розробка методів та інструментів для впровадження вебсистеми візуалізації, аналізу та оцінювання статичних зображень / В.В. Войтко, А.В. Денисюк, О.В. Гаврилюк, Н.Є. Барчук, О.С. Юдін // Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025): збірник доповідей [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/.

3. The Evolution of Websites_From Static Pages to Dynamic Interactivity [Електронний ресурс] – Режим доступу до ресурсу: https://www.slideshare.net/slideshow/the-evolution-of-websites_-from-static-pages-to-dynamic-interactivity-pdf/276117491 (дата звернення: 03.05.2025).

4. The Business Research Company [Електронний ресурс] – Режим доступу до ресурсу: https://www.thebusinessresearchcompany.com/report/photo-sharing-global-market-report (дата звернення: 03.05.2025).

5. Crackitt. State of Visual Content Marketing: The Statistics [Електронний ресурс] – Режим доступу до ресурсу: https://www.crackitt.com/state-of-visual-content-marketing-videos-images-statistics/ (дата звернення: 06.05.2025).

6. ImgPost [Електронний ресурс] – Режим доступу до ресурсу: https://imgpost.net/ (дата звернення: 06.05.2025).

7. Unsplash [Електронний ресурс] – Режим доступу до ресурсу: https://unsplash.com/ (дата звернення: 06.005.2025).

8. Pexels [Електронний ресурс] – Режим доступу до ресурсу: https://www.pexels.com/uk-ua/ (дата звернення: 08.05.2025).

9. Pixabay [Електронний ресурс] – Режим доступу до ресурсу: https://pixabay.com/ (дата звернення: 09.05.2025).

10. Software Architecture Patterns: What Are the Types and Which Is the Best One for Your Project [Електронний ресурс] – Режим доступу до ресурсу: https://www.turing.com/blog/software-architecture-patterns-types (дата звернення: 11.05.2025).

11. Handling image uploads with PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://cloudinary.com/blog/guest_post/handling-image-uploads-with-postgres-in-redwoodjs (дата звернення: 11.05.2025).

12. Використання UML діаграм для аналізу і проектування інформаційних систем [Електронний ресурс] – Режим доступу до ресурсу: https://pns.hneu.edu.ua/pluginfile.php/321048/mod_resource/content/3/%D0%92%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D0%BD%D1%8F%20UML%20%D0%B4%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC.pdf (дата звернення: 12.05.2025).

13. Use Case Diagram Relationships Explained with Examples [Електронний ресурс] – Режим доступу до ресурсу: https//creately.com/blog/diagrams/use-case-diagram-relationships/ (дата звернення: 14.05.2025).

14. Різниця між UI та UX [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/difference-between-ui-and-ux/ (дата звернення: 14.05.2025).

15. Джеф Ґотельф. Lean UX. Створення класних продуктів із командами Agile. Київ: ArtHuss, 2024. 206 с.

16. JavaScript Рядкові методи [Електронний ресурс] – Режим доступу до ресурсу: https://w3schoolsua.github.io/js/js_string_methods.html#gsc.tab=0 (дата звернення: 16.05.2025).

17. UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 16.05.2025).

18. Застосування UML (частина 3). Діаграма класів - Class Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://duikt.edu.ua/ua/news-1-626-8002-zastosuvannya-uml-chastina-3-diagrama-klasiv----class-diagram_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy (дата звернення: 16.05.2025).

19. Що таке алгоритми: кроки, приклади, конструкції, мислення [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/shho-take-algorytmy-kroky-pryklady-konstrukcziyi-myslennya/ (дата звернення: 17.05.2025).

20. База даних PostgreSQL: інформація та короткий гайд [Електронний ресурс] – Режим доступу до ресурсу: https://itproger.com/ua/news/baza-dannih-postgresql-informatsiya-i-kratkiy-gayd (дата звернення: 17.05.2025).

21. Практичне застосування Zustand, від теорії до практики [Електронний ресурс] – Режим доступу до ресурсу: https://hackyourmom.com/osvita/praktychne-zastosuvannya-zustand-vid-teoriyi-do-praktyky/ (дата звернення: 18.05.2025).

22. Як використовувати Sass CSS [Електронний ресурс] – Режим доступу до ресурсу: https://javascript.org.ua/yak-vikoristovuvati-sass-css-posibnik-dlya-pochatkivcziv-po-stilizaczi%D1%97-vashogo-sajtu-yak-profesional/ (дата звернення: 18.05.2025).

23. Як працювати з Chart.js: Посібник для початківців [Електронний ресурс] – Режим доступу до ресурсу: https://markup-ua.com/yak-pracyuvati-z-chart-js-posibnik-dlya-pochatkivciv/ (дата звернення: 18.05.2025).

24. Frontend, Backend чи Full Stack: в чому відмінності, та що краще обрати для старту кар‘єри в IT-сфері [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/frontend-backend-ili-full-stack-v-chem-razlichija-i-chto-luchshe-vybrat-dlja-starta-karery-v-it-sfere/ (дата звернення: 19.05.2025).

25. Основні переваги JavaScript для веб-розробників [Електронний ресурс] – Режим доступу до ресурсу: https://codelabsacademy.com/uk/blog/top-javascript-advantages-for-web-developers/ (дата звернення: 20.05.2025).

26. Де використовується Python і чому вам потрібно знати цю мову [Електронний ресурс] – Режим доступу до ресурсу: https://genius.space/lab/de-

vikoristovuyetsya-python-i-chomu-vam-potribno-znati-tsyu-movu/ (дата звернення: 21.05.2025).

27. Чому Ruby – це гарний вибір для розробника у 2022/2023 роках [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/40283/ (дата звернення: 22.05.2025).

28. Про середовище програмування C# [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/seredovyshche-prohramuvannia-si-sharp/ (дата звернення: 23.05.2025).

29. Який редактор коду краще: Visual Studio Code чи Sublime Text? [Електронний ресурс] – Режим доступу до ресурсу: https://highload.tech/uk/yakyj-redaktor-kodu-krashhe-visual-studio-code-chy-sublime-text/ (дата звернення: 23.05.2025).

30. 5 чеснот Atom, які викликають звикання [Електронний ресурс] – Режим доступу до ресурсу: https://ni4yja.medium.com/five-reasons-to-love-atom-4732e2f052b9 (дата звернення: 24.05.2025).

31. ТОП-7 популярних IDE для програмування [Електронний ресурс] – Режим доступу до ресурсу: https://itvdn.com/ua/blog/article/cplspls-top7 (дата звернення: 25.05.2025).

32. Легше, швидше, масштабніше. Чому топові компанії використовують Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://prjctr.com/mag/whynode (дата звернення: 25.05.2025).

33. JSON Web Token (JWT) [Електронний ресурс] – Режим доступу до ресурсу: https://www.vpnunlimited.com/ua/help/cybersecurity/json-web-token?srsltid=AfmBOoqOJtjp4rx6GUtwq8iKzx3gG2vhjde1ma95ElHAX6cX6Tj9Pqua (дата звернення: 25.05.2025).

34. Як створити модальне вікно на React? [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/modalne-vikno-react/ (дата звернення: 26.05.2025).

35. React хук useMemo [Електронний ресурс] – Режим доступу до ресурсу: https://w3schoolsua.github.io/react/react_usememo.html#gsc.tab=0 (дата звернення: 26.05.2025).

36. Що таке тестування програмного забезпечення? [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/shho-take-testuvannya-programnogo-zabezpechennya/ (дата звернення: 27.05.2025).

37. Візуалізація тестування та покриття [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/visualization-of-testing-and-test-coverage/ (дата звернення: 27.05.2025).

38 Інструкція користувача [Електронний ресурс] – Режим доступу до ресурсу: https://smih.oda.v.ua/vzaiemodiya/shho-take-instrukciya-koristuvacha.html (дата звернення: 29.05.2025).

39. Як скоротити час відгуку сервера і вийти в топ пошукової видачі? [Електронний ресурс] – Режим доступу до ресурсу: https://hostpro.ua/blog/ua/how-to-reduce-server-response-time/ (дата звернення: 30.05.2025).

40. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

ДОДАТКИ

101

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«21» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка методу та засобів реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Войтко В. В.
«21» березня 2025 р.

Виконав:
студент гр. ЗПІ-21б Юдін О.С.
«21» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка методу та засобів реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень».

Галузь застосування – сфера цифрових медіа та онлайн-комунікацій: соціальні платформи для обміну візуальним контентом

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки

Метою роботи є підвищення рівня якості аналізу й оцінювання статичних зображень шляхом розробки методу та засобів реалізації вебсистеми, що дозволить автоматизувати обробку графічного контенту, забезпечити гнучку систему фільтрації та оцінювання візуальних матеріалів.

Призначення роботи – розробка методу та засобів реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень.

4. Вихідні дані для проведення НДР

1. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

2. Основні переваги JavaScript для веб-розробників [Електронний ресурс] – Режим доступу до ресурсу: https://codelabsacademy.com/uk/blog/top-javascript-advantages-for-web-developers/ (дата звернення: 02.05.2025).

3. Джеф Ґотельф. Lean UX. Створення класних продуктів із командами Agile. Київ: ArtHuss, 2024. 206 с.

5. Технічні вимоги

Комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц.

Рекомендований об’єм оперативної пам’яті 6 ГБ RAM, вільний простір на диску 2 ГБ. Операційна система Windows 10 / 11, Linux, macOS. З клієнтського боку: мобільний пристрій з ОС Android або iOS; ПК з будь-якою операційною системою та будь-який браузер.

6. Конструктивні вимоги

Вебзастосунок та веб-клієнт повинні відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

- пояснювальна записка до бакалаврської кваліфікаційної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку програм для візуалізації, аналізу й оцінювання статичних зображень та постановка задач розробки	25.03.25 – 29.03.25
2	Розробка методів, моделі та алгоритмів роботи програми	30.03.25 – 10.04.25
3	Розробка програмного забезпечення	11.04.25 – 10.05.25
4	Тестування програми	11.05.25 – 14.05.25
5	Оформлення матеріалів до захисту БКР	15.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

105

Додаток Б. Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка методу та засобів реалізації вебсистеми візуалізації, аналізу й оцінювання статичних зображень.

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Войтко В. В.

Показники звіту подібності

Оригінальність	98 %
Загальна схожість	2 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism.

Автор роботи _____ Юдін О. С.

Керівник роботи _____ Войко В. В.

