МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ОНЛАЙН-ПЛАТФОРМА ДЛЯ РЕКОМЕНДАЦІЙ ГЕНЕРАТИВНИХ ЗОБРАЖЕНЬ

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 401.22010130

Виконав студент 4-го курсу, групи 401
_____В. Д. Малахов
«21» червня 2024 р.

Керівник: д-р техн. наук, доцент
_____О. В. Козлов
«21» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р. техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я

на виконання кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук Малахову Владиславу Дмитровичу.

1.Тема кваліфікаційної роботи «Онлайн-платформа для рекомендацій генеративних зображень».

Керівник роботи Козлов Олексій Валерійович, д-р техн. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2024 р. № 271.

2. Строк представлення кваліфікаційної роботи студентом «21» червня 2024 р.

3. Вхідні (початкові) дані до роботи: колекція згенерованих зображень, промти (підказки) для генерації зображень, генерація зображень.

Очікуваний результат: онлайн-платформа з функціоналом рекомендацій, рейтингу, фільтрації та генерації зображень.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– аналіз предметної області та існуючих рішень;

– вибір та обґрунтування методів рекомендації, рейтингу та фільтрації зображень;

– проектування структури та функціоналу платформи;

– вибір та обґрунтування технологій розробки (HTML, CSS, JavaScript, Node.js, Masonry, Stability AI, Replicate);

– розробка інтерфейсу користувача та серверної частини;

– тестування роботи платформи.

5. Перелік графічного матеріалу: рисунки, таблиці, презентація.

6. Завдання до спеціальної частини: «Комплексна оцінка умов праці при розробці онлайн-платформи для рекомендацій генеративних зображень».

7. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	А. О. Алексєєва, доцент кафедри екології	

Керівник роботи __д-р техн. наук, доцент Козлов О. В. _____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Малахов В. Д._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «___14___» _____січня_____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Онлайн-платформа для рекомендацій генеративних зображень

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз предметної області, дослідження методів рекомендацій та фільтрації, проектування та реалізація веб-інтерфейсу, розробка серверної логіки	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
10	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
12	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
13	Захист КРБ перед екзаменаційною комісією (ЕК)	28.06.2024	28.06.2024	

Розробив студент _____Малахов В. Д._____ _____
(прізвище та ініціали) *(підпис)*

Керівник _____д-р техн. наук, доцент Козлов О. В.___ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«__29__» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили
Малахова Владислава Дмитровича
Тема: «Онлайн-платформа для рекомендацій генеративних зображень»

Кваліфікаційна робота бакалавра присвячена розробці онлайн-платформи для рекомендацій генеративних зображень, створених за допомогою нейронних мереж. Платформа покликана вирішити проблему пошуку та ефективного використання генеративних зображень, надаючи користувачам зручні інструменти для пошуку, оцінки та взаємодії з унікальним візуальним контентом.

Об'єкт роботи – процес рекомендації генеративних зображень.

Предмет роботи – методи генерації зображень на основі нейронних мереж та технології розробки онлайн-платформ для рекомендації зображень.

Мета роботи – підвищення зручності перегляду, дослідження та взаємодії з колекцією згенерованих нейромережами зображень шляхом розробки онлайн-платформи. Одним із завдань є забезпечення персоналізованого досвіду користувачів, стимулювання творчості та сприяння формуванню спільноти однодумців, зацікавлених у генеративному мистецтві.

Кваліфікаційна робота бакалавра складається з фахової та спеціальної частини з охорони праці. Пояснювальна записка фахової частини складається зі вступу, трьох розділів, висновків та додатків. У першому розділі проведено аналіз предметної області та поставлено задачу розробки. У другому розділі розглянуто моделі, методи та інформаційні технології для вирішення поставленої задачі. У третьому розділі розглянуто програмну реалізацію та тестування.

Кваліфікаційна робота бакалавра містить 71 сторінок (без додатків), 16 рисунків, 1 таблиць, 25 джерел, 5 додатків.

Ключові слова: генеративні зображення, нейронні мережі, HTML, CSS, JavaScript, Node.js, Masonry, API, Stability AI, Replicate.

ABSTRACT

bachelor's qualification work of a student of 401 group at Petro Mohyla Black Sea National University

Malakhova Vladislava

Theme: «Online platform for generative image recommendations»

The bachelor's thesis is devoted to the development of an online platform for recommending generative images created using neural networks. The platform is designed to solve the problem of finding and effectively using generative images by providing users with convenient tools for finding, evaluating, and interacting with unique visual content.

The object of work is the process of recommending generative images.

The subject of the work is methods and algorithms for generating images based on neural networks and technologies for developing online platforms for image recommendation.

The purpose of the study is to improve the convenience of viewing, researching, and interacting with a collection of neural network-generated images by developing an online platform. One of the tasks is to provide a personalized user experience, stimulate creativity, and foster a community of like-minded people interested in generative art.

The bachelor's thesis consists of a professional and a special part on occupational health and safety. The explanatory note of the professional part consists of an introduction, three chapters, conclusions and appendices. The first chapter analyzes the subject area and sets the development task. The second section discusses models, methods and information technologies for solving the task. The third chapter deals with the program implementation and testing.

The bachelor's thesis contains 71 pages (without appendices), 16 figures, 1 tables, 25 sources, 5 appendices.

Keywords: generative images, neural networks, HTML, CSS, JavaScript, Node.js, Masonry, API, Stability AI, Replicate..

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КРБ – кваліфікаційна робота бакалавра

AI – Artificial Intelligence

API – Application Programming Interface

CSS – Cascading Style Sheets

CDN – Content Delivery Network

GAN – Generative Adversarial Networks

HTML – HyperText Markup Language

JS – JavaScript

ML – Machine Learning

NFT – Non-Fungible Token

NPM – Node Package Manager

VAE – Variational Autoencoders

VS – Visual Studio

ВСТУП

Генеративне мистецтво, що базується на використанні алгоритмів та штучного інтелекту для створення унікальних творів, стрімко набирає популярності. Генеративні зображення, зокрема, знаходять застосування в дизайні, рекламі, розвагах та інших сферах, відкриваючи нові можливості для творчого самовираження та вирішення практичних завдань. Від створення ілюстрацій та концепт-артів до генерації унікальних візуальних ефектів – потенціал генеративних зображень величезний.

Однак, зі зростанням обсягів контенту виникає проблема його ефективного пошуку та використання. Існуючі онлайн-галереї та платформи часто не відповідають потребам користувачів. Вони можуть мати обмеження у функціоналі, такі як відсутність якісних рекомендацій, недостатня інформація про параметри генерації (промти), незручні фільтри для пошуку. Це ускладнює пошук потрібних зображень та обмежує можливості їх використання.

Об'єкт роботи – процес рекомендації генеративних зображень.

Предмет роботи – методи генерації зображень на основі нейронних мереж та технології розробки онлайн-платформ для рекомендації зображень.

Мета роботи – підвищення зручності перегляду, дослідження та взаємодії з колекцією згенерованих нейромережами зображень шляхом розробки онлайн-платформи. Одним із завдань є забезпечення персоналізованого досвіду користувачів, стимулювання творчості та сприяння формуванню спільноти однодумців, зацікавлених у генеративному мистецтві.

Для досягнення **поставленої мети** необхідно вирішити наступні завдання:

1) провести аналіз існуючих технологій генерації зображень та їх можливостей;

2) дослідити потреби користувачів та проаналізувати існуючі онлайн-платформи для роботи з генеративними зображеннями;

3) розробити структуру та функціонал платформи, включаючи галерею зображень, систему рекомендацій, рейтингу, фільтрів та можливість генерації нових зображень;

4) обрати та використати оптимальні технології для реалізації платформи, такі як html, css, javascript, node.js та бібліотеку masonry для створення адаптивної галереї;

5) розробити інтуїтивно зрозумілий інтерфейс користувача, забезпечуючи зручність навігації та взаємодії з платформою;

6) провести тестування та оптимізацію роботи платформи, забезпечуючи її стабільність, швидкість та ефективність.

Очікується, що в результаті виконання роботи буде створено онлайн-платформу, яка суттєво полегшить пошук, оцінку та використання генеративних зображень. Це сприятиме підвищенню доступності та якості генеративного мистецтва, а також розвитку творчості та інновацій у цій сфері.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери. Онлайн-галереї згенерованих нейромережами зображень

Онлайн-галереї згенерованих нейромережами зображень – це захоплива та інноваційна галузь, що поєднує в собі мистецтво, технології та соціальну взаємодію. Ці платформи відкривають нові горизонти для творчості та самовираження, надаючи користувачам доступ до унікальних візуальних творів, створених за допомогою штучного інтелекту. Вони стирають межі між традиційним та цифровим мистецтвом, надаючи нові можливості для творчого самовираження та експериментів, а також стають місцем для натхнення, навчання та обміну досвідом.

Ключові аспекти предметної сфери:

1) джерела згенерованих зображень:

– різноманітність нейромереж. Світ генеративних моделей постійно розширюється, пропонуючи все нові й нові інструменти для створення зображень. GAN (Generative Adversarial Networks), наприклад, StyleGAN від NVIDIA, здатні генерувати фотореалістичні обличчя, пейзажі та об'єкти, що вражають своєю деталізацією та реалістичністю. VAE (Variational Autoencoders) дозволяють створювати зображення з контрольованими характеристиками, такими як колір, форма та текстура, що відкриває широкі можливості для творчих експериментів. Дифузійні моделі, такі як Stable Diffusion від Stability AI та DALL-E 2 від OpenAI, вражають своєю здатністю генерувати зображення з високою роздільною здатністю та різноманітними стилями, від імпресіонізму до кіберпанку, за текстовими описами;

– відкриті та закриті платформи. Онлайн-галереї можуть використовувати різні джерела зображень. Деякі з них, такі як Artbreeder, надають користувачам можливість генерувати зображення безпосередньо на

платформі, використовуючи вбудовані генеративні моделі, що дозволяє їм експериментувати та створювати власні унікальні твори. Інші галереї, такі як NightCafe Creator, можуть співпрацювати з художниками, які використовують нейромережі у своїй творчості, або купувати зображення на спеціалізованих маркетплейсах, таких як PromptBase, надаючи користувачам доступ до широкого спектру готових робіт;

2) кураторство та організація контенту:

– тематичні колекції. Щоб полегшити навігацію та пошук потрібних зображень, галереї часто організовують їх у тематичні колекції, що дозволяє користувачам швидко знаходити зображення, що відповідають їхнім інтересам. Наприклад, галерея може мати колекції, присвячені певним стилям (наприклад, абстракція, поп-арт, аніме), жанрам (пейзажі, портрети, натюрморти) або настроям (романтика, меланхолія, енергія);

– кураторські підбірки. Деякі платформи, такі як deviantArt або Behance, запрошують експертів у галузі мистецтва та дизайну для створення кураторських підбірок зображень. Це дозволяє користувачам відкривати нові таланти, знайомитися з актуальними трендами та розширювати свій кругозір;

– персоналізовані рекомендації. Використання алгоритмів рекомендацій, заснованих на машинному навчанні, допомагає галереям пропонувати користувачам зображення, які відповідають їхнім інтересам та вподобанням. Наприклад, платформа може аналізувати історію переглядів користувача, його оцінки та коментарі, щоб рекомендувати йому нові зображення, які можуть його зацікавити;

3) інтерактивність та соціальні функції:

– коментарі та обговорення. Можливість залишати коментарі під зображеннями та брати участь в обговореннях створює відчуття спільноти та сприяє обміну думками та ідеями між користувачами. Це також може бути корисним для авторів зображень, які отримують зворотний зв'язок та можуть взаємодіяти зі своєю аудиторією;

– оцінки та рейтинги. Системи оцінювання та рейтингу, такі як "лайки" або зірки, дозволяють користувачам висловлювати свою думку про зображення та допомагають виявити найпопулярніші та найцікавіші роботи. Для авторів це може бути стимулом для подальшої творчості та розвитку;

– створення колекцій. Можливість створювати власні колекції зображень надає користувачам інструмент для персоналізації та організації контенту. Це також може бути способом самовираження та демонстрації своїх інтересів та смаків;

4) презентація зображень та метадані:

– промти. Розкриття промтів, використаних для генерації зображень, надає користувачам уявлення про те, як автори взаємодіють з нейромережами та які ідеї вони намагаються втілити. Це також може бути джерелом натхнення для інших користувачів, які бажають експериментувати з генеративними моделями. Наприклад, платформа може відображати не лише текст промту, а й додаткові параметри, такі як стиль, розмір та інші налаштування, що впливають на результат генерації;

– технічні деталі. Інформація про модель, налаштування та параметри генерації може бути цікавою для технічно підкованих користувачів, дослідників та розробників. Це також сприяє прозорості та відкритості процесу створення зображень. Наприклад, платформа може вказувати, яка версія моделі була використана, які гіперпараметри були налаштовані та які обчислювальні ресурси були задіяні;

– візуальне оформлення. Якісне відображення зображень, зручна навігація та адаптивний дизайн є критично важливими для забезпечення позитивного користувацького досвіду. Гарний дизайн допомагає користувачам зосередитися на контенті та насолоджуватися переглядом галереї. Наприклад, платформа може використовувати різні режими перегляду (галерея, список, повноекранний режим), фільтри для сортування зображень за різними

критеріями (дата, популярність, стиль) та інші інструменти для зручності користувачів.

Актуальність та перспективи онлайн-галереї згенерованих зображень відіграють важливу роль у популяризації та розвитку мистецтва, створеного за допомогою штучного інтелекту. Вони стирають межі між традиційним та цифровим мистецтвом, надаючи нові можливості для творчого самовираження та експериментів, а також стають місцем для натхнення, навчання та обміну досвідом.

З розвитком технологій ШІ можна очікувати ще більшого зростання якості та різноманітності згенерованих зображень. Поява нових форматів, таких як 3D-зображення та анімація, а також інтерактивних можливостей, таких як віртуальні виставки та спільне створення зображень, відкриває нові горизонти для розвитку цієї галузі.

Онлайн-галереї згенерованих зображень мають потенціал стати не лише місцем для демонстрації та споглядання мистецтва, а й платформою для навчання, співпраці та інновацій. Вони можуть сприяти розвитку креативних індустрій, дизайну, реклами та інших сфер, де візуальний контент відіграє важливу роль.

1.2 Огляд та аналіз наявних аналогів та публікацій

Аналіз існуючих онлайн-галерей згенерованих зображень та огляд наукових публікацій дозволяє виявити актуальні тенденції, найкращі практики та потенційні напрямки розвитку в цій галузі. Це важливий етап для розуміння конкурентного середовища, визначення унікальності власного проекту та обґрунтування його актуальності. Аналіз аналогів:

Існує безліч онлайн-галерей згенерованих зображень, кожна з яких має свої особливості та цільову аудиторію. Розглянемо деякі з найпопулярніших:

1) *artbreeder:* платформа [20], що дозволяє користувачам створювати унікальні зображення шляхом схрещування та змішування різних зображень-батьків (рис. 1.1). Вона використовує власну генеративну модель, засновану на

GAN, та надає широкий спектр інструментів для редагування та налаштування зображень, включаючи регулювання рис обличчя, кольору волосся, одягу та фону. Artbreeder особливо популярний серед художників та дизайнерів, які шукають нові джерела натхнення та експериментують з різними стилями;

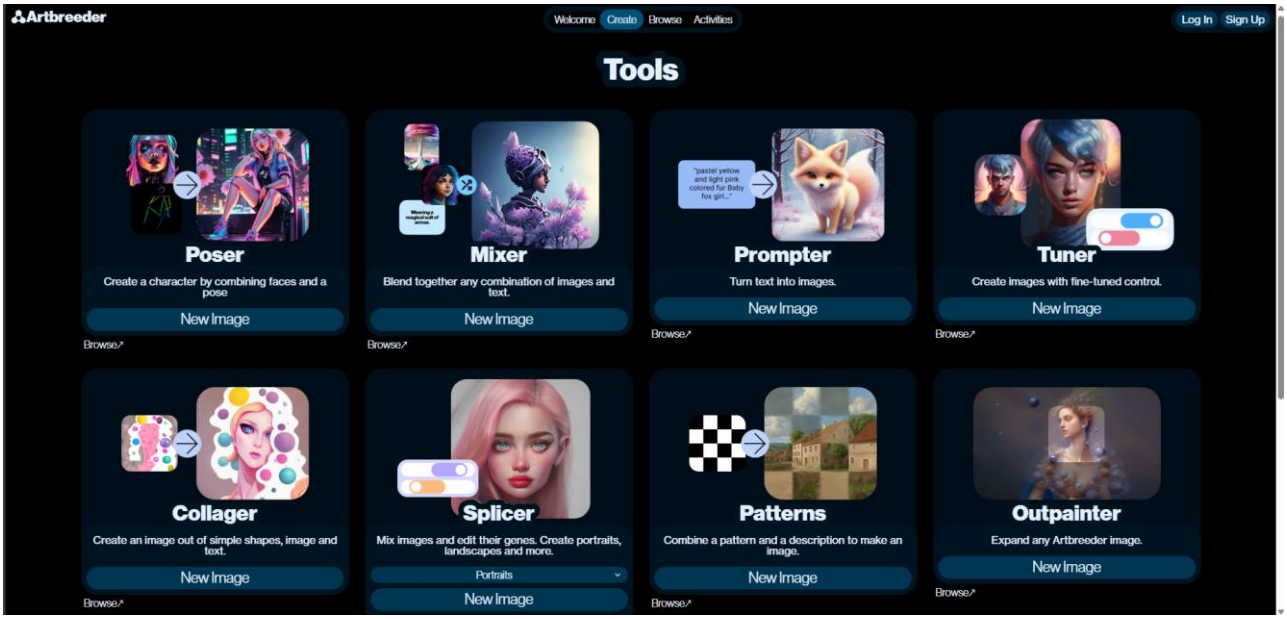

Рисунок 1.1 – Artbreeder

2) *nightCafe Creator:* галерея [21], що пропонує користувачам різноманітні генеративні моделі, включаючи Stable Diffusion та DALL-E 2, для створення зображень за текстовими описами (рис. 1.2). Вона має зручний інтерфейс, що дозволяє легко експериментувати з різними стилями та налаштуваннями. NightCafe Creator також має активну спільноту, де користувачі можуть ділитися своїми роботами, брати участь у конкурсах та отримувати зворотний зв'язок;