Додаток В. Лістинг програми

mark.entity.ts

```
import {
  Entity,
  Column,
  ManyToMany,
  OneToOne,
  BeforeInsert,
  BeforeUpdate,
  AfterUpdate,
} from 'typeorm';

import { AbstractEntity } from './abstract-entity';
import { ArticleEntity } from 'src/entities/article.entity';
import { UpdateArticleDTO } from 'src/models/article.models';
import { UpdateMarkDto } from 'src/marks/dto/update-mark.dto';

@Entity('marks')
export class MarkEntity extends AbstractEntity {
  @OneToOne(() => ArticleEntity, (article) => article.mark)
  article: ArticleEntity;

  @Column({ default: 0 })
  imageRating: number;

  @Column({ default: 0 })
  titleRating: number;

  @Column({ default: 0 })
  contentRating: number;

  @Column({ default: 0 })
  tagsRating: number;

  @Column({ default: 0, type: 'float' })
  totalRating: number;
```

```
  calculateMarkFields(marks: UpdateMarkDto) {
    this.imageRating += marks.imageRating;
    this.titleRating += marks.titleRating;
    this.contentRating += marks.contentRating;
    this.tagsRating += marks.tagsRating;
    this.totalRating =
      1 * this.imageRating +
      0.5 * this.titleRating +
      0.5 * this.contentRating +
      0.3 * this.tagsRating;
  }
}
```

MarathonVotePostModal.tsx

```
type Props = {
  onClose: () => void;
  post: ArticleType | null;
  markId: number | null;
};

const INITIAL_VALUES: UpdateEvaluationArticleType = {
  titleRating: 0,
  contentRating: 0,
  tagsRating: 0,
  imageRating: 0,
};

export const MarathonVotePostModal: FC<Props> = ({
  onClose,
  post,
  markId = 0,
}) => {
  const [activeStep, setActiveStep] = useState(0);
  const [marks, setMarks] =
    useState<UpdateEvaluationArticleType>(INITIAL_VALUES);
  const postEvaluationVote = useEvaluateStore(
```

```
  (state) => state.postEvaluationVote
);

const handleSubmitStep = useCallback(
  (fieldName: string, value: number) => {
    setActiveStep(activeStep + 1);
    setMarks((prev) => ({ ...prev, [fieldName]: value }));
  },
  [activeStep]
);

const handleSubmitForm = useCallback(() => {
  onClose();
  postEvaluationVote(markId, marks);
}, [markId, marks, onClose, postEvaluationVote]);

const Content = useMemo(() => {
  switch (activeStep) {
    case 0:
      return (
        <GeneralPostInformationStep post={post} onSubmit={handleSubmitStep} />
      );

    case 1:
      return (
        <TitlePostInformationStep post={post} onSubmit={handleSubmitStep} />
      );
    case 2:
      return (
        <TextPostInformationStep post={post} onSubmit={handleSubmitStep} />
      );
    case 3:
      return (
        <TagsPostInformationStep post={post} onSubmit={handleSubmitStep} />
      );
    case 4:
      return (
        <ImagePostInformationStep
```

```
          post={post}
          onSubmit={handleSubmitStep}
          isDisabled={!!marks.image}
        />
      );
    default:
      return <Results values={marks} onSubmit={handleSubmitForm} />;
  }
 }, [activeStep, handleSubmitForm, handleSubmitStep, marks, post]);

 return (
  <Modal
   open={true}
   className={styles.modal}
   onClose={onClose}
   disableAutoFocus
  >
   <div className={styles.wrapper}>
    <CustomStepper activeStep={activeStep} />
    {Content}
   </div>
  </Modal>
 );
};
```

GeneralPostInformationStep.tsx

```
return (
  <div className={styles.wrapper}>
   <InstructionItem
    description="На даному етапі Ви повинні ознайомитися із загальною інформацією про
    роботу. Це допоможе Вам сформувати своє враження про роботу і бути в
    контексті, під час проходження наступних кроків."
   />
   <TableContainer component={Paper}>
    <Table aria-label="simple table" className={styles.table}>
     <TableBody>
      {rows.map((row) => (
```

```
            <TableRow
              key={row.name}
              sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
            >
              <TableCell component="th" scope="row">
                {row.name}
              </TableCell>
              <TableCell align="right">{row.value}</TableCell>
            </TableRow>
          ))}
          <TableRow
            sx={{ '&:last-child td, &:last-child th': { border: 0 } }}
          >
            <TableCell component="th" scope="row">
              Теги
            </TableCell>
            <TableCell align="right">
              <TagList
                list={getPostTags(post?.tags)}
                className={styles.tags}
              />
            </TableCell>
          </TableRow>
        </TableBody>
      </Table>
    </TableContainer>
    <Button variant="contained" color="warning" onClick={onSubmit}>
      Я ознайомився(-лася) із інформацією
    </Button>
  </div>
 );
};
```

EvaluateStore.ts

```
import { create } from 'zustand';

import axios from '@core/configs/axios.interceptor';
```

```
import { EvaluateStoreType } from '@stores/EvaluateStore/types';
import {
  EvaluateArticleItemType,
  UpdateEvaluationArticleType,
} from '@type/evaluate';

export const useEvaluateStore = create<EvaluateStoreType>((set, get) => ({
  participants: [] as EvaluateArticleItemType[],
  fetchParticipants: async () => {
    try {
      const { data } = await axios.get<EvaluateArticleItemType[]>('/marks');

      set({ participants: data });
    } catch (error: any) {
      alert(error?.data?.message);
    }
  },
  postEvaluationVote: async (id: number, body: UpdateEvaluationArticleType) => {
    try {
      const { data } = await axios.post<EvaluateArticleItemType[]>(
        `/marks/${id}`,
        body
      );

      set({ participants: data });
    } catch (error: any) {
      alert(error?.data?.message);
    }
  },
  postArticleForEvaluation: async (articleId: number) => {
    try {
      await axios.post('/marks', { articleId });
    } catch (error: any) {
      alert(error?.data?.message);
    }
  },
  removeArticleFromEvaluation: async (markId) => {
    try {
```

```
      await axios.delete(`/marks/${markId}`);
    } catch (error: any) {
      alert(error?.data?.message);
    }
  },
}));
```

article.service.ts

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  Repository,
  Like,
  ILike,
  In,
  FindOptionsWhere,
  Brackets,
} from 'typeorm';

import { ArticleEntity } from 'src/entities/article.entity';
import { UserEntity } from 'src/entities/user.entity';
import { TagEntity } from 'src/entities/tag.entity';
import {
  CreateArticleDTO,
  UpdateArticleDTO,
  FindAllQuery,
  FindFeedQuery,
  ArticleResponse,
  FindArticlesQuery,
} from 'src/models/article.models';

@Injectable()
export class ArticleService {
  constructor(
    @InjectRepository(ArticleEntity)
    private articleRepo: Repository<ArticleEntity>,
    @InjectRepository(UserEntity) private userRepo: Repository<UserEntity>,
```

```
  @InjectRepository(TagEntity) private tagRepo: Repository<TagEntity>,
) {}

private async upsertTags(tagList: string[]): Promise<TagEntity[]> {
  // Отримуємо вже існуючі теги
  const foundTags = await this.tagRepo.find({
    where: tagList.map((tag) => ({ tag })),
  });

  // Визначаємо, яких тегів ще немає в БД
  const existingTagNames = new Set(foundTags.map((tag) => tag.tag));
  const newTags = tagList
    .filter((tag) => !existingTagNames.has(tag))
    .map((tag) => this.tagRepo.create({ tag }));

  // Зберігаємо нові теги
  const savedNewTags = await this.tagRepo.save(newTags);

  // Повертаємо всі теги (і старі, і нові)
  return [...foundTags, ...savedNewTags];
}

async findAll(
  user: UserEntity,
  query: FindAllQuery,
): Promise<ArticleResponse[]> {
  let findOptions: any = {
    where: {},
  };

  if (query.author) {
    findOptions.where['author.username'] = query.author;
  }
  if (query.favorited) {
    findOptions.where['favoritedBy.username'] = query.favorited;
  }
  if (query.tag) {
    findOptions.where.tagList = Like(`%${query.tag}%`);
```

```
  }
  if (query.offset) {
    findOptions.offset = query.offset;
  }
  if (query.limit) {
    findOptions.limit = query.limit;
  }
  return (await this.articleRepo.find(findOptions)).map((article) =>
    article.toArticle(user),
  );
}

async findArticlesByUserId(authorId: string) {
  return await this.articleRepo.find({
    where: {
      author: {
        id: +authorId,
      },
    },
    relations: ['author', 'comments', 'tags', 'mark'],
  });
}

async viewArticle(id: string) {
  const article = await this.articleRepo.findOneBy({ id: +id });

  article.views += 1;

  await article.save();
}

async searchArticles(
  user: UserEntity,
  query: FindArticlesQuery,
): Promise<ArticleResponse[]> {
  const { search, tags, username, comments } = query || {};
  console.log(query, 'queryqueryquery');
  const tagList = tags || [];
```

```
let queryBuilder = this.articleRepo
  .createQueryBuilder('article')
  .leftJoinAndSelect('article.author', 'author')
  .leftJoinAndSelect('article.favoritedBy', 'favoritedBy')
  .leftJoinAndSelect('article.comments', 'comments')
  .leftJoinAndSelect('article.tags', 'tags');

// Фільтр по автору
if (username) {
  queryBuilder = queryBuilder.andWhere('author.username = :username', {
    username,
  });
}

// Фільтр по пошуку (title, description, body)
if (search) {
  queryBuilder = queryBuilder.andWhere(
    new Brackets((qb) => {
      qb.where('article.title ILIKE :search', { search: `%${search}%` })
        .orWhere('article.description ILIKE :search', {
          search: `%${search}%`,
        })
        .orWhere('article.body ILIKE :search', { search: `%${search}%` });
    }),
  );
}

// Фільтр по тегах
if (tagList.length > 0) {
  queryBuilder = queryBuilder.andWhere(
    new Brackets((qb) => {
      tagList.forEach((tag, index) => {
        qb.orWhere(`tags.tag ILIKE :tag${index}`, {
          [`tag${index}`]: `%${tag}%`,
        });
      });
    }),
```

```
    );
  }

  // Виконуємо запит
  let articles = await queryBuilder.getMany();

  // Фільтр по кількості коментарів (якщо передано)
  if (comments) {
    articles = articles.filter(
      (article) => article.comments.length >= +comments,
    );
  }

  return articles.map((article) => article.toArticle(user));
}

async findFeed(
  user: UserEntity,
  query: FindFeedQuery,
): Promise<ArticleResponse[]> {
  const { followee } = await this.userRepo.findOne({
    where: { id: user.id },
    relations: ['followee'],
  });
  const findOptions = {
    ...query,
    where: followee.map((follow) => ({ author: { id: follow.id } })),
  };
  return (await this.articleRepo.find(findOptions)).map((article) =>
    article.toArticle(user),
  );
}

findBySlug(slug: string): Promise<ArticleEntity> {
  return this.articleRepo.findOne({
    where: { slug },
  });
}
```

```
private ensureOwnership(user: UserEntity, article: ArticleEntity): boolean {
  return article.author.id === user.id;
}

async createArticle(
  user: UserEntity,
  data: CreateArticleDTO,
): Promise<ArticleResponse> {
  const article = this.articleRepo.create(data);
  article.author = user;
  const tags = await this.upsertTags(data.tagList);
  article.tags = tags;
  const { slug } = await article.save();

  return article.toArticle(user);
}

async getFavouriteArticlesByUsername(username: string) {
  const articles = await this.articleRepo
    .createQueryBuilder('article')
    .innerJoin('article.favoritedBy', 'user', 'user.username = :username', {
      username,
    })
    .leftJoinAndSelect('article.author', 'author')
    .leftJoinAndSelect('article.favoritedBy', 'favoritedBy')
    .getMany();

  return articles;
}

async updateArticle(
  slug: string,
  user: UserEntity,
  data: UpdateArticleDTO,
): Promise<ArticleResponse> {
  const article = await this.findBySlug(slug);
  if (!this.ensureOwnership(user, article)) {
```

```
    throw new UnauthorizedException();
  }

  // Розбираємо вхідні дані (відокремлюємо список тегів)
  const { tagList, ...spreadedData } = data;

  // Отримуємо або створюємо необхідні теги
  const tags = tagList ? await this.upsertTags(tagList) : [];

  // Оновлюємо інші дані статті
  await this.articleRepo.update({ slug }, spreadedData);

  // Оновлюємо зв’язки з тегами
  await this.articleRepo
    .createQueryBuilder()
    .relation(ArticleEntity, 'tags')
    .of(article)
    .addAndRemove(tags, article.tags);

  // Отримуємо оновлену статтю
  const updatedArticle = await this.findBySlug(slug);
  return updatedArticle.toArticle(user);
}

async deleteArticle(
  slug: string,
  user: UserEntity,
): Promise<ArticleResponse> {
  console.log(slug, 'slugslugslugslug');
  const article = await this.findBySlug(slug);
  console.log(article, user, 'article, user');
  if (!this.ensureOwnership(user, article)) {
    throw new UnauthorizedException();
  }
  await this.articleRepo.remove(article);
  return article.toArticle(user);
}
```

```
async favoriteArticle(
  slug: string,
  user: UserEntity,
): Promise<ArticleResponse> {
  const article = await this.findBySlug(slug);
  article.favoritedBy.push(user);
  await article.save();
  console.log(article);
  return (await this.findBySlug(slug)).toArticle(user);
}

async unfavoriteArticle(
  slug: string,
  user: UserEntity,
): Promise<ArticleResponse> {
  const article = await this.findBySlug(slug);
  article.favoritedBy = article.favoritedBy.filter(
    (fav) => fav.id !== user.id,
  );
  await article.save();
  return (await this.findBySlug(slug)).toArticle(user);
}

async exchangeArticle(
  user: UserEntity,
  partnerArticleSlug: string,
  ownArticleSlug: string,
) {
  const partnerArticleBySlug = await this.findBySlug(partnerArticleSlug);
  const ownArticleBySlug = await this.findBySlug(ownArticleSlug);

  ownArticleBySlug.author = partnerArticleBySlug.author;
  partnerArticleBySlug.author = user;

  await partnerArticleBySlug.save();
  await ownArticleBySlug.save();

  return (await this.findBySlug(partnerArticleSlug)).toArticle(user);
```

```
  }
}
```

ExchangeStore.ts

```
import { create } from 'zustand';

import axios from '@core/configs/axios.interceptor';
import { ExchangeStoreType } from '@stores/ExchangeStore/types';
import {
  ArticleListResponseType,
  ExchangeArticleresponseType,
} from '@types/article';

export const useExchangeStore = create<ExchangeStoreType>((set, get) => ({
  initiatorPosts: [],
  partnerPosts: [],
  fetchInitiatorPosts: async (id) => {
    try {
      const { data } = await axios.get<ArticleListResponseType>(
        `/articles/feed/${id}`
      );

      set({ initiatorPosts: data.articles });
    } catch (error: any) {
      alert(error?.data?.message);
    }
  },
  fetchPartnerPosts: async (id) => {
    try {
      const { data } = await axios.get<ArticleListResponseType>(
        `/articles/feed/${id}`
      );

      set({ partnerPosts: data.articles });
    } catch (error: any) {
      alert(error?.data?.message);
    }
```

```
  },
  claimDeal: async (initiatorArticleSlug, partnerArticleSlug) => {
    try {
      await axios.post<ExchangeArticleresponseType>(
        `/articles/exchange?partnerArticleSlug=${partnerArticleSlug}&ownArticleSlug=${initiatorArticleSlug}`
      );
    } catch (error: any) {
      alert(error?.data?.message);
    }
  },
}));
```

ExchangePostModal.tsx

```
export const ExchangePostModal: FC<Props> = ({
  initiatorId,
  partnerId,
  onClose,
}) => {
  const { push } = useRouter();
  const [exchangePostSlug, setExchangePostSlug] = useState<
    Record<string | number, string | null>
  >({
    [initiatorId]: null,
    [partnerId]: null,
  });
  const fetchInitiatorPostList = useExchangeStore(
    (state) => state.fetchInitiatorPosts
  );
  const fetchPartnerPostList = useExchangeStore(
    (state) => state.fetchPartnerPosts
  );
  const claimDeal = useExchangeStore((state) => state.claimDeal);
  const initiatorPosts = useExchangeStore((state) => state.initiatorPosts);
  const partnerPosts = useExchangeStore((state) => state.partnerPosts);

  useEffect(() => {
    fetchInitiatorPostList(initiatorId);
```

```
  fetchPartnerPostList(partnerId);
}, [fetchInitiatorPostList, fetchPartnerPostList, initiatorId, partnerId]);

const handleClickInitiatorPost = useCallback(
  (slug: string) => {
    if (!!exchangePostSlug[initiatorId]) return;

    setExchangePostSlug((prev) => ({ ...prev, [initiatorId]: slug }));
  },
  [exchangePostSlug, initiatorId]
);
const handleClickPartnerPost = useCallback(
  (slug: string) => {
    if (!!exchangePostSlug[partnerId]) return;

    setExchangePostSlug((prev) => ({ ...prev, [partnerId]: slug }));
  },
  [exchangePostSlug, partnerId]
);

const isClearDisabled = Object.values(exchangePostSlug).every(
  (item) => !item
);
const isSubmitDisabled = Object.values(exchangePostSlug).some(
  (item) => !item
);

const handleClickClear = () => {
  setExchangePostSlug({
    [initiatorId]: null,
    [partnerId]: null,
  });
};
const handleSubmitExchange = () => {
  if (!exchangePostSlug[initiatorId] || !exchangePostSlug[partnerId])
    return null;

  claimDeal(exchangePostSlug[initiatorId], exchangePostSlug[partnerId]);
```

```
  onClose();
  push('/me');
 };

 return (
  <Modal
   open={true}
   className={styles.modal}
   onClose={onClose}
   disableAutoFocus
  >
   <div className={styles.wrapper}>
    <UserExchangePostList
     posts={initiatorPosts}
     onClick={handleClickInitiatorPost}
     chosenPostSlug={exchangePostSlug[initiatorId]}
    />
    <SwapCallsIcon sx={{ fill: 'black' }} className={styles.icon} />
    <UserExchangePostList
     posts={partnerPosts}
     onClick={handleClickPartnerPost}
     chosenPostSlug={exchangePostSlug[partnerId]}
    />
    <div className={styles.buttons}>
     <Button
      variant="contained"
      disabled={isSubmitDisabled}
      onClick={handleSubmitExchange}
     >
      Підтвердити
     </Button>
     <Button disabled={isClearDisabled} onClick={handleClickClear}>
      Очистити вибір
     </Button>
     <Button onClick={onClose}>Вийти</Button>
    </div>
   </div>
  </Modal>
```

```
  );
};
```

UserExchangePostList.tsx

```
type Props = {
  posts: ArticleType[];
  onClick: (postSlug: string) => void;
  chosenPostSlug: string | null;
};

export const UserExchangePostList: FC<Props> = ({
  posts,
  onClick,
  chosenPostSlug,
}) => {
  const author = posts?.[0]?.author;

  if (!posts.length) return null;

  return (
    <div className={styles.wrapper}>
      <UserAvatar avatarUrl={author.image} />
      <div className={styles.list}>
        {posts.map((post) => (
          <div
            key={post.id}
            className={classNames(styles.imageWrapper, {
              [styles.disabled]: chosenPostSlug && chosenPostSlug !== post.slug,
            })}
            onClick={() => onClick(post.slug)}
          >
            {chosenPostSlug && chosenPostSlug === post.slug && (
              <CheckCircleIcon
                className={styles.checkIcon}
                sx={{ width: '50px', height: '50px' }}
              />
            )}
```

```
          <Image src={post.image} alt="post" fill />
        </div>
      ))}
    </div>
  </div>
 );
};
```

Analytics.tsx

```
import { FC, useMemo } from 'react';

import { BarChart } from '@mui/x-charts';

import { ArticleType } from '@type/article';

import { EmptyList } from '@components/EmptyList';

import styles from './Analytics.module.scss';

type Props = {
 posts: ArticleType[] | null;
};

export const Analytics: FC<Props> = ({ posts }) => {
 const chartsData = useMemo(() => {
  if (!posts?.length) {
   return null;
  }

  const postsStatistic = [
   {
    data: posts.map(({ commentsCount }) => commentsCount),
    label: 'Коментарі',
   },
   {
    data: posts.map(({ mark }) => mark?.totalRating || 0),
    label: 'Рейтинг в оцінюванні (якщо робота подана)',
```

```
      },
      { data: posts.map(({ views }) => views), label: 'Перегляди' },
    ];

    const xAxises = [
      {
        scaleType: 'band',
        data: posts.map((postItem) => postItem.title),
      },
    ];

    return { postsStatistic, xAxises };
  }, [posts]);
  return (
    <EmptyList list={posts} message="Створіть пост для віображення аналітики.">
      <div className={styles.wrapper}>
        <p>Статистика моїх робіт</p>
        {chartsData && (
          <BarChart
            xAxis={chartsData.xAxises}
            series={chartsData.postsStatistic}
            width={700}
            height={500}
          />
        )}
      </div>
    </EmptyList>
  );
};
```

TagsPostInformationStep.tsx

```
type Props = {
  post: ArticleType | null;
  onSubmit: (fieldName: string, value: number) => void;
  isDisabled?: boolean;
};
export const TagsPostInformationStep: FC<Props> = ({
```

```
  post,
  onSubmit,
  isDisabled,
}) => {
  const [ratingValue, setRatingValue] = useState<number>(0);

  const handleSubmit = useCallback(() => {
    onSubmit('tagsRating', ratingValue);
  }, [onSubmit, ratingValue]);

  return (
    <div className={styles.wrapper}>
      <InstructionItem
        description="На даному етапі Ви повинні ознайомитися із тегами роботи.
      Оцінити відповідність із заголовком і вмістом."
      />
      <InstructionItem description="Увага! Можливість повернутися буде відсутньою. Будьте уважні, не спішіть." />
      <TagList list={getPostTags(post?.tags)} />
      <Stack spacing={1} className={styles.stack}>
        <Rating
          name="simple-controlled"
          value={ratingValue}
          onChange={(_event, newValue) => {
            console.log(newValue, 'sdfdsfdsfdfdsdsf');
            setRatingValue(newValue ?? 1);
          }}
          className={styles.rating}
          size="large"
          max={10}
        />
      </Stack>
      <Button
        variant="contained"
        color="warning"
        onClick={handleSubmit}
        disabled={isDisabled}
      >
```

```
      Я готовий(-ова) рухатися далі
    </Button>
   </div>
  );
};
```

TitlePostInformationStep.tsx