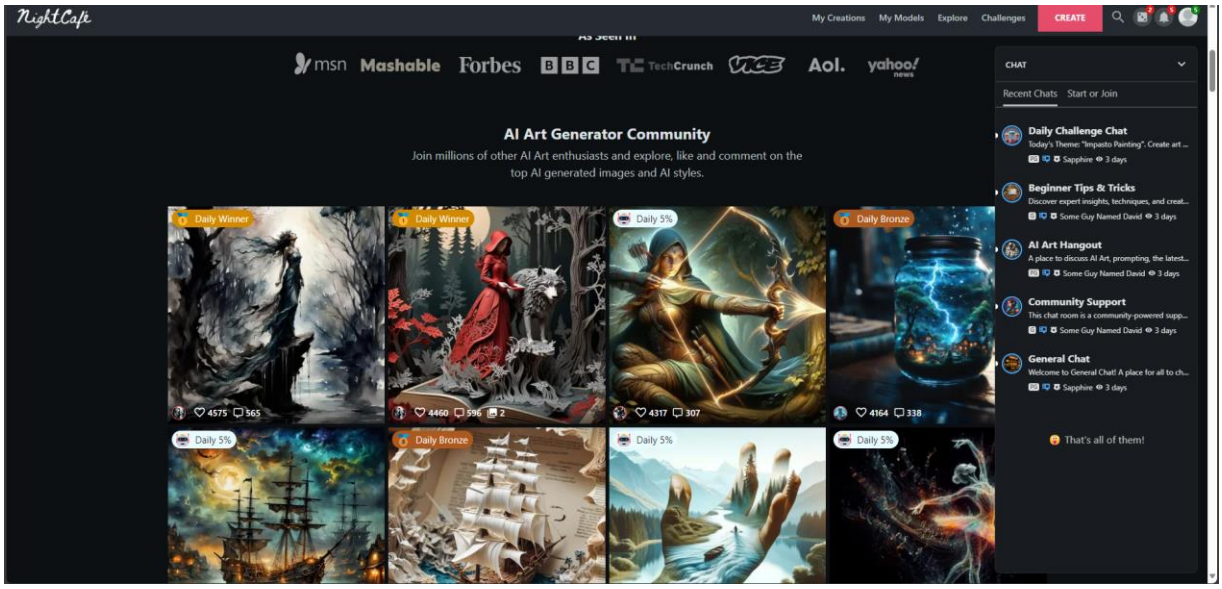

Рисунок 1.2 – NightCafe Creator

3) *lexica:* пошукова система та галерея зображень, згенерованих за допомогою Stable Diffusion (рис. 1.3). Вона дозволяє користувачам шукати зображення за ключовими словами, переглядати популярні запити та знаходити натхнення для власних проектів. Lexica також надає детальну інформацію про кожне зображення, включаючи промт, модель, налаштування та інші параметри, що може бути корисним для дослідників та розробників [22];

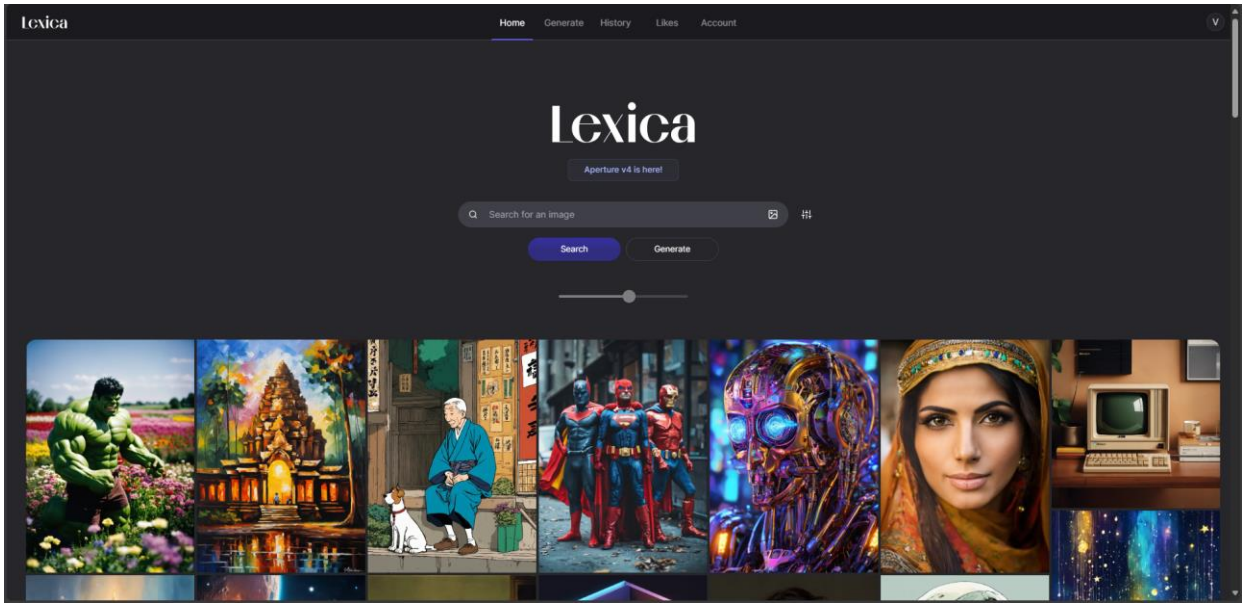

Рисунок 1.3 – Lexica

4) *openSea:* найбільший маркетплейс для NFT (рис. 1.4), де можна знайти та придбати унікальні цифрові твори мистецтва, включаючи згенеровані зображення. Він пропонує широкий вибір робіт різних авторів, від відомих художників до новачків, а також можливість створювати та продавати власні NFT. OpenSea є популярною платформою для колекціонерів [23], інвесторів та ентузіастів NFT, які шукають унікальні та цінні цифрові активи.

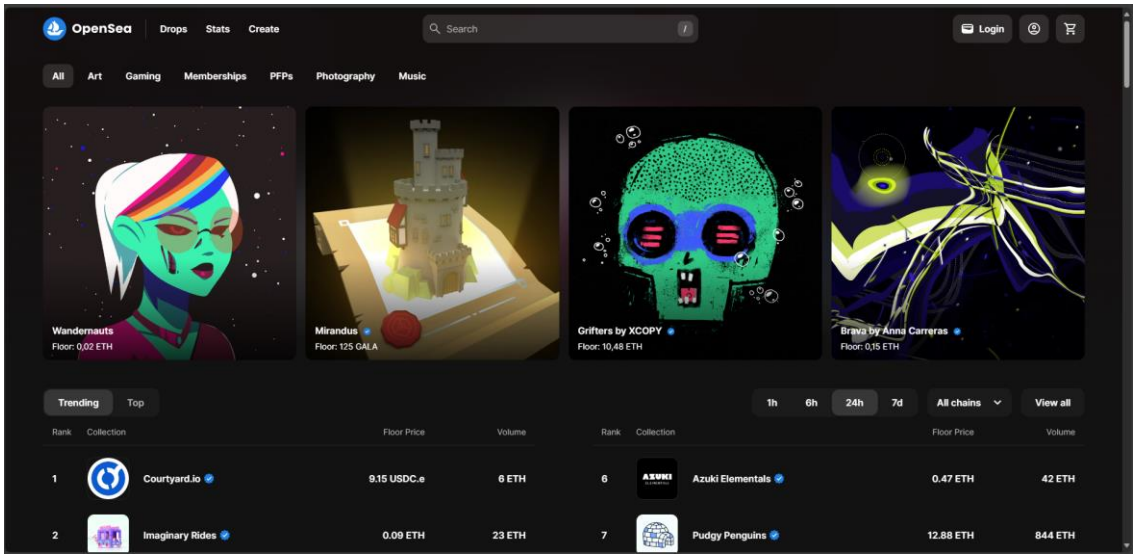

Рисунок 1.4 – OpenSea

Таблиця 1.1 – Порвіняння аналогів

Критерій	**Artbreeder**	**NightCafe Creator**	**Lexica**	**OpenSea**
Модель генерації	Власна (GAN)	Stable Diffusion, DALL-E 2	Stable Diffusion	Різні (залежить від автора)
Інструменти редагування	Широкий вибір (регулювання рис обличчя, кольору волосся тощо)	Обмежений (налаштування стилю, розміру)	Відсутні	Відсутні
Спільнота	Активна	Дуже активна (конкурси, обговорення)	Менш активна	Велика та різноманітна

Закінчення таблиці 1.1

Критерій	**Artbreeder**	**NightCafe Creator**	**Lexica**	**OpenSea**
Монетизація	Продаж підписок, комісійні з продажів NFT	Продаж підписок, друк зображень, комісійні з NFT	Безкоштовний доступ	Комісійні з продажів NFT
Цільова аудиторія	Художники, дизайнери	Широка аудиторія	Художники, дизайнери, дослідники ШІ	Колекціонери, інвестори, ентузіасти NFT

1) огляд наукових публікацій. Наукові дослідження в галузі генеративних моделей зображень активно розвиваються. Ось деякі ключові напрямки досліджень:

– покращення якості та різноманітності згенерованих зображень. Розробляються нові архітектури нейронних мереж, такі як StyleGAN3 та Latent Diffusion Models, які дозволяють генерувати ще більш реалістичні та деталізовані зображення з високою роздільною здатністю. Також досліджуються методи підвищення різноманітності зображень, щоб уникнути повторюваності та стереотипів;

– контроль та управління процесом генерації. Вчені працюють над створенням методів, що дозволяють користувачам більш точно контролювати характеристики згенерованих зображень, такі як стиль, зміст, композиція та інші аспекти. Це може включати використання текстових підказок, зображень-референсів, а також інтерактивних інструментів для редагування та налаштування зображень;

– застосування генеративних моделей у різних сферах. Вивчаються можливості використання генеративних моделей у дизайні, рекламі, кіноіндустрії, медицині, архітектурі та інших галузях. Наприклад, генеративні моделі можуть бути використані для створення унікальних дизайнів продуктів,

персоналізованої реклами, реалістичних спецефектів у фільмах або навіть для моделювання нових лікарських препаратів;

– етичні та соціальні аспекти. Розглядаються питання авторського права на згенеровані зображення, потенційного зловживання технологіями (наприклад, для створення діпфейків) та впливу генеративних зображень на суспільство. Важливо розробити етичні принципи та рекомендації щодо використання генеративних моделей, щоб забезпечити їх відповідальне та безпечне застосування;

2) висновки. Аналіз аналогів та огляд наукових публікацій дозволяють зробити такі висновки:

– існуючі онлайн-галереї згенерованих зображень пропонують різноманітні можливості для користувачів, але кожна з них має свої переваги та недоліки. Деякі платформи фокусуються на спільноті та творчості, інші - на монетизації та колекціонуванні;

– важливими факторами успіху галереї є якість та різноманітність зображень, зручність використання, наявність активної спільноти, можливості для монетизації контенту та відповідність етичним принципам;

– наукові дослідження в галузі генеративних моделей активно розвиваються, відкриваючи нові перспективи для створення більш реалістичних, різноманітних та контрольованих зображень, а також для їх застосування у різних сферах життя.

Цей аналіз допоможе визначити унікальність та конкурентні переваги вашого проекту, а також сформулювати основні завдання та вимоги до розробки платформи.

1.3 Постановка задачі. Технічне завдання

Створення онлайн-галереї згенерованих нейромережами зображень з акцентом на зручність використання, персоналізацію та соціальну взаємодію є

актуальною та перспективною задачею. Враховуючи аналіз існуючих аналогів та тенденції розвитку галузі, можна сформулювати наступне технічне завдання для проекту:

Мета проекту – розробка онлайн-платформи, яка дозволить користувачам зручно переглядати, досліджувати та взаємодіяти з колекцією згенерованих нейромережами зображень. Платформа має забезпечити персоналізований досвід, стимулювати творчість та сприяти формуванню спільноти однодумців, зацікавлених у генеративному мистецтві.

Основні завдання:

1) створення бази даних зображень:

1) збір та систематизація:

– визначення критеріїв відбору зображень (тематика, стиль, якість, ліцензія);

– автоматизований збір зображень з відкритих джерел (наприклад, за допомогою веб-скрейпінгу);

– ручний відбір та додавання зображень від художників та з маркетплейсів;

– каталогізація зображень за різними параметрами (теги, категорії, автори, моделі);

2) структура бази даних:

– проектування реляційної або нереляційної бази даних для зберігання зображень та метаданих;

– вибір оптимального формату зберігання зображень (JPEG, PNG, WebP) для балансу між якістю та розміром файлів;

– розробка механізмів індексації та пошуку для швидкого доступу до зображень;

2) розробка функціоналу платформи:

1) перегляд та пошук зображень:

– реалізація різних режимів перегляду (галерея, список, повноекранний режим з можливістю масштабування);

– розробка інтуїтивно зрозумілого пошуку з підтримкою фільтрів за різними параметрами (ключові слова, теги, автори, моделі, колірна гамма, стиль);

– впровадження механізмів автозаповнення та підказок для пошукових запитів;

2) фільтрація та сортування:

– надання можливості фільтрувати зображення за різними критеріями (тематика, стиль, колірна гамма, дата створення, популярність, рейтинг);

– реалізація гнучкого сортування за різними параметрами (дата додавання, популярність, рейтинг, випадковий порядок);

3) перегляд деталей зображення:

– відображення промтів, моделей, налаштувань та іншої технічної інформації про зображення;

– можливість копіювання промтів для подальшого використання в інших генеративних моделях;

– відображення інформації про автора зображення (ім'я, посилання на профіль);

4) соціальні функції:

– коментування та обговорення зображень;

– оцінювання зображень за допомогою "лайків" або рейтингової системи;

– можливість створювати власні колекції зображень та ділитися ними з іншими користувачами;

– підписка на улюблених авторів та отримання сповіщень про нові роботи;

5) персоналізовані рекомендації:

– використання алгоритмів колаборативної фільтрації, контентної фільтрації або гібридних підходів для формування персоналізованих рекомендацій;

– врахування історії переглядів, оцінок, коментарів та інших дій користувача для покращення якості рекомендацій;

– можливість налаштування параметрів рекомендацій (наприклад, вибір стилів або тематик, які цікавлять користувача);

3) дизайн та розробка інтерфейсу:

1) інтуїтивно зрозумілий дизайн:

– розробка логічної та зручної структури платформи, що дозволить користувачам легко знаходити потрібний контент та функції;

– використання зрозумілих іконок, кнопок та інших елементів інтерфейсу;

– надання підказок та інструкцій для нових користувачів;

2) адаптивний дизайн:

– забезпечення коректної роботи та відображення платформи на різних пристроях (комп'ютери, планшети, смартфони) з різними розмірами екранів;

– використання технологій адаптивного дизайну (css grid, flexbox) для створення гнучкого та масштабованого інтерфейсу.