```
type Props = {
  post: ArticleType | null;
  onSubmit: (fieldName: string, value: number) => void;
  isDisabled?: boolean;
};

export const TitlePostInformationStep: FC<Props> = ({
  post,
  onSubmit,
  isDisabled,
}) => {
  const [ratingValue, setRatingValue] = useState<number | null>(0);

  const handleChangeRating = useCallback(
    (_event: any, newValue: number | null) => {
      setRatingValue(newValue ?? 1);
    },
    []
  );

  const handleSubmit = useCallback(() => {
    onSubmit('titleRating', ratingValue ?? 0);
  }, [onSubmit, ratingValue]);

  return (
    <div className={styles.wrapper}>
      <InstructionItem
        description="На даному етапі Ви повинні ознайомитися із заголовком роботи.
      Оцінити його доречність."
      />
```

```
    <InstructionItem description="Увага! Можливість повернутися буде відсутньою. Будьте уважні, не спішіть." />
    <InstructionItem description={`Заголовок: "${post?.title}"`} />
    <Stack spacing={1} className={styles.stack}>
     <Rating
      name="simple-controlled"
      value={ratingValue}
      onChange={handleChangeRating}
      className={styles.rating}
      size="large"
      max={10}
     />
    </Stack>
    <Button
     variant="contained"
     color="warning"
     onClick={handleSubmit}
     disabled={isDisabled}
    >
     Я готовий(-ова) рухатися далі
    </Button>
   </div>
  );
};
```

MarathonItem.tsx

```
type Props = {
 mark: EvaluateArticleItemType;
 onClickVote: () => void;
};

export const MarathonItem: FC<Props> = ({ mark, onClickVote }) => {
 const { article } = mark;

 if (!article) return null;

 return (
```

```
<div className={styles.wrapper}>
  <Image src={article.image} width={100} height={100} alt="" />
  <div className={styles.body}>
    <p>{article.title}</p>
    <TagList list={getPostTags(article.tags)} />
    <div className={styles.analytics}>
      <CustomLinearProgress
        icon={
          <Tooltip title="Заголовок">
            <TitleIcon />
          </Tooltip>
        }
        value={mark.titleRating}
      />
      <CustomLinearProgress
        icon={
          <Tooltip title="Контент">
            <FormatShapesIcon />
          </Tooltip>
        }
        value={mark.contentRating}
      />
      <CustomLinearProgress
        icon={
          <Tooltip title="Теги">
            <TagIcon />
          </Tooltip>
        }
        value={mark.tagsRating}
      />
      <CustomLinearProgress
        icon={
          <Tooltip title="Зображення">
            <ImageSearchIcon />
          </Tooltip>
        }
        value={mark.imageRating}
      />
```

```
      </div>
    </div>
    <Button
      startIcon={<ThumbsUpDownIcon />}
      className={styles.voteButton}
      variant="contained"
      onClick={onClickVote}
    >
      Оцінити
    </Button>
  </div>
 );
};
```

comments.service.ts

```
@Injectable()
export class CommentsService {
 constructor(
  @InjectRepository(CommentEntity)
  private commentRepo: Repository<CommentEntity>,
  @InjectRepository(ArticleEntity)
  private articleRepo: Repository<ArticleEntity>,
 ) {}

 findByArticleSlug(slug: string): Promise<CommentResponse[]> {
  return this.commentRepo.find({
   where: { article: { slug } },
   relations: ['article'],
  });
 }

 findById(id: number): Promise<CommentResponse> {
  return this.commentRepo.findOne({ where: { id } });
 }

 async createComment(
  user: UserEntity,
```

```
    data: CreateCommentDTO,
    articleSlug: string,
  ): Promise<CommentResponse> {
    const article = await this.articleRepo.findOneBy({ slug: articleSlug });
    const comment = this.commentRepo.create(data);
    comment.author = user;
    comment.article = article;
    return await comment.save();
  }

  async deleteComment(user: UserEntity, id: number): Promise<CommentResponse> {
    const comment = await this.commentRepo.findOne({
      where: { id, author: { id: user.id } },
    });
    console.log(comment, 'sddskdkdskjkdkjds');
    await comment.remove();
    return comment;
  }
}
```

user.service.ts

```
import { Injectable, InternalServerErrorException } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

import { UserEntity } from 'src/entities/user.entity';
import { ProfileResponse } from 'src/models/user.model';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(UserEntity) private userRepo: Repository<UserEntity>,
  ) {}
  async findByUsername(
    username: string,
    user?: UserEntity,
  ): Promise<ProfileResponse> {
```

```
    return (
      await this.userRepo.findOne({
        where: { username },
        relations: ['followers'],
      })
    )?.toProfile(user);
  }

  async followUser(
    currentUser: UserEntity,
    username: string,
  ): Promise<ProfileResponse> {
    const user = await this.userRepo.findOne({
      where: { username },
      relations: ['followers'],
    });
    if (!user) return {} as ProfileResponse;
    user.followers.push(currentUser);
    await user.save();
    return user.toProfile(currentUser);
  }