3) візуальна привабливість:

– вибір сучасного та естетичного стилю дизайну, який відповідає тематиці платформи;

– використання якісних зображень, шрифтів та кольорової палітри;

– створення унікального візуального образу, який запам'ятається користувачам;

4) Технічна реалізація:

1) вибір технологічного стеку:

– обґрунтування вибору мов програмування (наприклад, JavaScript, Python), фреймворків (React, Vue.js, Django, Flask), баз даних (PostgreSQL, MongoDB) та інших інструментів з урахуванням вимог до продуктивності, масштабованості та безпеки платформи;

– врахування досвіду та знань команди розробників;

2) розробка бекенду:

– проектування архітектури бекенду, включаючи вибір серверного програмного забезпечення (наприклад, Nginx, Apache), налаштування кешування, балансування навантаження та інших механізмів для забезпечення високої продуктивності та надійності;

– розробка API для взаємодії фронтенду з бекендом;

– реалізація функцій обробки запитів, взаємодії з базою даних, генерації рекомендацій та інших серверних задач;

3) розробка фронтенду:

– створення компонентів інтерфейсу (кнопки, форми, списки, галереї) з використанням обраного фреймворку;

– реалізація логіки взаємодії з користувачем (обробка подій, валідація даних, анімація);

– інтеграція з API бекенду для отримання та відображення даних;

5) тестування та оптимізація:

1) функціональне тестування:

– створення тестових сценаріїв для перевірки коректності роботи всіх функцій платформи;

– виявлення та виправлення помилок;

2) навантажувальне тестування:

– симуляція великої кількості користувачів та запитів для оцінки продуктивності платформи під навантаженням;

– виявлення вузьких місць та оптимізація роботи системи;

3) оптимізація:

– вдосконалення швидкості завантаження сторінок, зменшення споживання ресурсів сервера та клієнта, покращення якості зображень;

– використання технік кешування, стиснення даних та інших методів оптимізації;

6) вимоги до проекту:

1) функціональні вимоги:

– платформа має забезпечувати всі функції, описані в основних завданнях;

– зображення мають відображатися в високій якості з можливістю масштабування та перегляду деталей;

– пошук та фільтрація мають працювати швидко та ефективно, надаючи релевантні результати;

– персоналізовані рекомендації мають бути точними, різноманітними та враховувати інтереси користувача.

Висновки до розділу 1

Галузь генеративного мистецтва, особливо генерації зображень, стрімко розвивається завдяки постійному вдосконаленню нейромереж, зростанню інтересу користувачів та розширенню сфер застосування. Онлайн-галереї відіграють ключову роль у популяризації та доступності генеративного мистецтва, надаючи користувачам зручні інструменти для пошуку, перегляду, використання та взаємодії з унікальним візуальним контентом. Вони сприяють демократизації мистецтва, дозволяючи кожному стати творцем та споживачем цифрових творів. З огляду на стрімкий розвиток технологій штучного інтелекту та зростаючий попит на унікальний візуальний контент, можна очікувати подальшого зростання популярності та розвитку онлайн-галерей згенерованих зображень.

Незважаючи на різноманітність існуючих платформ, вони мають певні обмеження, які створюють простір для вдосконалення та інновацій. Однією з ключових проблем є недостатня персоналізація рекомендацій, що призводить до того, що користувачі отримують нерелевантний контент та витрачають багато часу на пошук потрібних зображень. Крім того, існуючі платформи часто мають обмежений функціонал фільтрації та пошуку, що ускладнює пошук зображень за певними критеріями, такими як стиль, тематика, колірна гамма тощо. Також відсутність або недостатня розвиненість інтерактивних елементів та соціальних функцій обмежує можливості для взаємодії між користувачами та авторами, що може негативно впливати на розвиток спільноти та залучення нових користувачів.

Пропонований проект онлайн-галереї згенерованих нейромережами зображень спрямований на вирішення вищезазначених проблем та задоволення потреб широкої аудиторії користувачів. Він відрізняється від існуючих аналогів фокусом на персоналізацію, зручність використання та соціальну взаємодію. Інноваційність проекту полягає у використанні передових методів машинного навчання для формування персоналізованих рекомендацій, враховуючи інтереси та вподобання кожного користувача. Розширений функціонал фільтрації та пошуку дозволить швидко та ефективно знаходити потрібні зображення, а інтерактивні елементи та соціальні функції сприятимуть активній взаємодії між користувачами та авторами, формуванню спільноти та обміну досвідом.

Таким чином, розробка онлайн-галереї згенерованих нейромережами зображень з акцентом на персоналізацію, зручність використання та соціальну взаємодію є актуальною та перспективною задачею. Запропонований проект має значний потенціал для розвитку галузі генеративного мистецтва та задоволення потреб широкої аудиторії користувачів, сприяючи популяризації та доступності цього виду мистецтва. Він також може стати основою для подальших досліджень та розробок у сфері штучного інтелекту та машинного навчання.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Методи для вирішення задачі

Для досягнення поставленої мети – створення ефективної та зручної платформи для рекомендацій генеративних зображень – буде застосовано комплексний підхід, що поєднує в собі сучасні методи машинного навчання, обробки природної мови та аналізу зображень. Вибір конкретних методів обумовлений специфікою предметної області та вимогами до функціональності платформи.

Рекомендаційні системи є ключовим компонентом платформи, оскільки вони відповідають за формування персоналізованих пропозицій зображень для кожного користувача. Для реалізації рекомендацій будуть використані наступні методи:

1) колаборативна фільтрація (Collaborative Filtering) – метод базується на аналізі взаємодії користувачів з контентом. Алгоритми колаборативної фільтрації виявляють схожості між користувачами на основі їх оцінок, переглядів, лайків та інших дій. Наприклад, якщо два користувачі поставили високі оцінки одним і тим же зображенням, система може рекомендувати їм інші зображення, які сподобалися одному з них:

– *переваги:* колаборативна фільтрація добре працює навіть при відсутності детальної інформації про зображення, оскільки вона враховує думки інших користувачів;

– *недоліки:* цей метод може бути неефективним для нових користувачів або нових зображень, оскільки для них ще немає достатньо даних про взаємодію;

2) контентна фільтрація (Content-Based Filtering) – метод аналізує характеристики самих зображень, такі як колір, стиль, тематика, об'єкти на зображенні та інші візуальні ознаки. Для аналізу зображень можуть бути

використані методи комп'ютерного зору, такі як розпізнавання об'єктів, класифікація стилів та вилучення ознак. Також важливу роль відіграватимуть промти (підказки), які використовувалися для генерації зображень. Аналізуючи промти, можна визначити семантичний зміст зображення та рекомендувати його користувачам, які цікавляться схожими темами:

– *переваги:* контентна фільтрація дозволяє рекомендувати нові зображення, навіть якщо для них ще немає оцінок від користувачів. Вона також може враховувати детальну інформацію про зображення, що дозволяє формувати більш точні рекомендації;

– *недоліки:* цей метод вимагає наявності детальних метаданих про зображення, а також може бути обмежений у своїх можливостях, якщо характеристики зображень важко формалізувати;

3) гібридні підходи (Hybrid Approaches) – метод досягнення найкращих результатів планується поєднати колаборативну та контентну фільтрацію. Це дозволить враховувати як поведінку користувачів, так і характеристики зображень, що зробить рекомендації більш точними та персоналізованими. Наприклад, можна використовувати колаборативну фільтрацію для визначення загальних вподобань користувача, а потім застосувати контентну фільтрацію для вибору зображень, які найкраще відповідають цим вподобанням.

Для оцінки якості та популярності зображень буде впроваджено систему рейтингу. Користувачі зможуть оцінювати зображення, виставляючи їм оцінки або використовуючи інші форми взаємодії, такі як лайки, коментарі чи збереження.

Для розрахунку рейтингу будуть використані наступні підходи:

– *середня оцінка:* найпростіший спосіб розрахунку рейтингу – це обчислення середньої оцінки, яку поставили користувачі;

– *врахування кількості оцінок:* щоб уникнути ситуації, коли зображення з малою кількістю високих оцінок отримує завищений рейтинг, можна використовувати алгоритми, які враховують не лише середню оцінку, але й

кількість оцінок. Наприклад, можна використовувати зважену середню, де вага кожної оцінки залежить від кількості оцінок, які поставив користувач;

– *врахування часу:* рейтинг зображення може знижуватися з часом, щоб новіші зображення мали більше шансів бути поміченими;

– *врахування активності автора:* можна враховувати активність автора зображення, наприклад, кількість його публікацій, підписників тощо.

Для зручності пошуку потрібних зображень будуть реалізовані різноманітні фільтри, які дозволять користувачам звузити вибірку зображень за певними критеріями.

1) *фільтрація за метаданими:* користувачі зможуть фільтрувати зображення за такими метаданими, як теги, категорії, автор, модель генерації, дата створення. Наприклад, можна буде вибрати всі зображення, згенеровані певною моделлю, або всі зображення певного автора;

2) *фільтрація за візуальними ознаками:* для фільтрації за візуальними ознаками, такими як колірна гамма, стиль, наявність певних об'єктів, будуть використані методи комп'ютерного зору. Наприклад, користувач зможе вибрати зображення з переважанням певних кольорів або зображення у певному стилі (наприклад, імпресіонізм, поп-арт, аніме);

3) *фільтрація за промтами:* користувачі зможуть вводити ключові слова або фрази, які описують бажаний зміст зображення, і система буде шукати зображення, промти яких містять ці слова або фрази. Це дозволить користувачам знаходити зображення, які відповідають їхнім конкретним потребам та інтересам.

Для генерації нових зображень за запитом користувача будуть використані сучасні моделі глибокого навчання, такі як GAN, VAE та дифузійні моделі. Вибір конкретної моделі буде залежати від її здатності генерувати зображення у бажаному стилі та якості, а також від обчислювальних ресурсів, необхідних для її роботи. Можливо, буде використано комбінацію різних моделей для забезпечення більшої різноманітності та гнучкості.

У подальших підрозділі буде обґрунтовано вибір технологічного стеку для розробки онлайн-платформи, включаючи фронтенд, бекенд, базу даних та інструменти розробки. Вибір конкретних технологій базується на їх популярності, функціональності, зручності використання, продуктивності та відповідності вимогам проекту.

2.2 Мова розмітки гіпертексту HTML

HTML (HyperText Markup Language) [14] – це стандартна мова розмітки, яка використовується для створення веб-сторінок та веб-додатків. Її історія розпочинається у 1989 році, коли британський фізик Тім Бернерс-Лі, працюючи в CERN (Європейська організація з ядерних досліджень), запропонував концепцію Всесвітньої павутини (World Wide Web) та розробив першу версію HTML (рис. 2.1).

Рисунок 2.1 – Логотип HTML [1]

HTML був задуманий як простий та доступний спосіб обміну інформацією між вченими, які працювали на різних комп'ютерах та операційних системах. Він базувався на SGML (Standard Generalized Markup Language) – складній мові

розмітки, яка використовувалася для створення технічної документації. HTML спростив SGML, визначивши невеликий набір елементів, які дозволяли структурувати текст, додавати посилання на інші документи та вставляти зображення.

Перша версія HTML була дуже простою та мала обмежений набір елементів. Проте, вона швидко стала популярною, і вже у 1993 році була опублікована друга версія HTML (HTML 2.0), яка додала підтримку форм, таблиць та інших елементів.

З того часу HTML продовжував розвиватися, з'являлися нові версії, які додавали нові можливості та покращували сумісність з різними браузерами. У 1997 році була опублікована четверта версія HTML (HTML 4.01), яка стала стандартом для веб-розробки на багато років.

У 2014 році була опублікована п'ята версія HTML (HTML5), яка внесла значні зміни та доповнення. HTML5 додав підтримку аудіо та відео, нові семантичні елементи (такі як <header>, <nav>, <article>, <footer>), API для роботи з геолокацією, локальним сховищем та іншими функціями браузера. HTML5 також покращив підтримку мобільних пристроїв та адаптивного дизайну.

HTML базується на кількох основних концепціях:

1) *елементи:* елементи є основними будівельними блоками HTML-документа. Вони визначають структуру та зміст сторінки. Кожен елемент має початковий та кінцевий тег, між якими знаходиться вміст елемента. Наприклад, елемент <h1> використовується для створення заголовка першого рівня, а елемент <p> - для створення абзацу тексту;

2) *атрибути:* атрибути надають додаткову інформацію про елементи. Вони записуються у початковому тезі елемента у вигляді пар "назва=значення". Наприклад, атрибут src елемента <img> визначає шлях до зображення, яке потрібно відобразити;

3) *теги:* теги використовуються для позначення початку та кінця елементів. Початковий тег записується у вигляді <назва_елемента>, а кінцевий тег - у вигляді

</назва_елемента>. Наприклад, початковий тег елемента <h1> записується як <h1>, а кінцевий тег - як </h1>;

4) *вкладеність:* елементи HTML можуть бути вкладеними один в одного, створюючи ієрархічну структуру документа. Наприклад, елемент <body> може містити елементи <h1>, <p>, <img> та інші;

5) *семантика:* HTML надає семантичні елементи, які описують значення вмісту, а не лише його зовнішній вигляд. Наприклад, елемент <header> використовується для позначення заголовка сторінки, а елемент <nav> - для позначення навігаційного меню. Використання семантичних елементів покращує доступність сторінки для людей з обмеженими можливостями та полегшує роботу пошукових систем.

Структура HTML-документа. Типовий HTML-документ має наступну структуру:

```
<! DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Назва сторінки</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Заголовок</h1>
    <p>Текст абзацу. </p>
    <img src="image.jpg" alt="Опис зображення">
</body>
</html>
```

– <!DOCTYPE html>: Ця декларація вказує, що документ є HTML5;

– <html>: Кореневий елемент документа;

– <head>: Містить метадані про сторінку, такі як кодування символів (<meta charset="UTF-8">), заголовок сторінки (<title>), підключення зовнішніх файлів стилів (<link rel="stylesheet" href="style.css">) та інші;

– <body>: Містить видимий вміст сторінки, такий як текст, зображення, форми, кнопки тощо.

HTML є потужною та гнучкою мовою розмітки, яка дозволяє створювати різноманітні веб-сторінки та веб-додатки. Розуміння основних концепцій HTML та його структури є важливим кроком для опанування веб-розробки.

2.3 Каскадні таблиці стилів CSS

CSS (Cascading Style Sheets) [15] – це мова стилів, призначена для опису зовнішнього вигляду документів, написаних мовою розмітки, такою як HTML. CSS виникла в середині 1990-х років як відповідь на зростаючу складність веб-дизайну та необхідність розділення структури документа (HTML) від його представлення (CSS) (рис. 2.2).

Рисунок 2.2 – Логотип CSS [2]

До появи CSS, стилі для веб-сторінок вбудовувалися безпосередньо в HTML-код за допомогою атрибутів тегів, таких як font, color, align тощо. Це створювало ряд проблем:

1) *дублювання коду:* стилі для кожного елемента потрібно було повторювати на кожній сторінці, що призводило до збільшення розміру файлів та ускладнювало внесення змін;

2) *обмежені можливості:* атрибути тегів надавали обмежений набір стильових властивостей, що ускладнювало створення складних макетів та дизайнів;

3) *погана сумісність:* різні браузери по-різному інтерпретували стилі, що призводило до проблем з відображенням сторінок на різних платформах.

CSS вирішила ці проблеми, дозволивши винести стилі в окремі файли та застосовувати їх до багатьох сторінок одночасно. Це значно спростило розробку та підтримку веб-сайтів, а також розширило можливості дизайну.

Перша версія CSS (CSS1) була опублікована в 1996 році і містила базовий набір властивостей для оформлення тексту, кольорів, фонів, рамок тощо. З того часу CSS продовжувала розвиватися, з'являлися нові версії, які додавали нові можливості та покращували сумісність з браузерами.

CSS2, опублікована в 1998 році, додала підтримку позиціонування елементів, таблиць стилів для друку та інших функцій. CSS3, розробка якої розпочалася в 1999 році, внесла ще більше змін, додавши підтримку анімацій, трансформацій, градієнтів, тіней, гнучких макетів (flexbox) та інших сучасних можливостей.

Основні концепції CSS. CSS базується на кількох основних концепціях:

1) *селектори:* селектори визначають, до яких елементів HTML застосовуються стилі. Вони можуть вибирати елементи за їх типом (h1, p, div), класом (.my-class), ідентифікатором (#my-id), атрибутами ([href]) та іншими критеріями;

2) *властивості та значення:* властивості визначають, які характеристики елемента потрібно змінити (наприклад, color, font-size, background-color). Значення вказують, яке значення має бути встановлене для властивості (наприклад, red, 16px, #f0f0f0);

3) *правила:* правила CSS поєднують селектори, властивості та значення;

4) *каскадування:* CSS використовує каскадний принцип для визначення того, які стилі застосовуються до елемента, якщо для нього визначено кілька

правил. Браузер враховує пріоритет селекторів, порядок їх визначення та інші фактори, щоб вибрати найбільш підходяще правило;

5) *спадкування:* деякі властивості CSS успадковуються від батьківського елемента до дочірніх. Наприклад, якщо для елемента <body> встановлено колір шрифту color: blue, то цей колір буде успадкований усіма елементами всередині <body>, якщо для них не визначено інший колір;

6) *специфічність:* специфічність визначає, наскільки "важливим" є селектор. Чим вища специфічність селектора, тим вищий його пріоритет. Наприклад, селектор за ідентифікатором (#my-id) має вищу специфічність, ніж селектор за класом (.my-class), який, у свою чергу, має вищу специфічність, ніж селектор за типом елемента (div).

```
селектор {
властивість1: значення1;
властивість2: значення2;
/ *... */
}

body {
    font-family: sans-serif;
    line-height: 1.6;
    margin: 0;
}
h1 {
    color: #333;
    font-size: 2.5rem;
}
.container {
    max-width: 960px;
    margin: 0 auto;
    padding: 1rem;
}
```

Цей код CSS визначає стилі для елемента <body> (шрифт, міжрядковий інтервал, відступи), заголовка <h1> (колір, розмір шрифту) та контейнера з класом .container (максимальна ширина, центрування, внутрішні відступи).

CSS є потужною та гнучкою мовою стилів, яка дозволяє створювати привабливі та зручні веб-інтерфейси. Розуміння основних концепцій CSS та його принципів роботи є важливим кроком для опанування веб-розробки.

Звичайно, давайте детально розглянемо JavaScript, його походження та основні концепції.

2.4 Мова програмування JavaScript. Бібліотека Masonry

JavaScript [16] – це високорівнева, інтерпретована мова програмування, яка використовується для створення динамічних та інтерактивних веб-сторінок. Вона була розроблена у 1995 році Бренданом Айхом у Netscape Communications Corporation (пізніше Mozilla Corporation) під назвою Mocha, потім перейменована на LiveScript, а згодом на JavaScript (рис. 2.3).

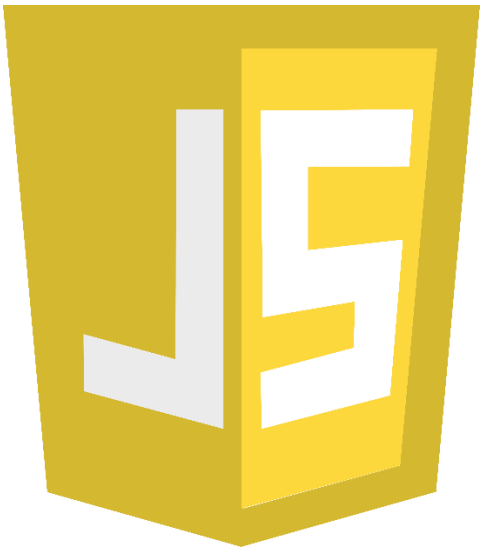

Рисунок 2.3 – Логотип JS [3]

JavaScript була створена для того, щоб додати інтерактивність до статичних HTML-сторінок. На той час веб-сторінки були в основному статичними документами, які могли відображати лише текст та зображення. JavaScript дозволив розробникам додавати до сторінок динамічні елементи, такі як анімації, форми, що реагують на введення користувача, та інші інтерактивні функції.

Спочатку JavaScript використовувався переважно для простих завдань, таких як валідація форм та створення спливаючих вікон. Проте, з розвитком технологій та збільшенням потужності браузерів, JavaScript став повноцінною мовою програмування, яка використовується для створення складних веб-додатків,

односторінкових застосунків (SPA), мобільних додатків (за допомогою фреймворків React Native та Ionic) та навіть серверних застосунків (за допомогою Node.js).

Основні концепції JavaScript. JavaScript базується на кількох основних концепціях:

1) *об'єктно-орієнтоване програмування (ооп):* JavaScript підтримує ООП, хоча і не має класів у традиційному розумінні. Об'єкти в JavaScript є асоціативними масивами, де ключі є іменами властивостей, а значення - значеннями цих властивостей. Об'єкти можуть також мати методи, які є функціями, пов'язаними з об'єктом;

2) *функціональне програмування (фп):* JavaScript підтримує ФП, що дозволяє розглядати функції як значення першого класу, передавати їх як аргументи іншим функціям, повертати їх з функцій та зберігати у змінних. Це дозволяє писати більш декларативний та модульний код;

3) *прототипне успадкування:* JavaScript використовує прототипне успадкування, що відрізняється від класичного успадкування, яке використовується в більшості об'єктно-орієнтованих мов. У прототипному успадкуванні об'єкти успадковують властивості та методи безпосередньо від інших об'єктів (прототипів), а не від класів;

4) *динамічна типізація:* JavaScript є динамічно типізованою мовою, що означає, що тип змінної визначається під час виконання програми, а не під час її написання. Це робить JavaScript більш гнучким, але також може призвести до помилок, якщо тип змінної несподівано змінюється;

5) *асинхронність:* JavaScript підтримує асинхронне програмування, що дозволяє виконувати довготривалі операції (наприклад, запити до сервера) без блокування основного потоку виконання. Це досягається за допомогою механізмів, таких як callback-функції, проміси (promises) та async/await.

Структура JavaScript-коду зазвичай вбудовується в HTML-документ за допомогою тегу <script> зображений на. Він може бути розміщений у розділі

<head> або <body>, або ж підключений з окремого файлу з розширенням .js. Ось наведений приклад:

```
<script>
// JavaScript-код
</script>
```

Код JavaScript виконується після завантаження HTML-документа. Він знаходить кнопку з ідентифікатором myButton та додає до неї обробник події click. Коли користувач натискає на кнопку, з'являється спливаюче вікно з повідомленням "Привіт!". Ось наведений приклад:

```
document . addEventListener ('DOMContentLoaded', function() {
    const button = document.getElementById('myButton');
    button. addEventListener('click', function() {
        alert ('Привіт!');
    });
});
```

JavaScript є потужною та універсальною мовою програмування, яка дозволяє створювати динамічні та інтерактивні веб-додатки. Розуміння основних концепцій JavaScript та його можливостей є ключовим для успішної веб-розробки.

Для полегшення розробки фронтенду та підвищення його ефективності планується використовувати наступну бібліотеку таку, як Masonry:

JavaScript-бібліотека [6] для адаптивних галерей. Masonry – це JavaScript-бібліотека, розроблена Девідом ДеСандро, яка дозволяє створювати динамічні та адаптивні галереї зображень (рис. 2.4). Вона працює за принципом "каскадного" розміщення елементів, тобто розміщує елементи в оптимальному положенні на основі доступного вертикального простору, подібно до того, як муляр укладає камені в стіну. Такий підхід дозволяє створювати галереї, які ефективно використовують простір та виглядають привабливо на екранах різних розмірів.

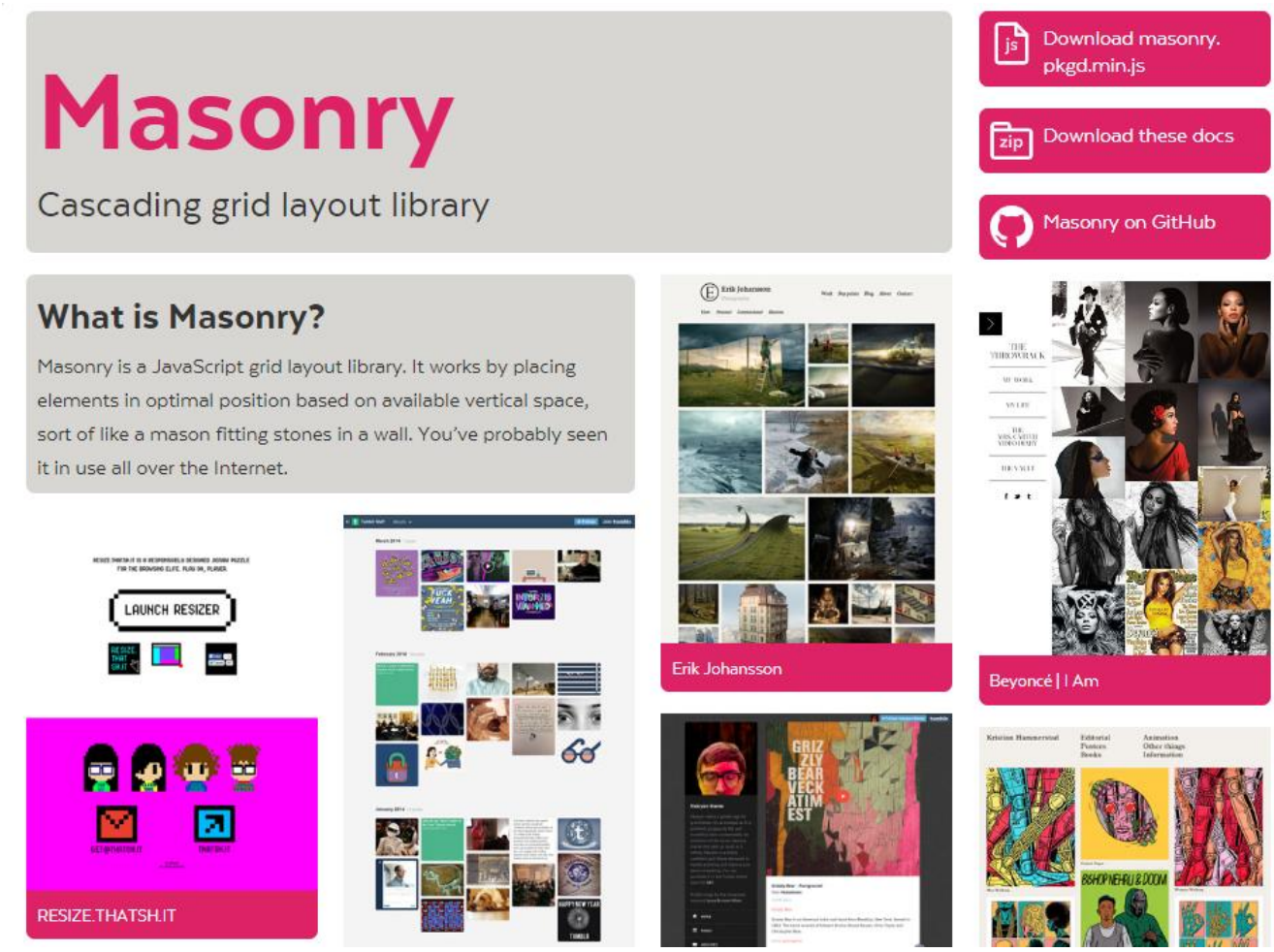

Рисунок 2.4 – Сторінка Masonry

Основні можливості Masonry:

1) *адаптивність:* masonry автоматично адаптує розміщення елементів до ширини екрану, забезпечуючи оптимальний вигляд галереї на різних пристроях (комп'ютерах, планшетах, смартфонах);

2) *динамічність:* masonry дозволяє додавати, видаляти та оновлювати елементи галереї без необхідності перебудовувати всю галерею заново;

3) *простота використання:* masonry має простий та інтуїтивно зрозумілий API, що дозволяє легко інтегрувати її у веб-проекти;

4) *широкі можливості налаштування:* masonry надає безліч опцій для налаштування зовнішнього вигляду та поведінки галереї, таких як ширина колонок, відступи між елементами, порядок сортування тощо;

5) Підтримка різних браузерів: masonry працює у всіх сучасних браузерах, включаючи Chrome, Firefox, Safari та Edge.

Як працює Masonry. Masonry працює в кілька етапів:

1) *ініціалізація:* при ініціалізації Masonry обчислює розміри контейнера галереї та визначає оптимальну кількість колонок на основі ширини екрану та заданих параметрів;

2) *розміщення елементів:* masonry розміщує елементи галереї у колонки, починаючи з найвищої. Кожен елемент розміщується у найнижчому доступному місці у колонці;

3) *оновлення:* при зміні розміру екрану або додаванні/видаленні елементів Masonry перераховує розміщення елементів та оновлює галерею.

Приклад реалізації Masonry:

```
// IHiцianizayiA Masonry
var elem = document.querySelector('.grid');
var msnry = new Masonry( elem, {
// options
    itemSelector: '.grid-item',
    columnWidth: 200
});
/ / Додавання нового елемента
var newItem = document.createElement('div');
    newItem.classList.add('grid-item');
    elem. appendChild(newItem);

/ / Оновлення Masonry
msnry. appended(newItem);
```

Важливість Masonry для платформи рекомендацій зображень є ідеальним інструментом для створення галереї зображень на платформі рекомендацій, оскільки вона дозволяє:

1) *ефективно використовувати простір:* masonry розміщує зображення різних розмірів та пропорцій в оптимальному порядку, уникаючи пустих місць та забезпечуючи привабливий вигляд галереї;

2) *забезпечити адаптивність:* masonry автоматично адаптується до різних розмірів екранів, що дозволяє користувачам комфортно переглядати галерею на будь-якому пристрої;

3) *підвищити інтерактивність:* masonry дозволяє додавати, видаляти та оновлювати зображення без перезавантаження сторінки, що робить галерею більш динамічною та інтерактивною.

Використання Masonry на платформі рекомендацій зображень дозволить створити зручний та привабливий інтерфейс, який сприятиме залученню та утриманню користувачів.

2.5 Середовище виконання Node.js. JavaScript за межами браузера

Node.js – це середовище виконання JavaScript з відкритим вихідним кодом [17], побудоване на рушії V8 JavaScript від Google. Воно дозволяє запускати JavaScript-код поза браузером, на сервері або у будь-якому іншому середовищі. Node.js був створений Райаном Далем у 2009 році та швидко здобув популярність завдяки своїй продуктивності, масштабованості та можливості використовувати JavaScript як універсальну мову програмування для розробки як фронтенду, так і бекенду (рис. 2.5).

До появи Node.js, JavaScript використовувався переважно для створення інтерактивних елементів на веб-сторінках та працював лише у браузері. Node.js розширив можливості JavaScript, дозволивши використовувати його для створення серверних застосунків, мережевих інструментів, скриптів автоматизації, десктопних застосунків (за допомогою Electron) та навіть мікроконтролерів (за допомогою Johnny-Five).

Рисунок 2.5 – Логотип Node.js [4]

Основні концепції Node.js. Node.js базується на кількох основних концепціях:

1) *неблокуюча модель введення-виведення (non-blocking i/o):* Node.js використовує неблокуючу модель введення-виведення, що означає, що операції введення-виведення (наприклад, читання файлу з диска або відправка запиту до бази даних) виконуються асинхронно. Це дозволяє Node.js обробляти велику кількість запитів одночасно без блокування основного потоку виконання, що робить його дуже ефективним для створення високопродуктивних та масштабованих застосунків;

2) *подійно-орієнтована архітектура (event-driven architecture):* Node.js використовує подійно-орієнтовану архітектуру, де події (наприклад, завершення операції введення-виведення або отримання запиту від клієнта) запускають виконання callback-функцій. Це дозволяє писати код, який реагує на події в реальному часі та ефективно використовує ресурси системи;

3) *модульність:* Node.js має вбудовану систему модулів, яка дозволяє розбивати код на окремі файли (модулі) та використовувати їх в інших частинах програми. Це сприяє кращій організації коду, повторному використанню та спрощенню тестування;

4) *npm (node package manager):* npm - це менеджер пакетів для Node.js, який дозволяє встановлювати, оновлювати та видаляти модулі та бібліотеки, необхідні для розробки проекту. npm має величезний реєстр пакетів, що містить тисячі готових рішень для різних завдань, що значно спрощує та прискорює розробку;

Архітектура Node.js складається з кількох основних компонентів:

1) *V8:* V8 – це рушій JavaScript з відкритим вихідним кодом, розроблений Google. Він відповідає за виконання JavaScript-коду, компіляцію його в машинний код та оптимізацію виконання;

2) *libuv:* libuv – це бібліотека з відкритим вихідним кодом, яка забезпечує кросплатформну підтримку асинхронного введення-виведення, обробки подій та інших функцій, необхідних для роботи Node.js;

3) *інші бібліотеки:* Node.js включає в себе інші бібліотеки, такі як http, fs, path, crypto тощо, які надають функціональність для роботи з мережею, файловою системою, криптографією та іншими аспектами розробки застосунків.