  async unfollowUser(
    currentUser: UserEntity,
    username: string,
  ): Promise<ProfileResponse> {
    const user = await this.userRepo.findOne({
      where: { username },
      relations: ['followers'],
    });
    if (!user) throw new InternalServerErrorException();
    user.followers = user.followers.filter(
      (follower) => follower !== currentUser,
    );
    await user.save();
    return user.toProfile(currentUser);
  }
}
```

Додаток Г. Ілюстративна частина

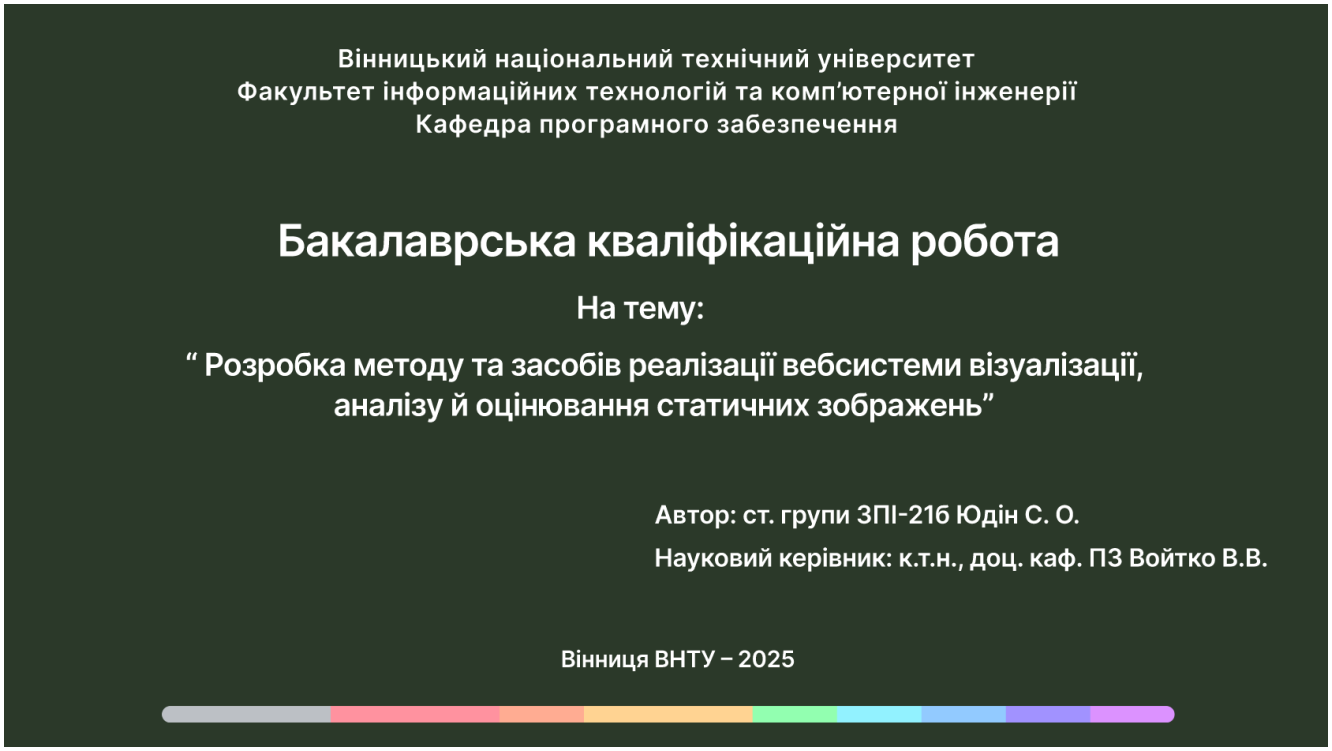

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність теми

Рисунок Г.3 – Мета та завдання, об’єкт, предмет дослідження

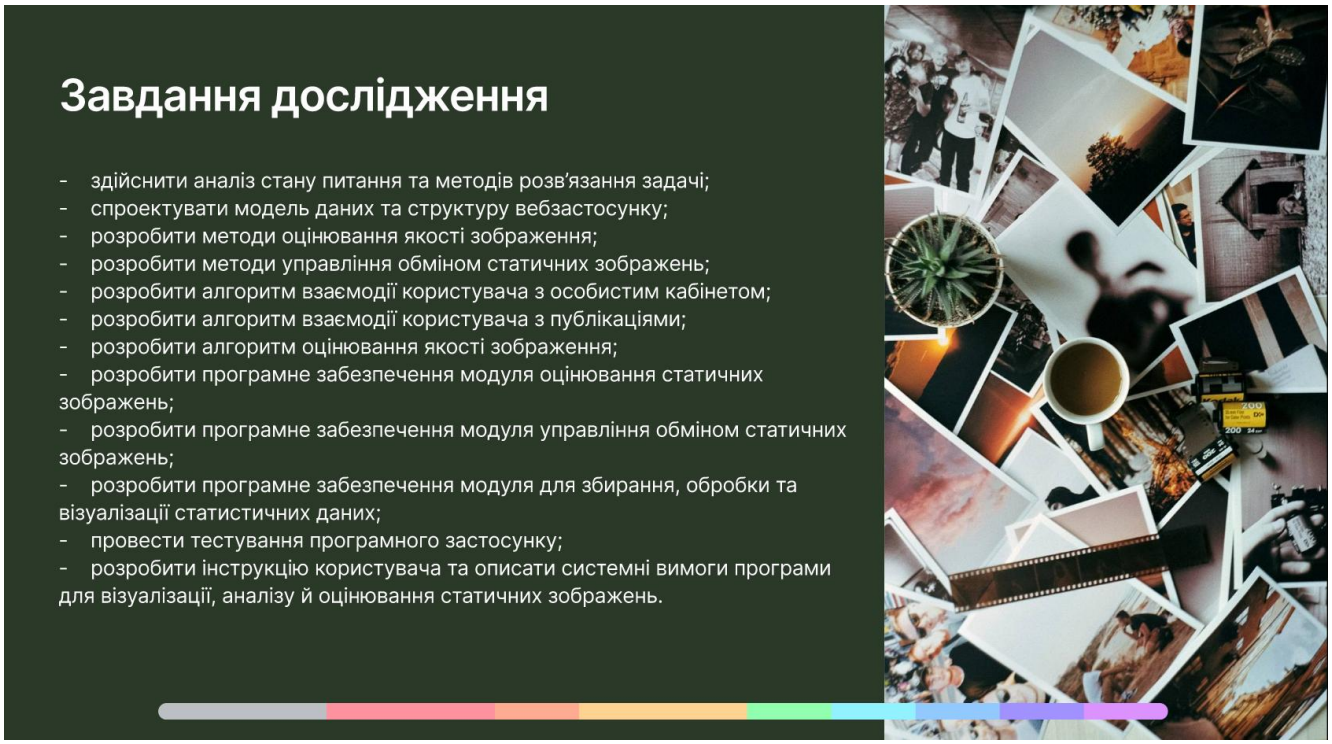

Рисунок Г.4 – Завдання дослідження

Рисунок Г.5 – Новизна отриманих результатів

Рисунок Г.6 – Практичне значення отриманих результатів

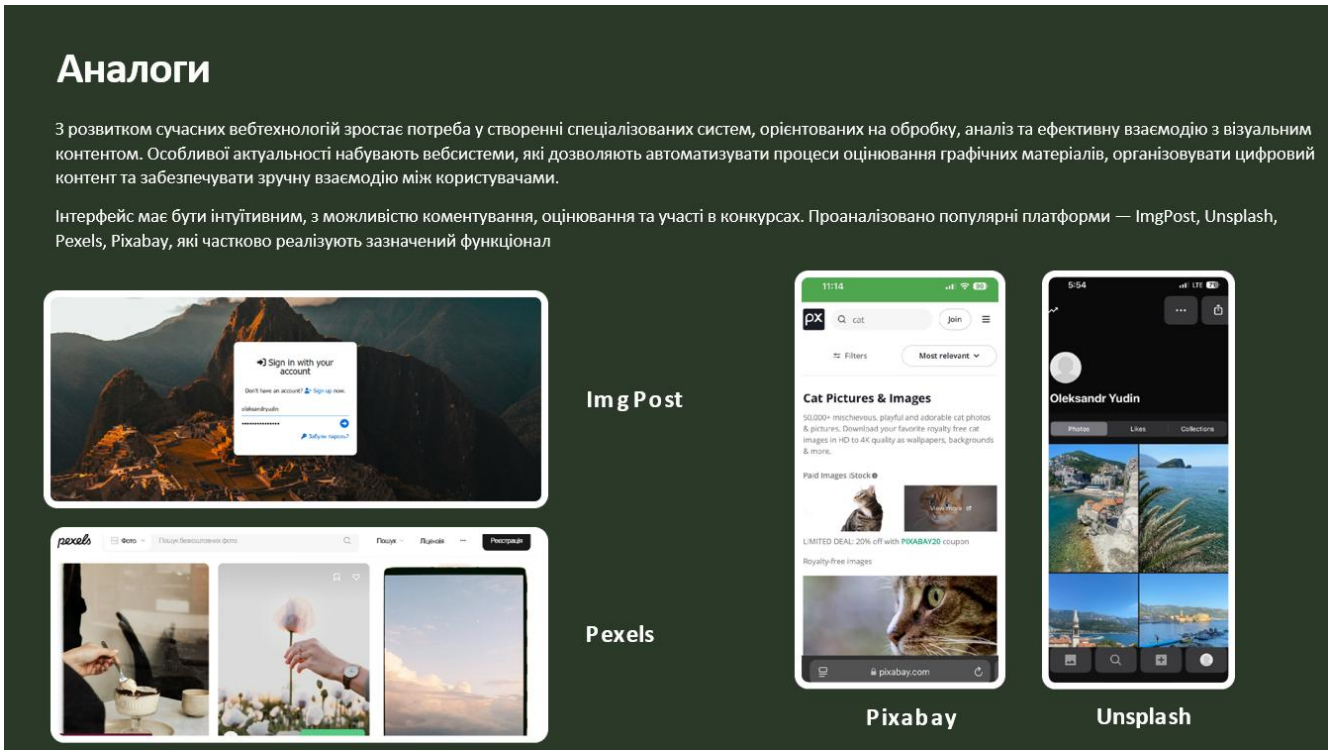

Рисунок Г.7 – Аналоги

Функціональні можливості	ImgPost	Unsplash	Pexels	Pixabay	Власна розробка
Додавання коментарів	0	0	0	0	1
Можливість додавати теги до постів	0	1	1	1	1
Система фільтрації контенту	0	1	1	1	1
Персональний профіль з усіма публікаціями	1	1	1	1	1
Конкурсне оцінювання зображень	0	0	1	0	1
Обмін постами між користувачами	0	0	0	0	1
Сумарний коефіцієнт	1	3	4	3	6

Рисунок Г.8 – Порівняльний аналіз аналогів

Рисунок Г.9 – Розробка методу оцінювання якості зображення

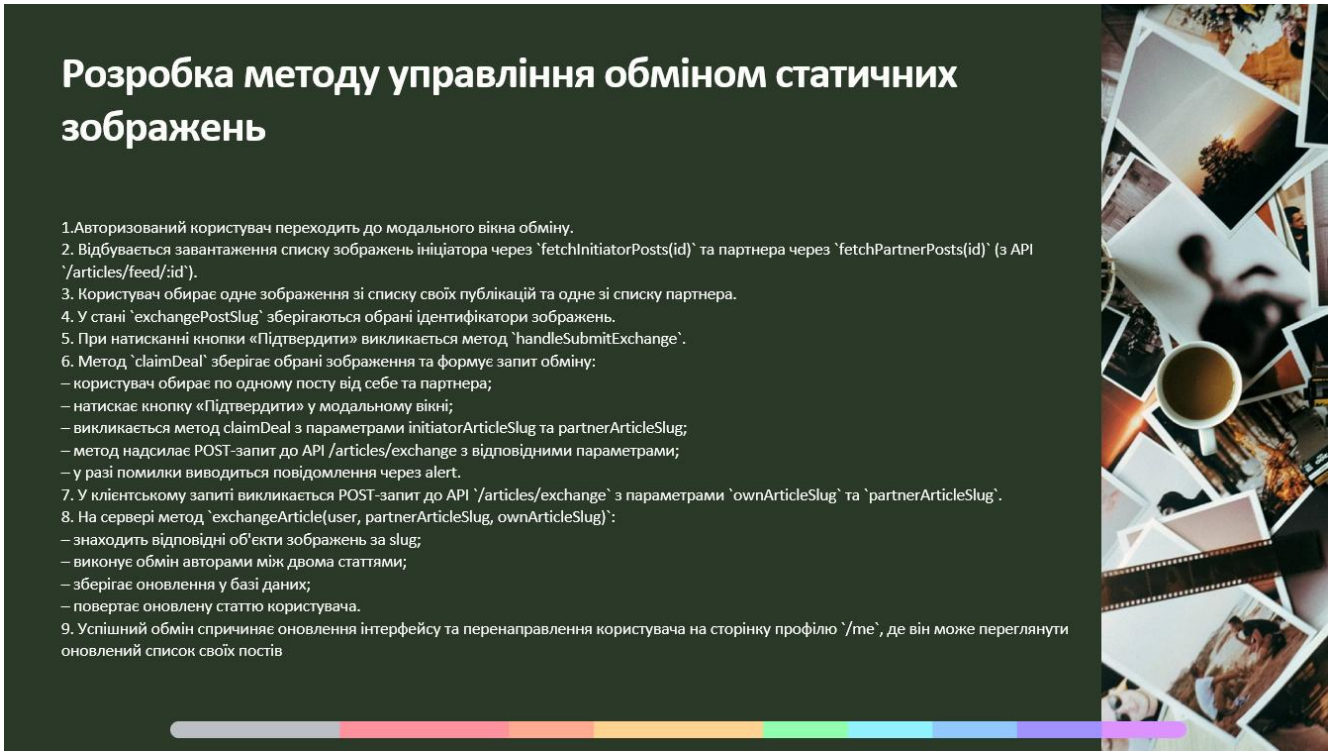

Рисунок Г.10 – Розробка методу управління обміном статичних зображень

Алгоритм взаємодії користувача з особистим кабінетом

Для взаємодії з особистим кабінетом користувач проходить автентифікацію: вводить логін і пароль, які перевіряються через базу даних PostgreSQL. У разі успішної перевірки система надає доступ до кабінету, реалізованого у середовищі React із використанням Zustang для управління сесією.
Особистий кабінет надає користувачу інтерфейс для перегляду, редагування, видалення або подання зображень на конкурси. Публікації містять інформацію про тип зображення, параметри обробки, а також доступні функції додавання тегів і налаштування прав доступу для завантаження іншими користувачами.
Важливим елементом є перегляд статистики щодо кожної публікації: кількість переглядів, лайків, коментарів або завантажень. Для зручного візуального аналізу даних використовуються інтерактивні графіки, побудовані з використанням бібліотеки Chart.js.