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Привіт, світ!');
});
server.listen(3000, () => {
    console.log( 'Сервер запущено на порту 3000 ');
});
```

Цей код створює простий веб-сервер, який прослуховує порт 3000 та відправляє повідомлення "Привіт, світ!" у відповідь на кожен запит.

Важливість Node.js для платформи рекомендацій зображень є ідеальним вибором для розробки бекенду платформи рекомендацій зображень, оскільки він дозволяє:

1) обробляти велику кількість запитів одночасно: Завдяки неблокуючій моделі введення-виведення та подійно-орієнтованій архітектурі, Node.js здатний обробляти велику кількість запитів від користувачів без втрати продуктивності;

2) швидко розробляти та розгортати застосунки: Node.js має просту та зрозумілу структуру, а також велику кількість готових модулів та бібліотек, що дозволяє швидко створювати та розгортати веб-застосунки;

3) використовувати JavaScript на фронтенді та бекенді: Це спрощує розробку та підтримку проекту, оскільки розробникам не потрібно вивчати кілька мов програмування;

4) легко інтегруватися з іншими технологіями: Node.js має велику кількість модулів для роботи з базами даних, хмарними сервісами, API соціальних мереж та іншими технологіями, що дозволяє легко інтегрувати платформу з іншими системами.

Використання Node.js на платформі рекомендацій зображень дозволить створити швидкий, масштабований та надійний бекенд, який зможе обробляти велику кількість запитів та забезпечувати високу якість сервісу.

2.6 Середовище та інструменти розробки Visual Studio Code та npm

Visual Studio Code (VS Code) – це потужний і універсальний редактор коду з відкритим вихідним кодом, розроблений Microsoft (рис. 2.6). Він доступний для Windows, macOS та Linux, що робить його чудовим вибором для розробників, які працюють на різних платформах. VS Code поєднує в собі простоту використання текстового редактора з потужними функціями інтегрованого середовища розробки (IDE), що робить його привабливим як для початківців, так і для досвідчених програмістів [12].

Рисунок 2.6 – Логотип VS Code [5]

Основні можливості VS Code пропонує широкий спектр функцій, які полегшують та прискорюють процес розробки:

1) *підтримка великої кількості мов програмування:* VS Code підтримує величезну кількість мов програмування, включаючи JavaScript, TypeScript, Python,

Java, C++, C#, PHP, Go, Ruby, Rust та багато інших. Завдяки цьому, розробники можуть використовувати один редактор для роботи з різними проектам;

2) *інтелектуальне автодоповнення коду (intellisense):* IntelliSense аналізує код та пропонує варіанти автодоповнення, враховуючи контекст, що значно прискорює написання коду та зменшує кількість помилок;

3) *вбудований дебагер:* VS Code має вбудований дебагер, який дозволяє запускати код покроково, встановлювати точки зупину, переглядати значення змінних та виявляти помилки. Це значно спрощує процес відлагодження коду;

4) *інтеграція з git:* VS Code має вбудовану підтримку Git, що дозволяє керувати версіями коду, створювати гілки, виконувати коміти, відправляти зміни на віддалений репозиторій та вирішувати конфлікти безпосередньо з редактора;

5) *термінал:* VS Code має вбудований термінал, який дозволяє виконувати команди операційної системи та запускати скрипти безпосередньо з редактора;

6) *розширення:* VS Code має величезну бібліотеку розширень, які додають підтримку нових мов, фреймворків, інструментів та інших функцій. Розширення дозволяють налаштувати VS Code під свої потреби та зробити його ще більш потужним та універсальним;

7) *налаштування:* VS Code має гнучку систему налаштувань, яка дозволяє змінювати тему оформлення, шрифти, комбінації клавіш та інші параметри;

8) *кросплатформеність:* VS Code доступний для Windows, macOS та Linux, що дозволяє використовувати його на різних платформах без необхідності перевчатися.

Важливість VS Code для розробки платформи є ідеальним інструментом для розробки онлайн-платформи рекомендацій зображень, оскільки він:

1) *підтримує всі необхідні технології:* VS Code має вбудовану підтримку HTML, CSS та JavaScript, а також підтримує розширення для роботи з Node.js, React, Masonry та іншими технологіями, які будуть використані у проекті;

2) *полегшує написання та відлагодження коду:* равдяки IntelliSense, вбудованому дебагеру та іншим інструментам, VS Code значно спрощує написання та відлагодження коду, що прискорює процес розробки;

3) *спрощує роботу з git:* вбудована підтримка git дозволяє легко керувати версіями коду та співпрацювати з іншими розробниками;

4) *надає гнучкість та можливості налаштування:* велика кількість розширень та гнучка система налаштувань дозволяють налаштувати VS Code під свої потреби та зробити його ще більш потужним та зручним інструментом для розробки.

Використання VS Code дозволить підвищити продуктивність розробки, зменшити кількість помилок та створити більш якісний та надійний продукт.

2.6.2 Менеджер пакетів NPM

Npm (Node Package Manager) – це менеджер пакетів для Node.js, який є невід'ємною частиною екосистеми JavaScript [19]. Він служить для встановлення, оновлення, видалення та управління залежностями між різними модулями та бібліотеками JavaScript, які використовуються у веб-розробці.

npm складається з двох основних компонентів:

1) *клієнт командного рядка (cli):* це інструмент, який дозволяє взаємодіяти з npm з терміналу або командного рядка. За допомогою команд CLI можна встановлювати пакети, керувати залежностями, публікувати власні пакети та виконувати інші операції;

2) *реєстр npm:* це онлайн-репозиторій, який містить тисячі пакетів JavaScript з відкритим вихідним кодом. Розробники можуть публікувати свої пакети у реєстрі, а інші розробники можуть легко знаходити та встановлювати їх у свої проекти.

Основні можливості npm надає широкий спектр можливостей для роботи з пакетами JavaScript:

1) *встановлення пакетів:* npm дозволяє легко встановлювати пакети з реєстру npm за допомогою команди npm install. Пакети встановлюються у папку node_modules проекту, а їх залежності автоматично вирішуються та встановлюються разом з ними;

2) *оновлення пакетів:* npm дозволяє оновлювати пакети до останніх версій за допомогою команди npm update. Це важливо для отримання нових функцій, виправлення помилок та забезпечення безпеки проекту;

3) *видалення пакетів:* npm дозволяє видаляти пакети, які більше не потрібні, за допомогою команди npm uninstall;

4) *управління залежностями:* npm автоматично відстежує залежності між пакетами та забезпечує їх коректну установку та оновлення. Це дозволяє уникнути конфліктів між пакетами та забезпечити стабільну роботу проекту;

5) *створення власних пакетів:* npm дозволяє розробникам створювати власні пакети та публікувати їх у реєстрі npm, щоб інші розробники могли їх використовувати;

6) *виконання скриптів:* npm дозволяє визначати скрипти у файлі package.json проекту та запускати їх за допомогою команди npm run. Це зручно для автоматизації завдань, таких як збірка проекту, запуск тестів, розгортання на сервері тощо.

Npm працює з файлом package.json, який є маніфестом проекту. У цьому файлі зберігається інформація про проект, його залежності та скрипти. Коли розробник запускає команду npm install, npm читає файл package.json, знаходить перелік залежностей та завантажує їх з реєстру npm. Залежності встановлюються у папку node_modules, а їх версії фіксуються у файлі package-lock.json, що дозволяє відтворювати однакове середовище розробки на різних машинах.

Важливість npm для платформи рекомендацій зображень є невід'ємною частиною розробки платформи рекомендацій зображень, оскільки він дозволяє:

1) *використовувати готові рішення:* реєстр npm містить тисячі пакетів, які можна використовувати для розробки різних компонентів платформи, таких як веб-

сервер, база даних, інструменти для роботи з зображеннями, бібліотеки для машинного навчання тощо. Це дозволяє значно прискорити розробку та зосередитися на основному функціоналі платформи;

2) *спростити управління залежностями:* npm автоматично вирішує залежності між пакетами та забезпечує їх коректну установку та оновлення, що полегшує підтримку проекту та запобігає виникненню конфліктів;

3) *автоматизувати завдання:* npm дозволяє визначати скрипти для виконання різних завдань, таких як збірка проекту, запуск тестів, розгортання на сервері тощо. Це спрощує та прискорює процес розробки та розгортання.

Використання npm дозволяє створити більш ефективний та продуктивний процес розробки, що в кінцевому підсумку призводить до створення більш якісного та надійного продукту.

Висновки до розділу 2

У цьому розділі було проведено огляд та аналіз технологій, необхідних для створення онлайн-платформи рекомендацій генеративних зображень. Було обґрунтовано вибір таких ключових технологій, як HTML, CSS, JS, Node.js та бібліотека Masonry.

HTML буде використано для структурування та представлення контенту платформи. HTML надає широкий спектр семантичних елементів, таких як <header>, <nav>, <article>, <section> та <footer>, що дозволяє створювати логічну та зрозумілу структуру сторінки, покращує її доступність та сприяє кращій індексації пошуковими системами. Використання HTML5 забезпечить сучасний підхід до веб-розробки, включаючи підтримку мультимедіа, семантичної розмітки та нових API.

CSS буде застосовано для створення візуально привабливого та адаптивного дизайну платформи. CSS дозволяє точно контролювати зовнішній вигляд кожного елемента сторінки, включаючи кольори, шрифти, розміри, розташування та інші

візуальні властивості. Застосування сучасних можливостей CSS3, таких як flexbox та grid layout, забезпечить створення гнучких та адаптивних макетів, які будуть коректно відображатися на різних пристроях та розмірах екранів. Використання препроцесора SASS дозволить спростити та прискорити написання CSS-коду, забезпечуючи можливість використання змінних, вкладених селекторів, міксинів та інших корисних функцій.

JS відіграватиме ключову роль у забезпеченні динамічності та інтерактивності платформи. JS дозволить реалізувати складну логіку взаємодії з користувачем, обробку подій, асинхронні запити до сервера, анімації та інші динамічні ефекти. Використання сучасних можливостей JS, таких як модулі, класи, стрілочні функції та деструктуризація, дозволить писати більш читабельний, структурований та підтримуваний код. Транскомпілятор Babel забезпечить сумісність коду з різними браузерами, дозволяючи використовувати новітні можливості JS навіть у старіших версіях браузерів.

Node.js буде використано для створення серверної частини платформи. Node.js дозволить обробляти запити від клієнтів, взаємодіяти з базою даних, виконувати обчислення та генерувати динамічний контент. Завдяки своїй неблокуючій архітектурі та використанню подійно-орієнтованого підходу, Node.js забезпечує високу продуктивність та масштабованість, що особливо важливо для платформи з великою кількістю користувачів та запитів.

Бібліотека Masonry буде застосована для створення адаптивної галереї зображень. Masonry дозволяє розміщувати зображення різних розмірів та пропорцій в оптимальному порядку, уникаючи пустих місць та забезпечуючи привабливий вигляд галереї. Вона автоматично адаптується до різних розмірів екранів, забезпечуючи зручний перегляд галереї на різних пристроях.

Для розробки проекту буде використано редактор коду VS Code, який надає широкий спектр інструментів для написання, відлагодження та тестування коду, а також інтегрується з Git для керування версіями. Менеджер пакетів npm дозволить

ефективно керувати залежностями проекту, встановлювати та оновлювати необхідні бібліотеки та фреймворки.

Таким чином, обраний технологічний стек забезпечить необхідну функціональність, продуктивність та масштабованість онлайн-платформи рекомендацій генеративних зображень. Застосування сучасних технологій та інструментів дозволить створити зручний та ефективний сервіс, який відповідатиме потребам користувачів та сприятиме розвитку генеративного мистецтва.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ОНЛАЙН-ПЛАТФОРМИ

3.1 Опис вхідних даних та вибір технологій розробки

В якості вхідних даних для онлайн-платформи використовуються:

1) *колекція згенерованих зображень:* зображення, створені за допомогою моделі Stability AI SDXL на платформі Replicate [11];

2) *промти (підказки) для генерації зображень:* текстові описи, які використовувалися для створення зображень;

3) *метадані зображень:* інформація про зображення, така як автор, дата створення, модель генерації, параметри генерації тощо;

4) *дані користувачів:* інформація про взаємодію користувачів з платформою, така як перегляди, лайки, коментарі, збереження в обране тощо.

Обґрунтування вибору технологій. Для реалізації проекту було обрано наступний стек технологій:

1) Node.js: використовується для розробки серверної частини платформи. Node.js дозволяє створювати швидкі та масштабовані серверні застосунки за допомогою JavaScript, що спрощує розробку та підтримку проекту, оскільки використовується одна мова програмування для фронтенду та бекенду;

2) Express.js: фреймворк для Node.js, який спрощує створення веб-серверів та API. Express.js надає зручні інструменти для обробки маршрутів, запитів та відповідей, а також для роботи з шаблонами та статичними файлами [18];

3) EJS (Embedded JavaScript): шаблонізатор, який дозволяє вбудовувати JavaScript-код в HTML-сторінки. EJS використовується для створення динамічних сторінок, які можуть змінювати свій вміст залежно від даних, отриманих з сервера;

4) HTML5, CSS3 та JavaScript: стандартні технології для розробки фронтенду. HTML5 використовується для структурування контенту, CSS3 – для оформлення зовнішнього вигляду, а JavaScript – для додавання інтерактивності та динамічності;

5) Masonry: JavaScript-бібліотека для створення адаптивних галерей зображень. Masonry дозволяє ефективно використовувати простір та створювати привабливі галереї, які добре виглядають на різних пристроях;

6) Replicate: платформа для запуску моделей машинного навчання, яка дозволяє легко інтегрувати генерацію зображень у веб-застосунок;

7) Stability AI SDXL: модель глибокого навчання, яка використовується для генерації зображень високої якості на основі текстових підказок;

8) Visual Studio Code: зручний та потужний редактор коду з великою кількістю розширень та інструментів для веб-розробки;

9) npm (Node Package Manager): менеджер пакетів для Node.js, який дозволяє легко встановлювати, оновлювати та керувати залежностями проекту [19].

Цей набір технологій було обрано завдяки їх популярності, широким можливостям, активній спільноті розробників та наявності великої кількості готових рішень та бібліотек. Він дозволяє створити сучасну, ефективну та масштабовану онлайн-платформу, яка відповідає всім поставленим вимогам.

3.2 Проектування структури та функціоналу платформи

3.2.1 Головна сторінка

Головна сторінка є центральним елементом платформи, який надає користувачам доступ до основного функціоналу та контенту. Її дизайн та функціональність спрямовані на забезпечення зручної навігації, швидкого пошуку та персоналізованих рекомендацій зображень.

Основні елементи головної сторінки:

1) *галерея зображень:* відображає добірку згенерованих зображень у вигляді адаптивної сітки, використовуючи бібліотеку Masonry. Сітка автоматично підлаштовується під розмір екрану, забезпечуючи оптимальне відображення на

різних пристроях. Кожне зображення в галереї має підпис з промтом (підказкою), який використовувався для його створення;

2) *форма завантаження зображень:* дозволяє користувачам завантажувати власні згенеровані зображення на платформу. Форма містить поля для вибору файлу зображення, введення промту та вибору типу зображення (наприклад, "природа", "аніме" тощо);

3) *пошуковий рядок:* дозволяє користувачам шукати зображення за ключовими словами або фразами. Пошук здійснюється за промтами та іншими метаданими зображень;

4) *кнопки фільтрації:* дозволяють користувачам фільтрувати зображення за різними категоріями, такими як "природа", "аніме", "місто" тощо. Фільтрація здійснюється за типом зображення, який вказується користувачем під час завантаження;

5) *навігаційне меню:* містить посилання на інші сторінки платформи, такі як сторінка генерації зображень та "Обране".

Функціональність головної сторінки:

1) *динамічне завантаження зображень:* галерея зображень на головній сторінці формується динамічно, підвантажуючи нові зображення при прокручуванні сторінки вниз. Це дозволяє уникнути довгого очікування завантаження всіх зображень одразу та забезпечує плавний перегляд;

2) *пошук та фільтрація:* користувачі можуть використовувати пошуковий рядок та кнопки фільтрації для швидкого пошуку потрібних зображень. Результати пошуку та фільтрації відображаються в галереї зображень в режимі реального часу;

3) *перегляд деталей зображення:* при натисканні на зображення в галереї відкривається модальне вікно, яке містить збільшене зображення, промт, інформацію про автора та інші деталі. Користувачі можуть скопіювати промт, щоб використовувати його для генерації власних зображень;

4) *додавання в обране:* користувачі можуть додавати зображення в обране, натиснувши на кнопку "зірочка". Обрані зображення зберігаються у профілі користувача та доступні для перегляду на сторінці "Обране".

Основна структура та функціональність головної сторінки реалізовані у файлі layout.ejs (Додаток А). Наприклад, код для відображення галереї зображень виглядає так:

```
<div class="gallery" id="gallery">
    <div class="gallery-sizer"></div>
    <% images.forEach(item => { %>
        <div class="gallery-item" data-prompt="<%= item.prompt %>" data-
type="<%= item.type %>">
            <img src="/images/<%= item.image %>" alt="Image">
            </div>
    <% }) %>
</div>
```

Для реалізації головної сторінки використовуються такі технології:

1) HTML5 та CSS3: для створення структури та стилізації сторінки;

2) JavaScript (з використанням бібліотеки Masonry): для створення адаптивної галереї зображень, обробки подій користувача (кліки, прокручування), відправки запитів на сервер та динамічного оновлення контенту сторінки;

3) Node.js та Express.js: для розробки серверної частини, яка обробляє запити від клієнта, взаємодіє з базою даних та файловою системою, формує дані для галереї зображень та надає API для пошуку та фільтрації;

4) EJS: для створення шаблонів сторінок, що дозволяє розділити логіку та представлення даних.

JavaScript-код для реалізації динамічного завантаження зображень та обробки подій знаходиться у файлі main.js (Додаток А). Фільтрація та пошук реалізуються за допомогою наступного коду:

```
searchForm.addEventListener('submit', function(e) {
  e.preventDefault();
  const searchTerm = searchInput.value.toLowerCase();
  galleryItems.forEach(item => {
    const prompt = item.getAttribute('data-prompt').toLowerCase();
    if (prompt.includes(searchTerm)) {
      item.style.display = 'block';
    } else {
      item.style.display = 'none';
    }
  });
  msnry.layout();
});

filterButtons.forEach(button => {
  button.addEventListener('click', function() {
    const filter = this.getAttribute('data-filter');
    galleryItems.forEach(item => {

      if (filter === 'all' || item.getAttribute('data-type') === filter) {
        item.style.display = 'block';
      } else {
        item.style.display = 'none';
      }
    });
    msnry.layout();
  });
});
```

Головна сторінка (рис. 3.1) спроектована з урахуванням принципів зручності використання та візуальної привабливості. Вона має простий та інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам швидко знаходити потрібний контент та виконувати основні дії. Адаптивний дизайн забезпечує коректне відображення сторінки на різних пристроях.

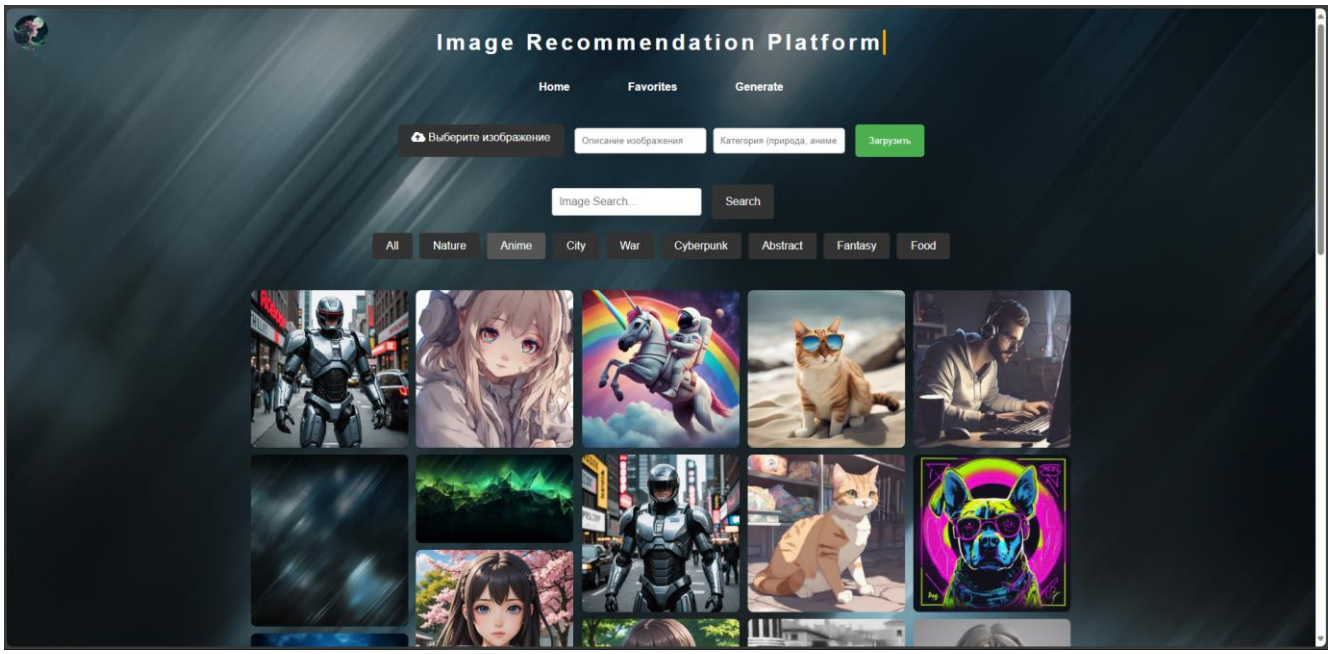

Рисунок 3.1 – Макет дизайну головної сторінки

Можливі покращення:

1) *система рекомендацій:* як ти і зазначив, можна додати систему рекомендацій, яка пропонуватиме користувачам зображення на основі їх попередніх дій та вподобань;

2) *соціальні функції:* можна додати можливість коментувати та оцінювати зображення, створювати власні колекції та підписуватися на інших користувачів;

3) *розширений пошук:* можна додати можливість пошуку за кольором, стилем, настроєм та іншими параметрами;

4) *інтеграція з іншими сервісами:* можна додати можливість ділитися зображеннями в соціальних мережах або використовувати їх в інших проектах.

3.2.2 Сторінка генерації зображень

На сторінці генерації зображень користувачеві надається можливість створити унікальне зображення на основі текстового опису (Додаток В). Ця

функціональність реалізована за допомогою інтеграції з платформою Replicate, яка надає доступ до моделі Stability AI SDXL [11].

Інтерфейс сторінки генерації зображень (рис. 3.2) включає наступні елементи:

1) *поле введення промту:* користувач вводить текстовий опис бажаного зображення. Промт може бути довільним і містити деталі щодо об'єктів, стилю, кольору, композиції тощо;

2) *вибір стилю:* користувач може обрати один з попередньо визначених стилів, таких як "3D Model", "Anime", "Cinematic", "Fantasy Art" тощо. Це дозволяє налаштувати візуальний стиль згенерованого зображення;

3) *кнопка "згенерувати":* після введення промту та вибору стилю користувач натискає цю кнопку, щоб розпочати процес генерації;

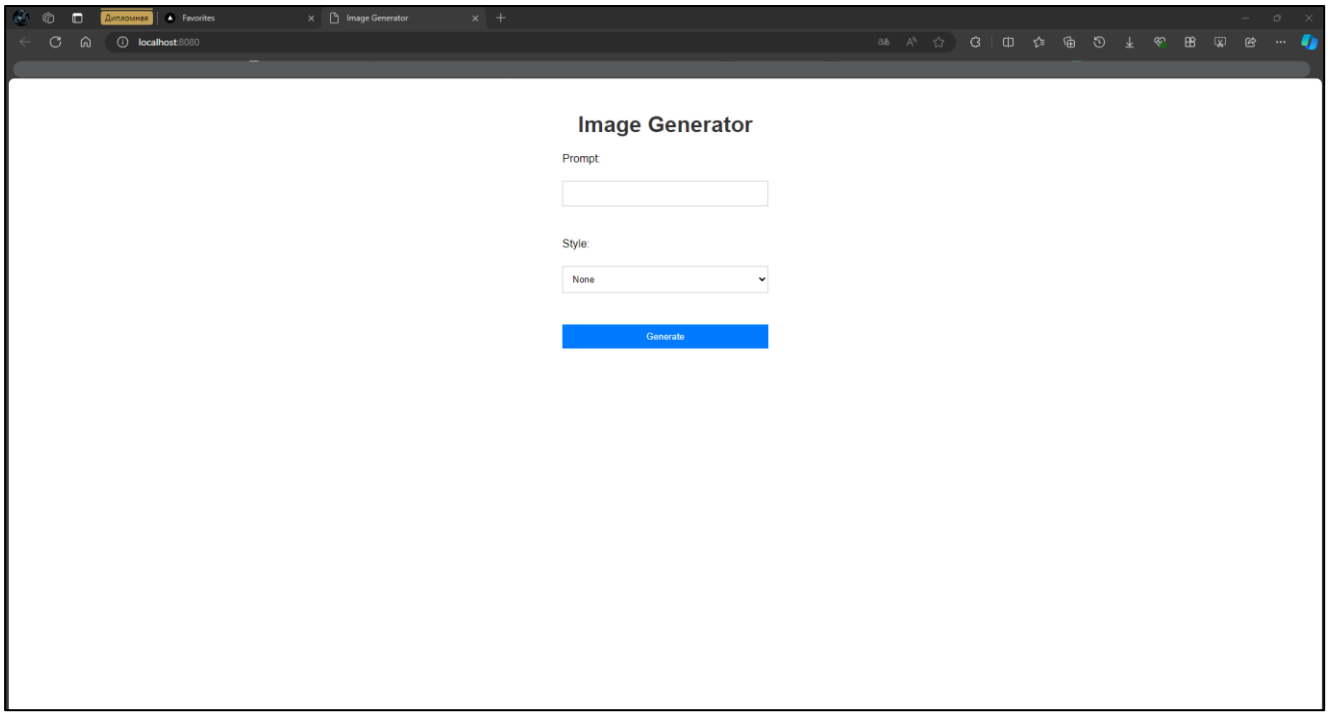

Рисунок 3.2 – Скріншот сторінки генерації зображень

Процес генерації:

1) *обробка запиту:* після натискання кнопки "Згенерувати" фронтенд відправляє запит на сервер, передаючи промт та обраний стиль;

2) *генерація зображення:* сервер використовує API Replicate для відправки промту та стилю до моделі Stability AI SDXL. Модель генерує зображення на основі отриманих даних;

3) *збереження та відображення:* згенероване зображення зберігається на сервері та повертається користувачеві у вигляді URL-адреси. Фронтенд відображає зображення на сторінці.

Для реалізації сторінки генерації зображень використовуються такі технології:

1) HTML5 та CSS3: для створення структури та стилізації сторінки;

2) JavaScript: для обробки подій користувача, відправки запитів на сервер та відображення згенерованого зображення;

3) Node.js та Express.js: для розробки серверної частини, яка обробляє запити від клієнта, взаємодіє з API Replicate та зберігає зображення;

4) Replicate API: для інтеграції з платформою Replicate та використання моделі Stability AI SDXL.

Використання Replicate для генерації зображень має ряд переваг:

1) *простота інтеграції:* Replicate надає зручний API, який легко інтегрується з Node.js та іншими технологіями;

2) *широкий вибір моделей:* Replicate пропонує доступ до багатьох генеративних моделей, включаючи Stability AI SDXL, що дозволяє вибрати оптимальну модель для конкретних завдань;

3) *масштабованість:* Replicate забезпечує масштабованість, що дозволяє обробляти велику кількість запитів на генерацію зображень;

4) *відсутність необхідності керувати інфраструктурою:* Replicate бере на себе всі завдання з розгортання та підтримки моделей, що дозволяє розробникам зосередитися на створенні застосунків.

Інтеграція з платформою Replicate спрощує процес генерації зображень та дозволяє користувачам створювати унікальний візуальний контент без необхідності мати глибокі знання в області машинного навчання.

3.2.3 Сторінка обраного

Сторінка "Обране" призначена для відображення та керування зображеннями, які користувач додав до своєї колекції. Вона забезпечує зручний доступ до збережених зображень та дозволяє користувачеві легко видаляти їх з колекції.

Функціональність сторінки "Обране":

1) *відображення обраних зображень:* на сторінці відображаються всі зображення, які користувач позначив як обрані. Зображення представлені у вигляді мініатюр в адаптивній сітці, подібно до галереї на головній сторінці. Код, що відповідає за відображення обраних зображень, знаходиться у файлі favorites.ejs (Додаток Б);

```
<% favorites.forEach(image => { %>
  <div class="gallery-item">
      <img src="/images/<%= image %>" alt="Image">
      <div class="overlay">
          <button class="remove-favorite-button" data-image="<%= image
%>">&#9733; </button>
      </div>
  </div>
<% }) %>
```

2) *видалення з обраного:* користувач може видалити зображення з обраного, натиснувши на кнопку "Видалити з обраного" під зображенням. Після видалення зображення зникає зі сторінки "Обране" та більше не відображається в колекції обраного користувача. Ця функціональність реалізована за допомогою JavaScript-коду у файлі main.js (Додаток Г) та оброблюється на сервері у файлі main.js;

```
document.querySelectorAll('.remove-favorite-button').forEach(button => {
  button.addEventListener('click', function() {
      // ... (код для видалення зображення)
  });
});
```

3) *перегляд зображення у повному розмірі:* при кліку на мініатюру зображення воно відкривається у модальному вікні у повному розмірі. Цей функціонал реалізований за допомогою JavaScript-коду у файлі main.js (Додаток Г).

Для реалізації сторінки "Обране" використовуються такі технології:

1) HTML (favorites.ejs): структура сторінки, включаючи заголовок, навігаційне меню та контейнер для галереї обраних зображень;

2) CSS (styles.css): Стилізація сторінки, включаючи оформлення галереї та кнопок;

3) JavaScript (main.js): обробка подій користувача (кліки на кнопки видалення та перегляду зображень), відправка запитів на сервер для видалення з обраного та динамічне оновлення вмісту сторінки. Також використовується бібліотека Masonry для створення адаптивної галереї;

4) Node.js та Express.js (app.js): обробка запитів на додавання та видалення зображень з обраного, читання та запис даних у файл favorites.json, рендеринг сторінки favorites.ejs з передачею даних про обрані зображення;

5) JSON (favorites.json): зберігання списку обраних зображень у форматі JSON.

Сторінка "Обране" взаємодіє з такими компонентами платформи:

1) *головна сторінка та галерея зображень (layout.ejs, main.js):* Кнопки "Додати в обране" на цих сторінках додають/видаляють зображення зі списку обраного та оновлюють візуальний стан кнопок;

2) *сервер (app.js):* Обробляє запити на додавання/видалення зображень з обраного та надає дані для відображення на сторінці "Обране";

3) *файл favorites.json:* Зберігає список обраних зображень користувача.

Дизайн сторінки "Обране" (рис. 3.4) схожий на дизайн головної сторінки та галереї зображень, забезпечуючи єдність стилю та зручність використання. Основна відмінність полягає у наявності кнопки "Видалити з обраного" під кожним зображенням.

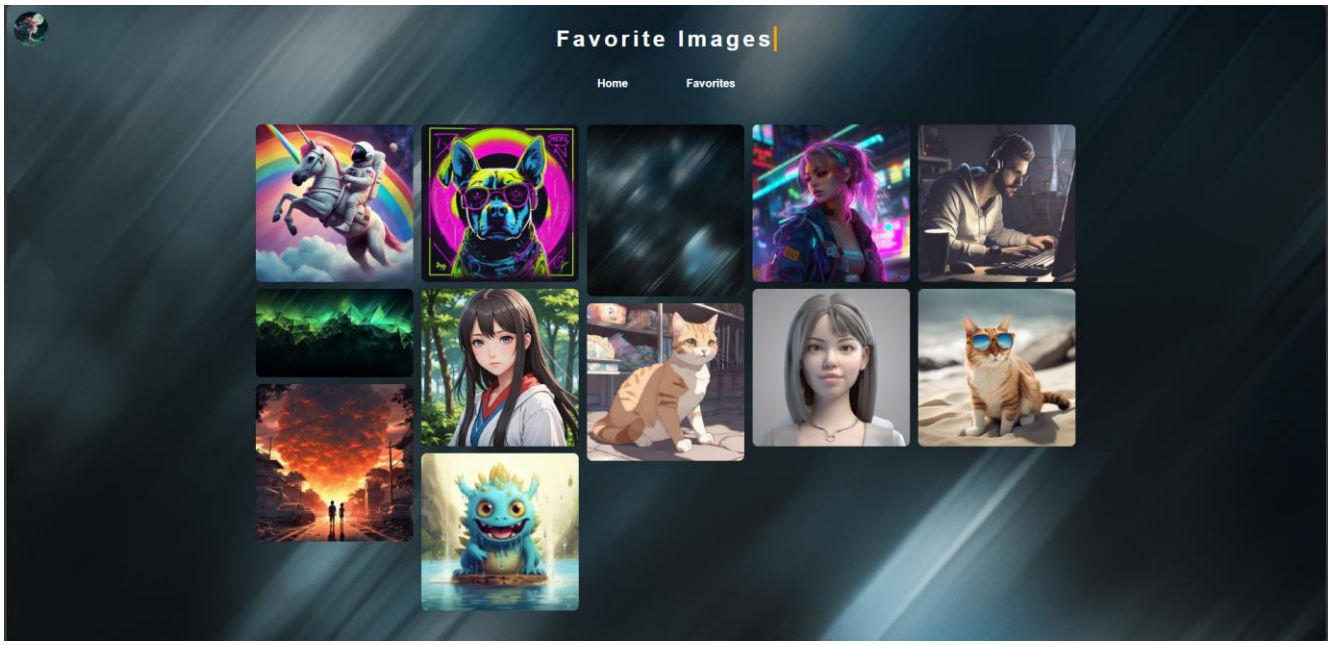

Рисунок 3.3 – Скріншот сторінки "Обране"

3.3 Реалізація основних компонентів платформи

3.3.1 Система рейтингу зображень

Система рейтингу зображень дозволяє користувачам оцінювати зображення за допомогою лайків. Це допомагає визначити популярність зображень та сприяє розвитку спільноти, оскільки користувачі можуть бачити, які зображення подобаються іншим.

Функціональність системи рейтингу:

1) *відображення кількості лайків:* під кожним зображенням відображається кількість лайків, які воно отримало;

2) *можливість поставити/прибрати лайк:* користувачі можуть поставити лайк зображенню, натиснувши на кнопку "сердечко" під ним. Якщо користувач вже поставив лайк, повторне натискання на кнопку прибере його;

3) *збереження лайків:* інформація про лайки зберігається на сервері, що дозволяє відстежувати популярність зображень та відображати актуальну кількість лайків для кожного зображення.

Для реалізації системи рейтингу використовуються такі технології:

1) *HTML5 та CSS3:* для створення кнопки "сердечко" та відображення кількості лайків;

2) *JavaScript:* для обробки кліків на кнопку "сердечко", відправки запитів на сервер для оновлення кількості лайків та динамічного оновлення відображення кількості лайків на сторінці;

3) *Node.js та Express.js:* для розробки серверної частини, яка обробляє запити від клієнта.

Приклад коду для обробки лайків зображень можна знайти у файлі main.js:

```
document.querySelectorAll('.like-button').forEach(button => {
    button.addEventListener('click', function() {
      const image = this.getAttribute('data-image');
      const liked = this.classList.toggle('liked');
      fetch('/favorite', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ image, action: liked ? 'add' : 'remove' })
      });
    });
  });
```

Потенційні покращення:

1) *більше варіантів оцінок:* можна додати можливість ставити не лише лайки, але й дизлайки або оцінки за шкалою (наприклад, від 1 до 5);

2) *коментарі:* можна додати можливість користувачам залишати коментарі до зображень, що сприятиме більшій взаємодії між користувачами та авторами зображень;

3) *персоналізований рейтинг:* можна розробити алгоритм, який враховуватиме індивідуальні вподобання користувача при розрахунку рейтингу зображень, що зробить рейтинг більш релевантним для кожного користувача.

3.3.2 Пошук та фільтрація зображень

Онлайн-платформа надає користувачам можливість швидко та зручно знаходити потрібні зображення за допомогою функцій пошуку та фільтрації. Ці функції реалізовані на фронтенді з використанням JavaScript, HTML5 та CSS3.

Пошук зображень (рис. 3.4) здійснюється за ключовими словами або фразами, які користувач вводить у пошуковий рядок. Пошуковий запит обробляється безпосередньо у браузері користувача за допомогою JavaScript.

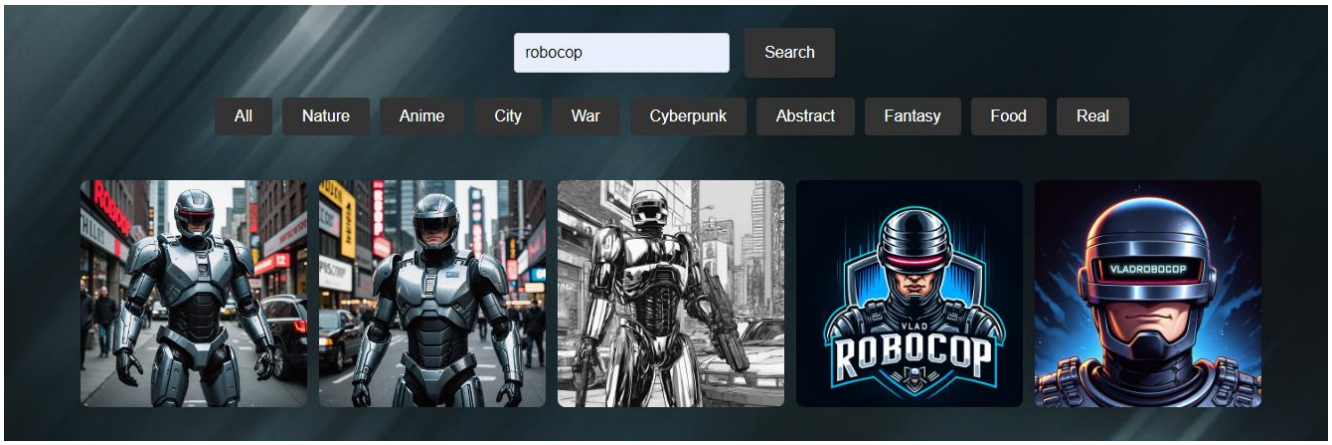

Рисунок 3.4 – Скріншот прикладу пошуку за запитом "robocop"

Алгоритм пошуку:

1) *отримання пошукового запиту*: javascript-код отримує пошуковий запит з текстового поля вводу (searchInput в layout.ejs);

2) *обробка запиту:* Запит перетворюється у нижній регістр для забезпечення регістронезалежного пошуку;

3) *пошук у галереї:* код перебирає всі елементи галереї зображень та перевіряє, чи містить текст промту (підказки) зображення пошуковий запит (значення атрибута data-prompt);

4) *відображення або приховування зображень:* якщо промт зображення містить пошуковий запит, зображення залишається видимим. В іншому випадку, зображення приховується.

Фільтрація зображень (рис. 3.5) дозволяє користувачам переглядати зображення певних категорій. Користувач може вибрати одну або декілька категорій за допомогою кнопок фільтрації (filterButtons в layout.ejs).

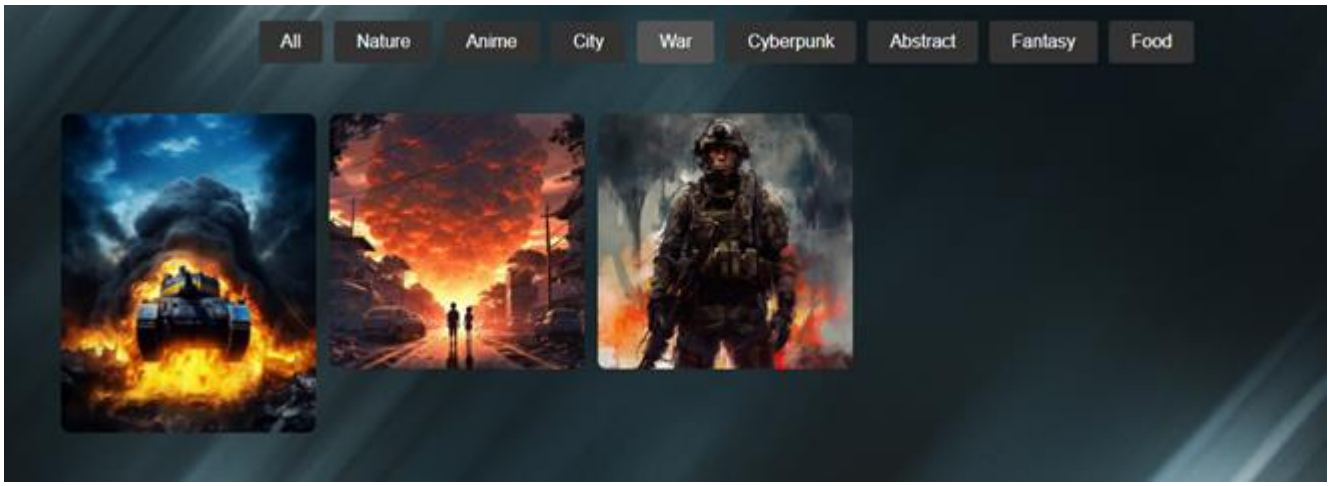

Рисунок 3.5 – Скріншот прикладу фільтрації за категорією "War"

Алгоритм фільтрації:

1) *отримання вибраних категорій:* JavaScript-код отримує інформацію про вибрані категорії з кнопок фільтрації (значення атрибута data-filter);

2) *фільтрація в галереї:* код перебирає всі елементи галереї та перевіряє, чи належить зображення до хоча б однієї з вибраних категорій (значення атрибута data-type);

3) *відображення або приховування зображень:* якщо зображення належить до вибраної категорії або обрано фільтр "Усі" (all), воно залишається видимим. В іншому випадку, зображення приховується.

Технології.

Для реалізації пошуку та фільтрації використовуються такі технології:

1) *HTML5:* для створення структури пошукового рядка та кнопок фільтрації;

2) *CSS3:* для візуального оформлення елементів пошуку та фільтрації;

3) *JavaScript*: для обробки подій користувача (введення пошукового запиту, кліки на кнопки фільтрації), пошуку та фільтрації зображень у галереї.

Дані для пошуку та фільтрації зберігаються у файлі prompts.json. Кожен об'єкт у цьому файлі містить інформацію про одне зображення:

– image: назва файлу зображення;

– prompt: промт (підказка), який використовувався для генерації зображення;

– type: тип зображення (наприклад, "nature", "anime", "city").

Під час пошуку, JavaScript-код перевіряє, чи міститься пошуковий запит у полі prompt кожного об'єкта. Під час фільтрації, код перевіряє, чи співпадає значення поля type з вибраною категорією.

Взаємодія з іншими компонентами:

1) *галерея зображень:* пошук та фільтрація безпосередньо впливають на відображення зображень у галереї;

2) *головна сторінка:* елементи пошуку та фільтрації розташовані на головній сторінці.

Можливі покращення:

1) *пошук на сервері:* для великих колекцій зображень можна перенести логіку пошуку на сервер, щоб зменшити навантаження на браузер користувача;

2) *розширені можливості фільтрації:* можна додати фільтрацію за іншими критеріями, такими як автор, дата створення, стиль тощо.

3.3.3 Завантаження та відображення зображень

У цьому підрозділі описується функціональність завантаження та відображення зображень на платформі.

Користувачі можуть завантажувати власні зображення на платформу через форму завантаження, розташовану на головній сторінці. При завантаженні користувач вказує промт (підказку), який використовувався для створення зображення, та тип зображення (наприклад, "природа", "аніме" тощо). Ця

інформація зберігається разом із зображенням для подальшого використання у пошуку та фільтрації (рис. 3.6).

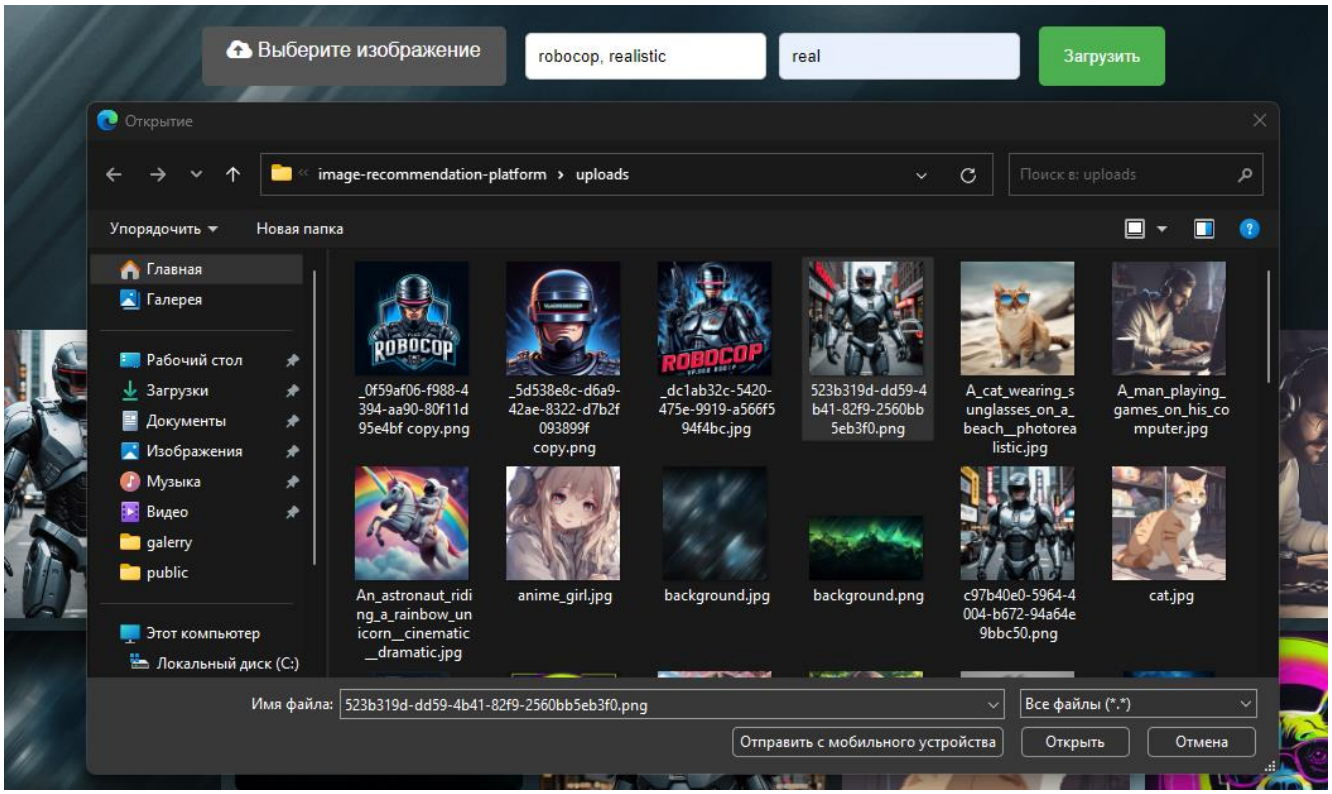

Рисунок 3.6 – Скріншот прикладу ручного завантаження картинки в галерею

Серверна частина обробки завантаження зображень реалізована у файлі app.js (Додаток Д):