Користувач також може створювати нові публікації через форму, де вказуються опис, зображення, теги та параметри обробки. Після підтвердження всі дані автоматично зберігаються у системі й відображаються в кабінеті.
Завершення сесії здійснюється натисканням кнопки «Вийти». Після цього відбувається безпечний вихід із системи та перенаправлення на головну сторінку або форму входу, що гарантує захист персональних даних.

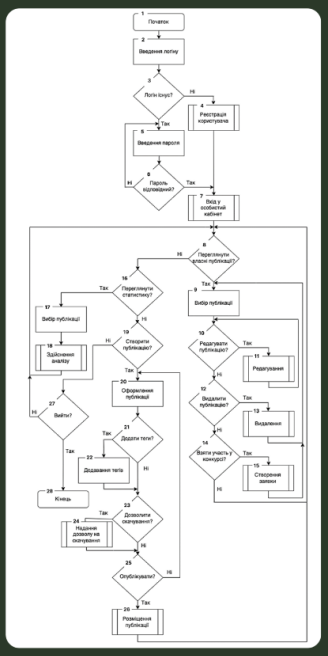

Рисунок Г.11 – Алгоритм взаємодії користувача з ососбистим кабінетом

Алгоритм взаємодії користувача з публікаціями

З метою забезпечення ефективної взаємодії з платформою для візуалізації та оцінювання зображень розроблено базовий алгоритм, який охоплює основні дії: завантаження контенту, його перегляд, взаємодію з публікаціями, залишення лайків, коментарів і можливість обміну зображеннями.
Після ініціації сесії відбувається завантаження стрічки публікацій через API-запити (Node.js + Axios). Користувач може налаштувати фільтри (Zustand) і переглядати публікації через адаптивний React-інтерфейс. Реалізовано функції вподобань (POST-запити), коментування (збереження в базі через зовнішній ключ) і обміну зображеннями із перевіркою прав доступу.
Завершення взаємодії передбачає вихід із сесії та повернення до головного інтерфейсу або авторизації, що гарантує безпеку даних користувача.

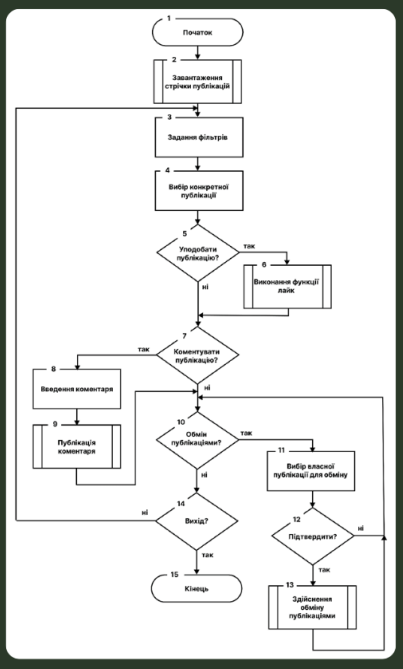

Рисунок Г.12 – Алгоритм взаємодії користувача з публікаціями

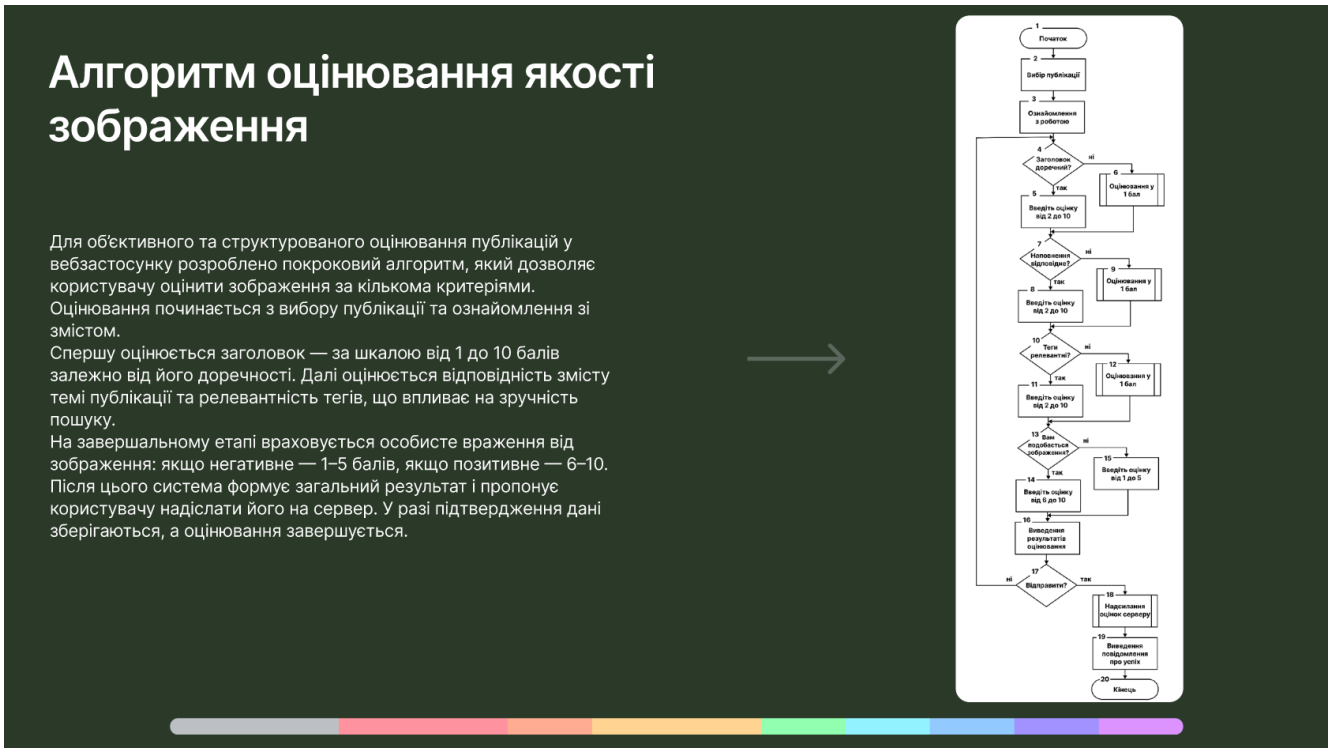

Рисунок Г.13 – Алгоритм оцінювання якості зображення

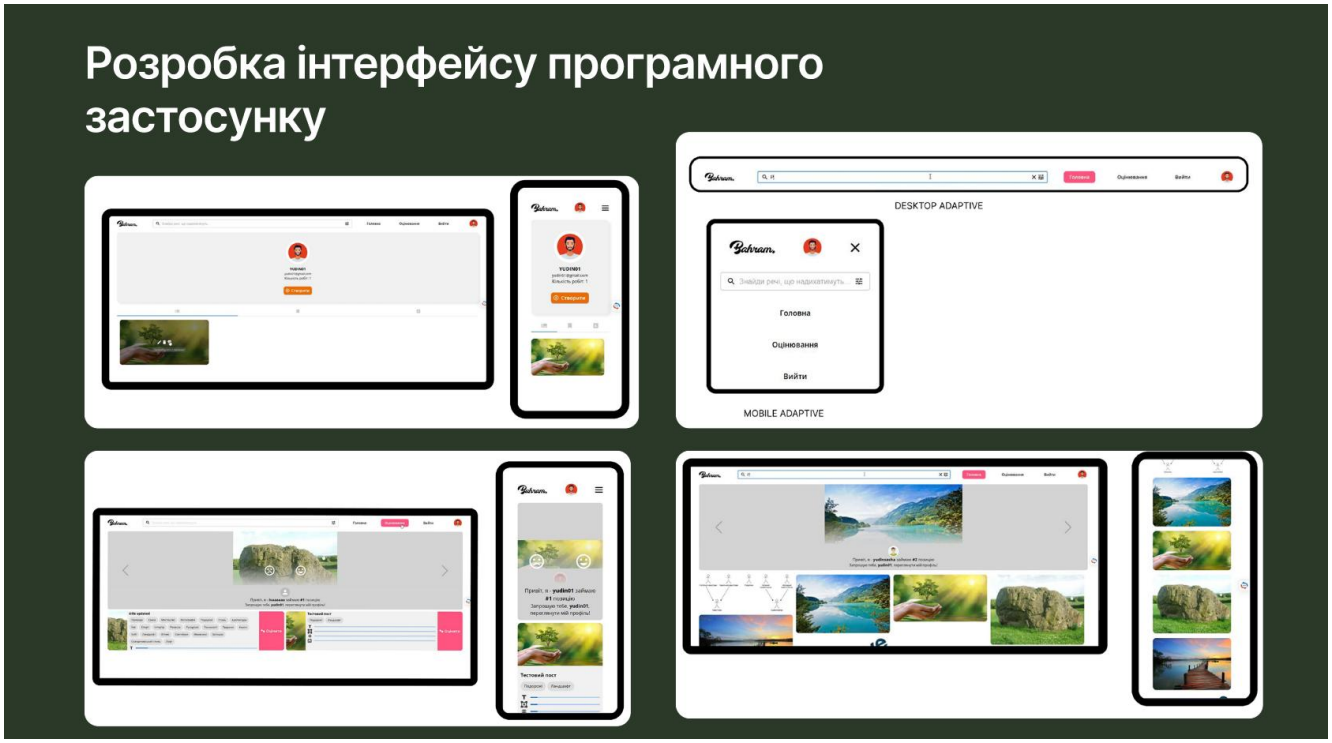

Рисунок Г.14 – Розробка інтерфейсу програмного застосунку

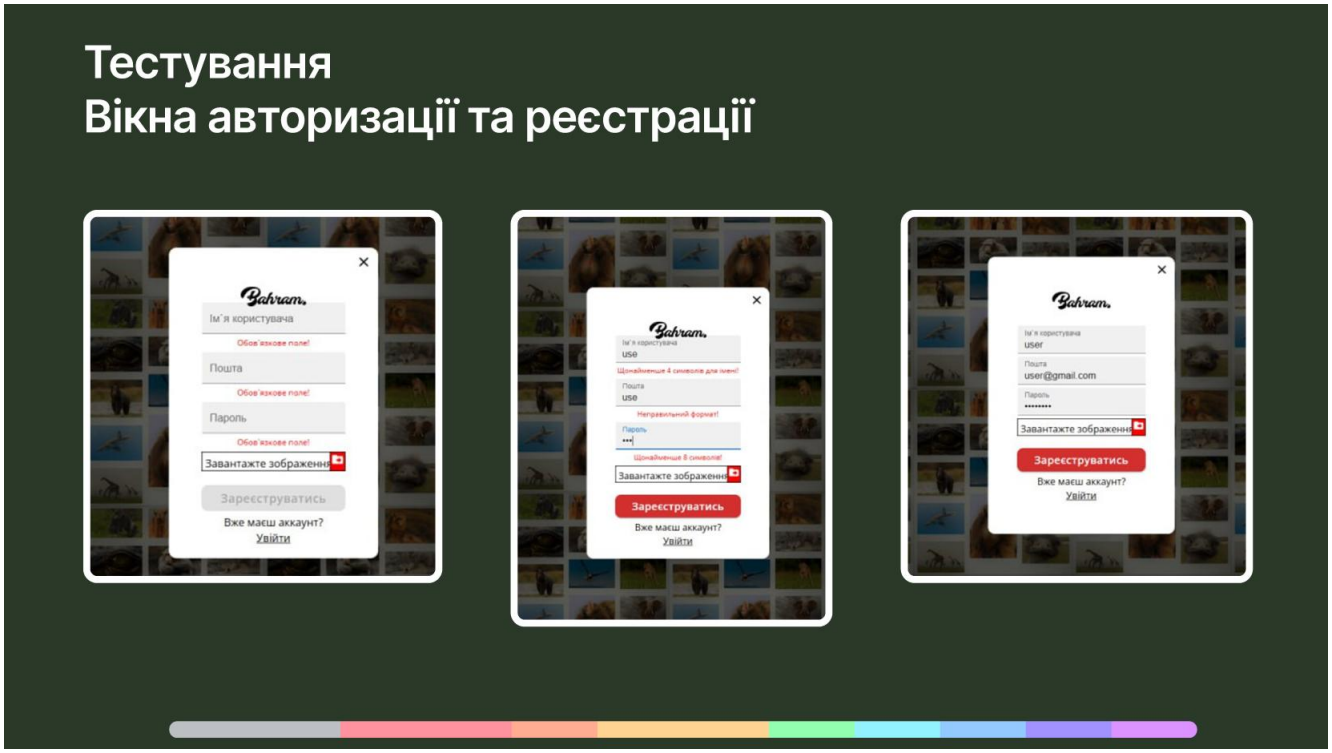

Рисунок Г.15 – Вікна авторизації та реєстрації

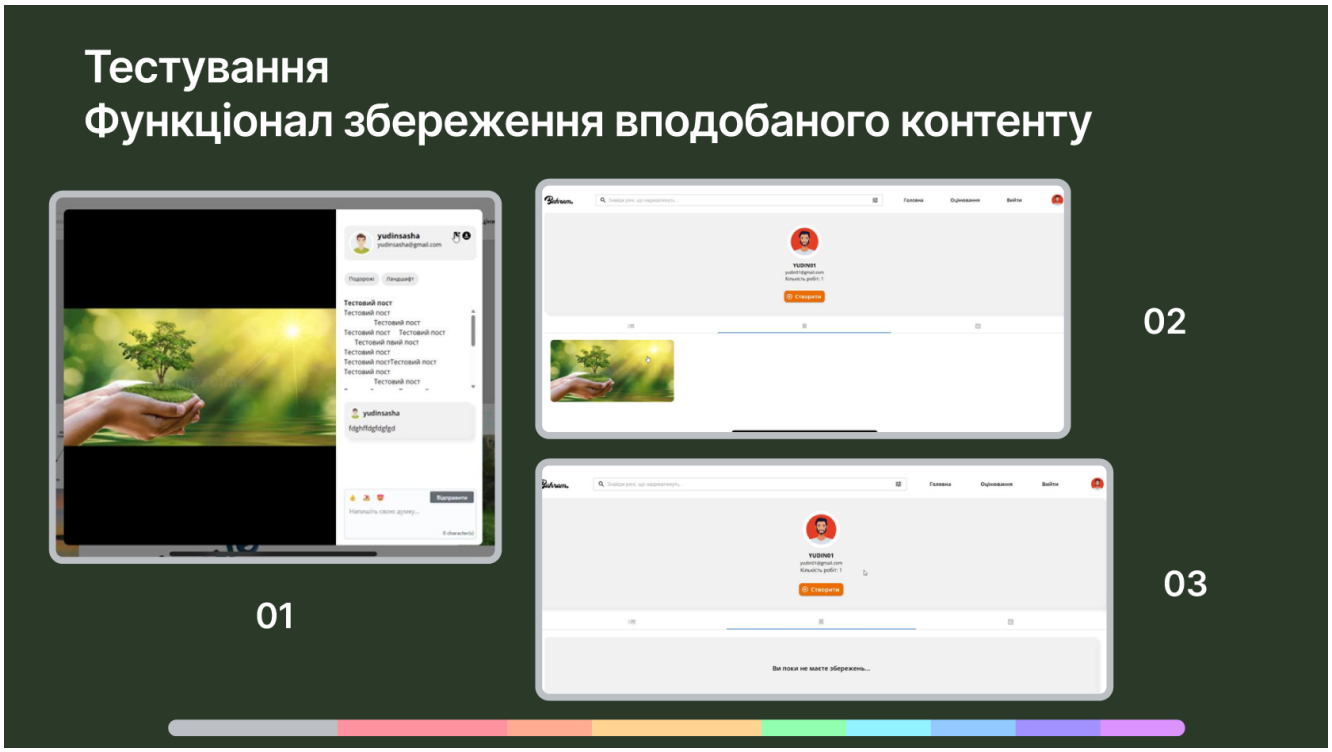

Рисунок Г.16 – Функціонал збереження вподобаного контенту

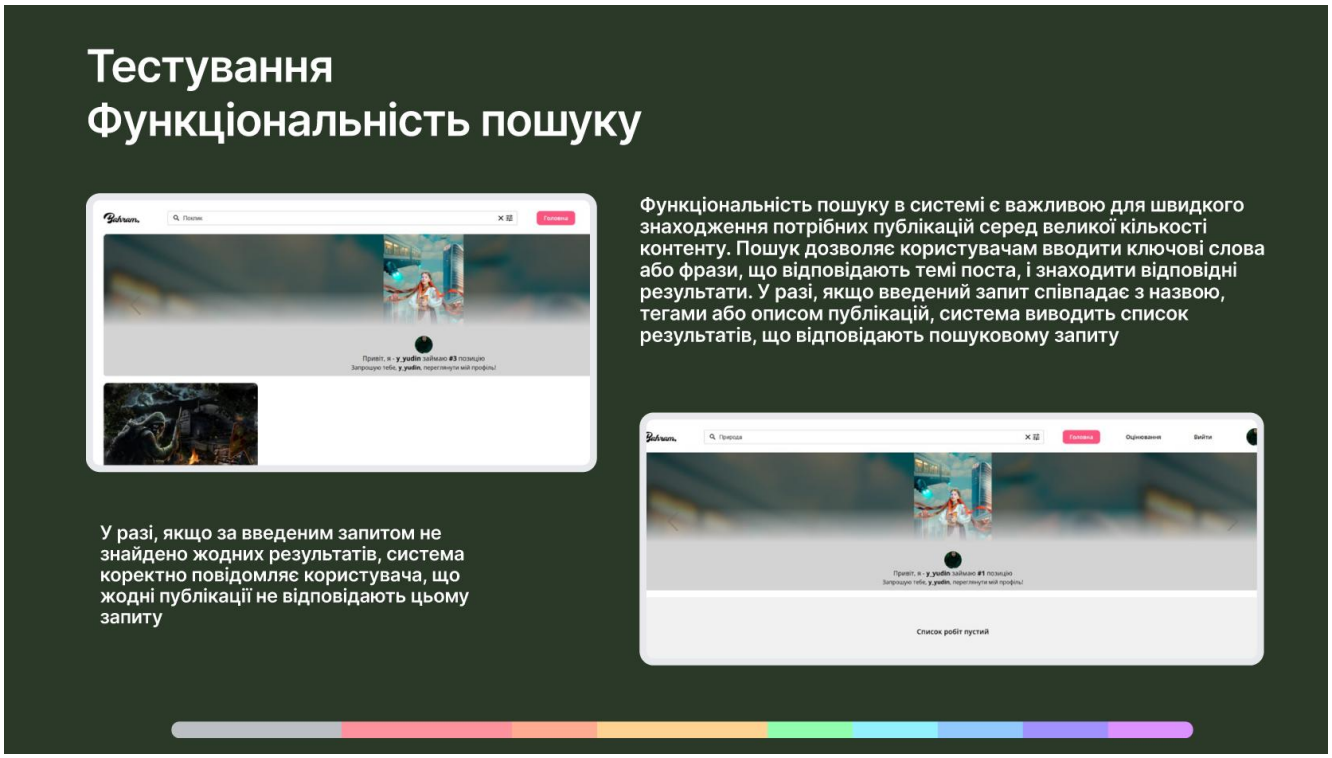

Рисунок Г.17 – Функціональність пошуку

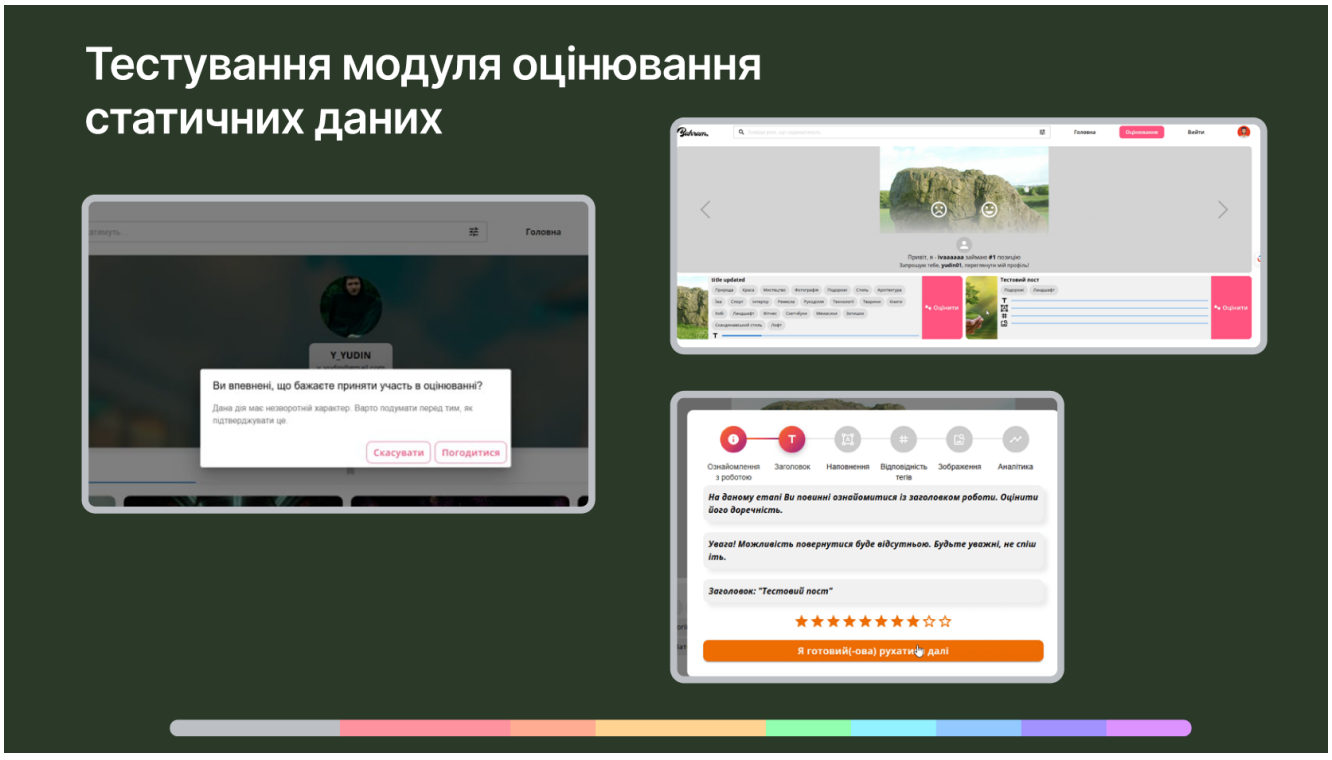

Рисунок Г.18 – Тестування модуля оцінювання статичних даних

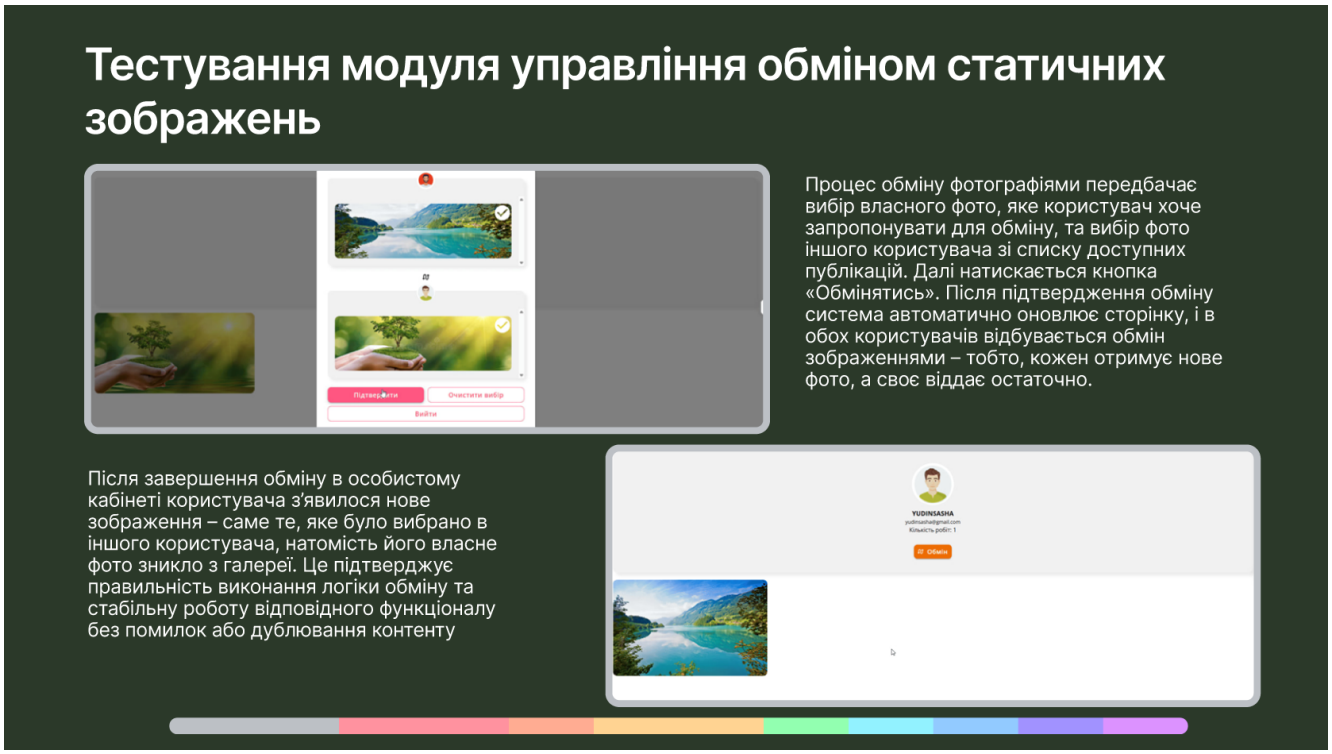

Рисунок Г.19 – Тестування модуля управління обміном статичних зображень

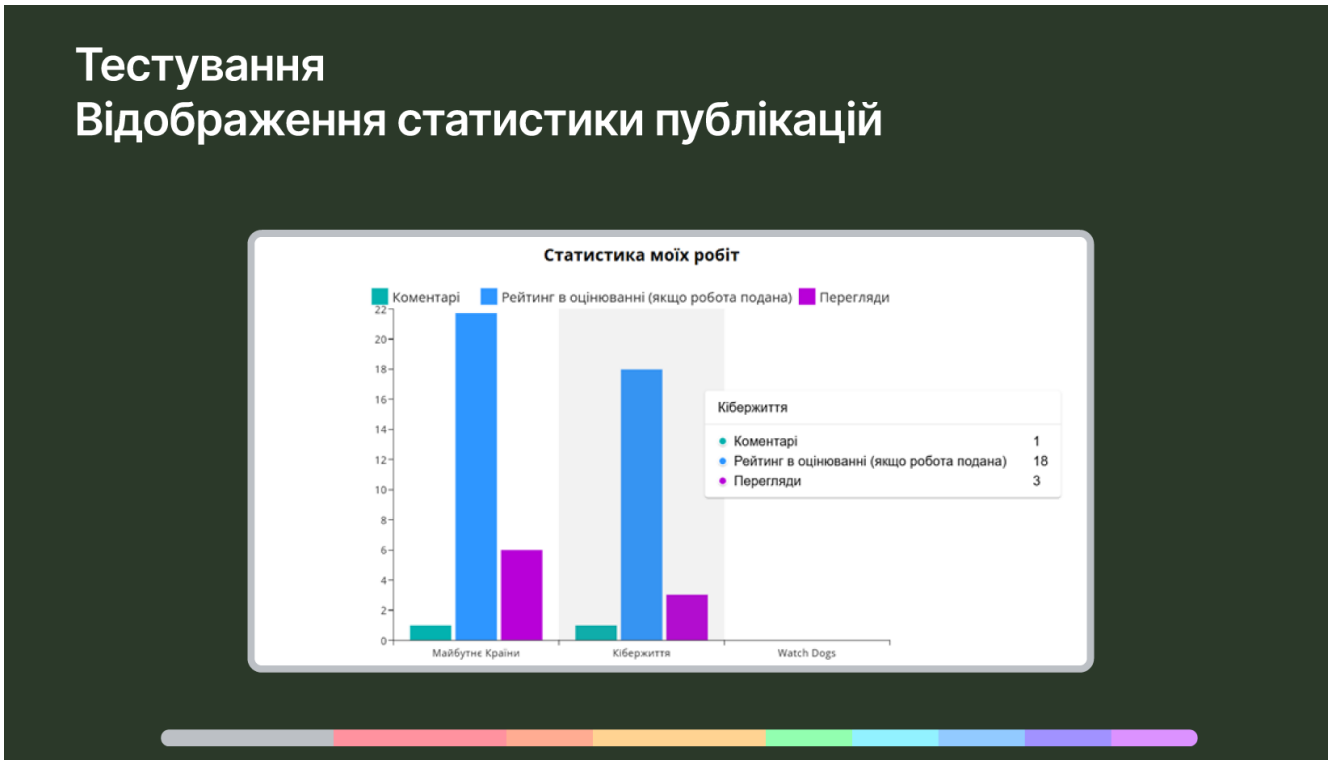

Рисунок Г.20 – Відображення статистики публікацій

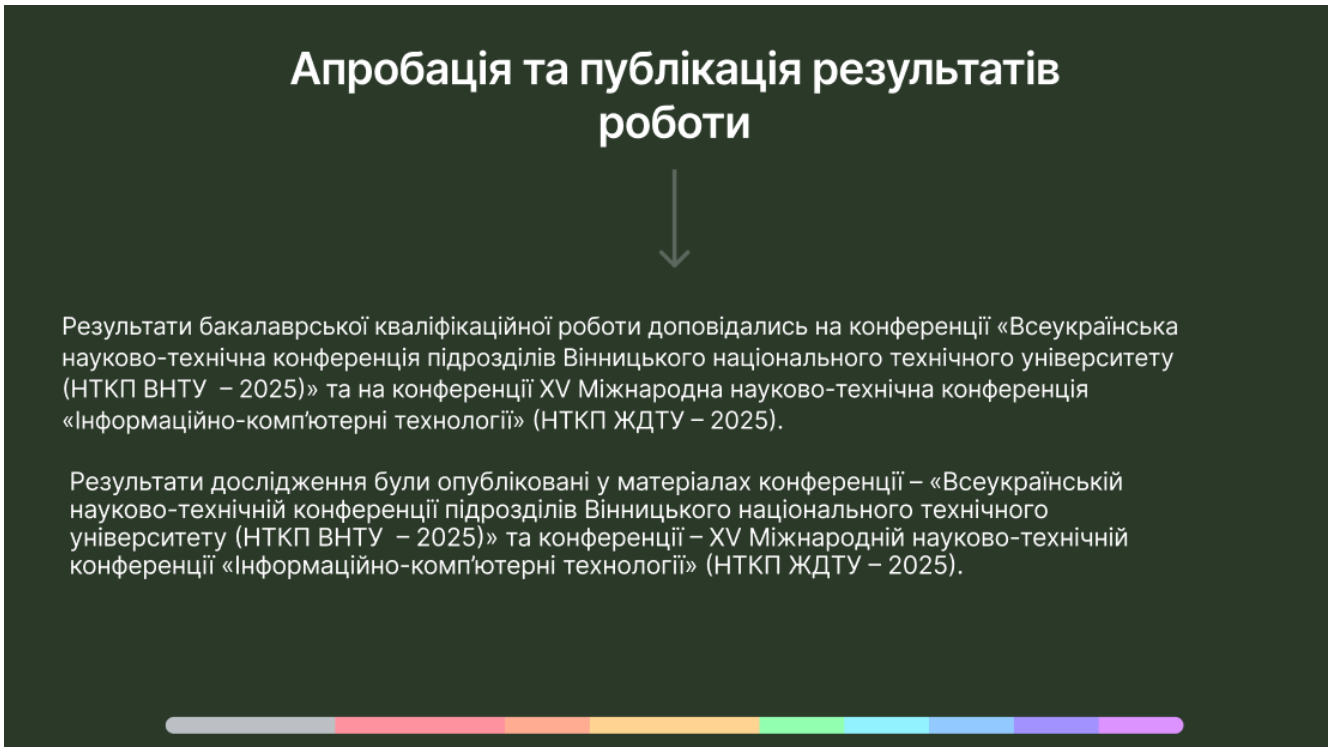

Рисунок Г.21 – Апробація та публікація результатів роботи

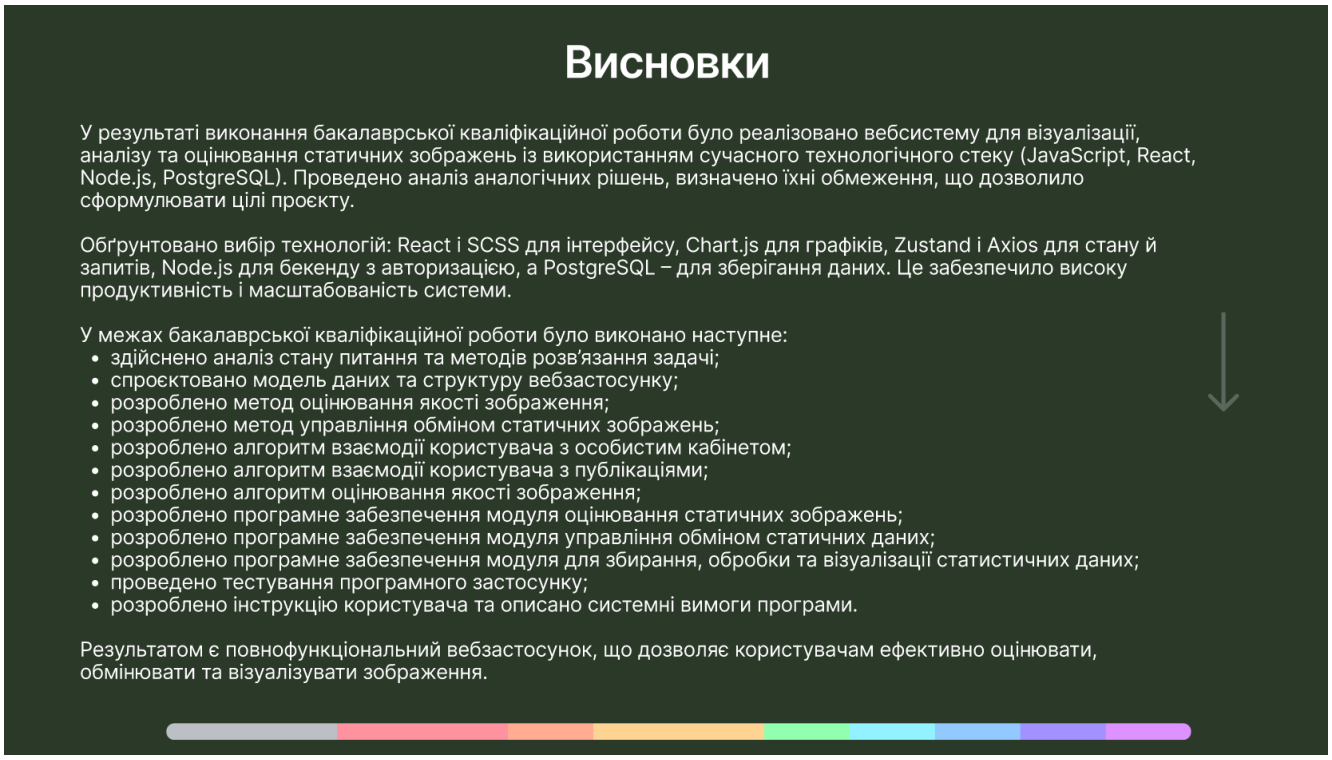

Рисунок Г.22 – Висновки