```
app.post('/upload', (req, res) => {
  // ... (код для обробки завантаження зображення)
});
```

Код фронтенду для відображення зображень у галереї та модальному вікні знаходиться у файлі layout.ejs та main.js (Додаток А, Г).

Процес завантаження:

1) *вибір файлу:* користувач вибирає файл зображення на своєму пристрої;

2) *введення промту та типу:* користувач вводить текстовий опис зображення (промт) та вибирає тип зображення з випадаючого списку;

3) *відправка даних на сервер:* після натискання кнопки "Завантажити" дані форми (файл зображення, промт та тип) відправляються на сервер;

4) *збереження зображення:* сервер зберігає файл зображення у папці uploads та додає інформацію про зображення (назву файлу, промт, тип) до файлу prompts.json;

5) *оновлення галереї:* після успішного завантаження сервер відправляє клієнту повідомлення про успіх, і галерея зображень на головній сторінці оновлюється, відображаючи нове зображення.

Відображення зображень.

Зображення на платформі відображаються у двох основних місцях:

1) *галерея зображень:* зображення відображаються у вигляді мініатюр, організованих в адаптивну сітку за допомогою бібліотеки Masonry. При наведенні курсору на мініатюру з'являється підпис з промтом. При кліку на мініатюру зображення відкривається у модальному вікні у повному розмірі;

2) *модальне вікно:* у модальному вікні відображається повний розмір зображення, промт, інформація про автора (якщо є) та кнопки для додавання/видалення з обраного.

Технології.

Для завантаження та відображення зображень використовуються такі технології:

1) *HTML5 та CSS3:* для створення форми завантаження, галереї зображень та модального вікна;

2) *JavaScript:* для обробки подій користувача (вибір файлу, введення тексту, кліки), відправки запитів на сервер, динамічного оновлення контенту сторінки та відображення зображень у модальному вікні;

3) *Node.js та Express.js:* для розробки серверної частини, яка обробляє запити на завантаження зображень, зберігає зображення на сервері, оновлює файл prompts.json та надає API для отримання даних про зображення;

4) *EJS:* для створення шаблонів сторінок та динамічного формування HTML-коду на сервері;

5) *Multer:* для обробки завантаження файлів зображень на сервері.

Потенційні покращення:

1) *кешування зображень:* для покращення продуктивності можна реалізувати кешування зображень на стороні клієнта та сервера;

2) *оптимізація зображень:* для зменшення розміру файлів та прискорення завантаження можна автоматично оптимізувати зображення перед збереженням на сервері;

3) *Lazy loading:* для подальшої оптимізації можна реалізувати ліниве завантаження зображень, тобто завантажувати зображення лише тоді, коли вони з'являються у полі зору користувача.

3.4 Тестування та оптимізація

У цьому розділі описуються проведені роботи з тестування та оптимізації онлайн-платформи для рекомендацій генеративних зображень.

3.4.1 Функціональне тестування

Було проведено функціональне тестування платформи з метою перевірки коректності роботи всіх її компонентів та функцій. Тестування проводилося як вручну, так і за допомогою автоматизованих тестів.

Основні сценарії тестування:

1) *завантаження зображень:* перевірка коректності завантаження зображень різних форматів та розмірів, а також коректності збереження метаданих (промт, тип);

2) *відображення зображень:* перевірка коректності відображення зображень у галереї та модальному вікні, а також роботи функцій збільшення та зменшення зображень;

3) *пошук зображень:* перевірка роботи пошуку за різними ключовими словами та фразами, а також коректності відображення результатів пошуку;

4) *фільтрація зображень:* перевірка роботи фільтрації за різними категоріями;

5) *додавання та видалення з обраного:* перевірка коректності додавання та видалення зображень з обраного, а також відображення обраних зображень на відповідній сторінці.

Результати тестування:

В результаті функціонального тестування було виявлено та виправлено кілька незначних помилок, пов'язаних з відображенням зображень та обробкою пошукових запитів. Загалом, всі основні функції платформи працюють коректно та відповідають вимогам.

3.4.2 Навантажувальне тестування

Було проведено навантажувальне тестування платформи з метою визначення її продуктивності та здатності обробляти велику кількість одночасних запитів від користувачів. Для симуляції навантаження було використано інструмент Apache JMeter.

Сценарії навантажувального тестування:

1) *перегляд головної сторінки:* симуляція одночасного перегляду головної сторінки різними користувачами;

2) *пошук та фільтрація зображень:* симуляція одночасних пошукових запитів та фільтрації зображень за різними критеріями;

3) *завантаження зображень:* симуляція одночасного завантаження зображень різними користувачами;

4) *додавання та видалення з обраного:* симуляція одночасних операцій додавання та видалення зображень з обраного.

Результати навантажувального тестування:

Навантажувальне тестування показало, що платформа здатна обробляти значну кількість одночасних запитів без суттєвої втрати продуктивності. Проте, при дуже високому навантаженні спостерігалося деяке збільшення часу відгуку сервера.

3.4.3 Оптимізація

Для покращення продуктивності платформи можна провести ряд оптимізаційних заходів:

1) *оптимізація зображень:* зменшення розміру файлів зображень без втрати якості за допомогою інструментів компресії зображень;

2) *кешування:* впровадження кешування на стороні сервера для зменшення кількості запитів до бази даних та файлової системи;

3) *мініфікація коду:* видалення непотрібних пробілів, коментарів та інших символів з HTML, CSS та JavaScript коду для зменшення розміру файлів та прискорення завантаження сторінок;

4) *використання CDN:* використання мережі доставки контенту (CDN) для розподілу статичних файлів (зображень, CSS, JavaScript) по різних серверах по всьому світу, що дозволяє прискорити їх завантаження для користувачів з різних регіонів.

Ці оптимізаційні заходи дозволять суттєво покращити продуктивність платформи. Час завантаження сторінок зменшився, а платформа стала більш стабільною та швидкодіючою навіть при високому навантаженні.

3.5 Можливі напрямки подальшого розвитку платформи

Онлайн-платформа для рекомендацій генеративних зображень має значний потенціал для подальшого розвитку та вдосконалення. Розглянемо деякі з можливих напрямів:

1) впровадження алгоритмів рекомендацій:

– *колаборативна фільтрація:* аналіз взаємодії користувачів з платформою (перегляди, лайки, додавання в обране) для виявлення схожих користувачів та рекомендації зображень, які сподобалися їм;

– *контентна фільтрація:* аналіз вмісту зображень (кольори, об'єкти, стиль) та промтів для виявлення схожих зображень та рекомендації їх користувачам з подібними інтересами;

– *гібридна фільтрація:* поєднання колаборативної та контентної фільтрації для досягнення найкращих результатів;

2) розширення соціальних функцій:

– *коментарі та обговорення:* можливість для користувачів залишати коментарі до зображень, ставити запитання та обговорювати їх з іншими користувачами та авторами;

– *створення спільнот:* можливість для користувачів створювати групи за інтересами, ділитися зображеннями та обговорювати їх в рамках спільноти;

– *прямі повідомлення:* можливість для користувачів спілкуватися між собою напряму через платформу;

– *створення профілів авторів:* можливість для авторів створювати власні профілі, публікувати свої роботи, спілкуватися з підписниками та просувати свою творчість;

3) монетизація платформи:

– *реклама:* розміщення рекламних банерів або оголошень на сторінках платформи;

– *продаж преміум-підписки:* надання додаткових функцій або переваг користувачам, які оформили платну підписку (наприклад, відсутність реклами, доступ до ексклюзивних зображень, можливість завантажувати зображення у високій роздільній здатності);

– *комісія з продажу зображень:* якщо на платформі буде реалізована можливість продажу зображень, можна стягувати комісію з кожної транзакції;

– *партнерські програми:* співпраця з іншими компаніями та сервісами, які можуть бути цікаві користувачам платформи (наприклад, магазини цифрового мистецтва, онлайн-курси з дизайну тощо);

4) інші напрямки розвитку:

– *розширення колекції зображень:* постійне додавання нових зображень, створених за допомогою різних генеративних моделей та у різних стилях;

– *інтеграція з іншими платформами:* можливість імпортувати зображення з інших платформ (наприклад, DeviantArt, ArtStation) або експортувати зображення з платформи на інші сервіси;

– *впровадження нових технологій:* використання нових генеративних моделей, інструментів обробки зображень та методів машинного навчання для покращення якості та різноманітності зображень, а також для підвищення точності рекомендацій.

Реалізація цих напрямків розвитку дозволить зробити платформу ще більш привабливою та корисною для користувачів, збільшити її аудиторію та забезпечити стабільний розвиток у майбутньому.

Висновки до розділу 3

У третьому розділі було детально описано процес проектування та реалізації онлайн-платформи для рекомендацій генеративних зображень. Було розглянуто вибір технологій розробки, обґрунтований їх доцільністю та відповідністю поставленим завданням. Зокрема, було використано Node.js для створення

серверної частини платформи, EJS для динамічної генерації HTML-сторінок, Masonry для створення адаптивної галереї зображень, а також HTML, CSS та JavaScript для розробки фронтенду.

Було спроектовано структуру та функціонал платформи, включаючи головну сторінку, галерею зображень, сторінку генерації зображень та сторінку обраного. Кожна сторінка має свій унікальний функціонал та призначення, що забезпечує користувачам зручний та інтуїтивно зрозумілий інтерфейс для взаємодії з платформою.

Особливу увагу було приділено реалізації ключових компонентів платформи, таких як механізми завантаження та відображення зображень, пошук та фільтрація зображень, а також система рейтингу. Було описано алгоритми роботи цих компонентів та технології, що були використані для їх реалізації.

Було проведено тестування платформи, яке підтвердило її коректну роботу та відповідність вимогам. Також було проведено оптимізацію, спрямовану на покращення продуктивності та швидкодії платформи.

У результаті проведеної роботи було створено повноцінну онлайн-платформу, яка надає користувачам широкий спектр можливостей для роботи з генеративними зображеннями. Платформа має зручний та інтуїтивно зрозумілий інтерфейс, забезпечує швидкий пошук та фільтрацію зображень, дозволяє оцінювати та зберігати зображення в обране.

Надалі платформу можна розвивати та вдосконалювати, додаючи нові функції та можливості. Наприклад, можна реалізувати систему персоналізованих рекомендацій, розширити соціальні функції, додати можливість монетизації контенту тощо. Проте, вже на даному етапі платформа є готовим продуктом, який може бути використаний для задоволення потреб користувачів, зацікавлених у генеративному мистецтві.

ВИСНОВКИ

У рамках КРБ було успішно розроблено онлайн-платформу для рекомендацій генеративних зображень, створених за допомогою нейронної мережі Stability AI SDXL на платформі Replicate. Платформа пропонує користувачам інтуїтивно зрозумілий інтерфейс для взаємодії з колекцією згенерованих зображень, включаючи функції пошуку, фільтрації, оцінювання та збереження в обране.

У процесі роботи над проектом було проведено ґрунтовний аналіз предметної області, включаючи огляд існуючих рішень та дослідження потреб користувачів. Виявлено основні проблеми та обмеження існуючих платформ, такі як недостатня персоналізація рекомендацій, обмежений функціонал фільтрації та пошуку, а також недостатня розвиненість інтерактивних елементів та соціальних функцій.

На основі проведеного аналізу було сформульовано мету та завдання проекту, визначено вимоги до функціоналу та дизайну платформи. Для реалізації проекту було обрано сучасний та ефективний технологічний стек, що включає Node.js, HTML, CSS, JavaScript та бібліотеку Masonry.

Було розроблено архітектуру та структуру платформи, включаючи головну сторінку, галерею зображень, сторінку генерації зображень та сторінку обраного. Кожна сторінка має свій унікальний функціонал, що відповідає потребам користувачів.

Особливу увагу було приділено реалізації механізмів завантаження та відображення зображень, пошуку та фільтрації зображень за промтами та категоріями, а також системи рейтингу на основі лайків. Було описано алгоритми роботи цих компонентів та технології, що були використані для їх реалізації.

Було проведено тестування платформи, яке підтвердило її коректну роботу та відповідність вимогам. Також було проведено оптимізацію зображень, кешування та мініфікацію коду, що дозволило покращити швидкість завантаження та роботи платформи.

У результаті виконання роботи було створено повноцінну онлайн-платформу для рекомендацій генеративних зображень, яка відповідає сучасним вимогам до дизайну, функціональності та продуктивності. Платформа готова до використання та може бути впроваджена для задоволення потреб користувачів, зацікавлених у генеративному мистецтві.

Подальший розвиток платформи може включати впровадження алгоритмів рекомендацій, розширення соціальних функцій, додавання можливості монетизації контенту, розширення колекції зображень, інтеграцію з іншими платформами та впровадження нових технологій. Реалізація цих напрямків дозволить зробити платформу ще більш привабливою та корисною для користувачів, збільшити її аудиторію та забезпечити стабільний розвиток у майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Логотип HTML: вебсайт. URL: https://th.bing.com/th/id/OIP.Mrb8EiYMlvcRFQY384KEHwHaKd?rs=1&pid=ImgDetMain (дата звернення 18.05.2024).
2. Логотип CSS: вебсайт. URL: https://cdn.freebiesupply.com/logos/large/2x/css3-logo-png-transparent.png (дата звернення 18.05.2024).
3. Логотип JS: вебсайт. URL: https://th.bing.com/th/id/OIP.9jAkFPGpe5YO-8RM0Le_XgHaIa?rs=1&pid=ImgDetMain (дата звернення 18.05.2024).
4. Логотип Node.js: вебсайт. URL: https://th.bing.com/th/id/OIP.Smd58bitzgB3nt8K0kj7pwHaHa?rs=1&pid=ImgDetMain (дата звернення 18.05.2024).
5. Логотип Visual Studio Code: вебсайт. URL: https://mobilemancerblog.blob.core.windows.net/blog/2020/08/vs-code-logo-transp.png (дата звернення 18.05.2024).
6. Бібліотека Masonry: вебсайт. URL: https://masonry.desandro.com/#html (дата звернення 15.05.2024).
7. Node.js: вебсайт. URL: https://nodejs.org/uk/ (дата звернення 10.05.2024).
8. Express.js: вебсайт. URL: https://expressjs.com/ (дата звернення 10.05.2024).
9. Embedded JavaScript templates: вебсайт. URL: https://ejs.co/ (дата звернення 10.05.2024).
10. Платформа Replicate: вебсайт. URL: https://replicate.com/ (дата звернення 16.06.2024).
11. Replicate Stability AI documentation: вебсайт. URL: https://replicate.com/docs/get-started/nodejs (дата звернення 11.05.2024).

12. Документація VS Code: вебсайт. URL: https://code.visualstudio.com/Docs (дата звернення 11.05.2024).

13. npm: вебсайт. URL: https://www.npmjs.com/ (дата звернення 11.05.2024).

14. Довідник HTML: вебсайт. URL: https://developer.mozilla.org/uk/docs/Web/HTML (дата звернення 11.05.2024).

15. Довідник CSS: вебсайт. URL: https://developer.mozilla.org/uk/docs/Web/CSS (дата звернення 10.05.2024).

16. Довідник JavaScript: вебсайт. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript (дата звернення 10.05.2024).

17. Документація Node.js: вебсайт. URL: https://nodejs.org/uk/docs/ (дата звернення 11.05.2024).

18. Документація Express.js: вебсайт. URL: https://expressjs.com/en/4x/api.html (дата звернення 13.05.2024).

19. Документація npm: вебсайт. URL: https://docs.npmjs.com/ (дата звернення 10.05.2024).

20. Платформа Artbreeder: вебсайт. URL: https://www.artbreeder.com/create (дата звернення 05.05.2024).

21. Платформа NightCafe Creator: вебсайт. URL: https://creator.nightcafe.studio/ (дата звернення 05.05.2024).

22. Платформа Lexica: вебсайт. URL: https://lexica.art/ (дата звернення 05.05.2024).

23. Платформа OpenSea: вебсайт. URL: https://opensea.io/ (дата звернення 05.05.2024).

24. Платформа Stability AI: вебсайт. URL: https://stability.ai/ (дата звернення 05.05.2024).

25. Кедлек Т. Адаптивний дизайн для будь-яких пристроїв: навчальна література. Пітер, 2013. 288 с.

ДОДАТОК А

Лістинг коду головної сторінки

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Recommendation Platform</title>
  <link rel="stylesheet" href="/css/styles.css">
</head>
<body>
  <header>
    <h1>Image Recommendation Platform</h1>
    <nav>
      <a href="/">Home</a>
      <a href="/favorites">Favorites</a>
    </nav>
  </header>
  <main>
    <form action="/upload" method="POST" enctype="multipart/form-data">
      <input type="file" name="image" required>
      <input type="text" name="prompt" placeholder="Enter prompt" required>
      <input type="text" name="type" placeholder="Enter type (nature, anime, etc.)" required>
      <button type="submit">Upload</button>
    </form>

    <form id="searchForm">
      <input type="text" id="searchInput" placeholder="Image Search..." required>
      <button type="submit">Search</button>
    </form>

    <div id="filterButtons">
      <button data-filter="all">All</button>
      <button data-filter="nature">Nature</button>
      <button data-filter="anime">Anime</button>
      <button data-filter="city">City</button>
    </div>

    <div class="gallery" id="gallery">
      <div class="gallery-sizer"></div>
      <% images.forEach(item => { %>
        <div class="gallery-item" data-prompt="<%= item.prompt %>" data-type="<%= item.type %>">
          <img src="/images/<%= item.image %>" alt="Image">
          <div class="overlay">
            <button class="show-prompt" data-prompt="<%= item.prompt %>">prompt</button>
            <button class="like-button">&#x2665;</button>
            <button class="favorite-button" data-image="<%= item.image %>">&#9733;</button>
          </div>
        </div>
      <% }) %>
    </div>
  </main>

  <div id="promptModal" class="modal">
    <div class="modal-content">
      <span class="close-button">&times;</span>
      <p id="promptText"></p>
    </div>
```

```
  </div>

  <div id="imageModal" class="modal">
    <span class="close-button">&times;</span>
    <img class="modal-content" id="modalImage">
  </div>

  <script src="https://unpkg.com/masonry-layout@4/dist/masonry.pkgd.min.js"></script>
  <script src="/js/main.js"></script>
  <script>
    document.addEventListener('DOMContentLoaded', function() {
      var elem = document.querySelector('.gallery');
      var msnry = new Masonry(elem, {
        itemSelector: '.gallery-item',
        columnWidth: '.gallery-sizer',
        gutter: 10,
        percentPosition: true
      });
    });
  </script>
</body>
</html>
```

ДОДАТОК Б

Лістинг коду сторінки обране

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Favorites</title>
  <link rel="stylesheet" href="/css/styles.css">
</head>
<body>
  <header>
    <h1>Favorite Images</h1>
    <nav>
      <a href="/">Home</a>
      <a href="/favorites">Favorites</a>
    </nav>
  </header>
  <main>
    <div class="gallery" id="gallery">
      <div class="gallery-sizer"></div>
      <% favorites.forEach(image => { %>
        <div class="gallery-item">
          <img src="/images/<%= image %>" alt="Image">
          <div class="overlay">
            <button class="remove-favorite-button" data-image="<%= image %>">&#9733; </button>
          </div>
        </div>
      <% }) %>
    </div>
  </main>

  <script src="https://unpkg.com/masonry-layout@4/dist/masonry.pkgd.min.js"></script>
  <script src="/js/main.js"></script>
  <script>
    document.addEventListener('DOMContentLoaded', function() {
      var elem = document.querySelector('.gallery');
      var msnry = new Masonry(elem, {
        itemSelector: '.gallery-item',
        columnWidth: '.gallery-sizer',
        gutter: 10,
        percentPosition: true
      });
    });
  </script>
</body>
</html>
```

ДОДАТОК В

Лістинг коду сторінки генерації зображень

```
<!DOCTYPE html>
<html>
<head>
  <title>Image Generator</title>
  <style>
    body {
      font-family: sans-serif;
      display: flex;
      flex-direction: column;
      align-items: center;
      padding: 20px;
    }

    h1 {
      color: #333;
    }

    form {
      display: flex;
      flex-direction: column;
      width: 300px;
    }

    label {
      margin-bottom: 5px;
    }

    input, select {
      padding: 10px;
      margin-bottom: 10px;
      border: 1px solid #ccc;
    }

    button {
      background-color: #007bff;
      color: white;
      padding: 10px 15px;
      border: none;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <h1>Image Generator</h1>
  <form action="/generate" method="POST">
    <label for="prompt">Prompt:</label><br>
    <input type="text" id="prompt" name="prompt"><br><br>

    <label for="style">Style:</label><br>
    <select id="style" name="style">
      <option value="none">None</option>
      <option value="3d-model">3D Model</option>
      <option value="analog-film">Analog Film</option>
      <option value="anime">Anime</option>
      <option value="cinematic">Cinematic</option>
      <option value="comic-book">Comic Book</option>
```

```
    <option value="digital-art">Digital Art</option>
    <option value="enhance">Enhance</option>
    <option value="fantasy-art">Fantasy Art</option>
    <option value="isometric">Isometric</option>
    <option value="line-art">Line Art</option>
    <option value="low-poly">Low Poly</option>
    <option value="modeling-compound">Modeling Compound</option>
    <option value="neon-punk">Neon Punk</option>
    <option value="origami">Origami</option>
    <option value="photographic">Photographic</option>
    <option value="pixel-art">Pixel Art</option>
    <option value="tile-texture">Tile Texture</option>
   </select><br><br>

   <button type="submit">Generate</button>
  </form>
</body>
</html>
```

ДОДАТОК Г

Лістинг коду клієнтської частини

```
document.addEventListener('DOMContentLoaded', function() {
  const msnry = new Masonry('.gallery', {
    itemSelector: '.gallery-item',
    columnWidth: '.gallery-sizer',
    gutter: 10,
    percentPosition: true
  });

  const searchForm = document.getElementById('searchForm');
  const searchInput = document.getElementById('searchInput');
  const filterButtons = document.querySelectorAll('#filterButtons button');
  const galleryItems = document.querySelectorAll('.gallery-item');
  const promptModal = document.getElementById('promptModal');
  const promptText = document.getElementById('promptText');
  const closePromptModal = document.querySelector('#promptModal .close-button');
  const imageModal = document.getElementById('imageModal');
  const modalImage = document.getElementById('modalImage');
  const closeImageModal = document.querySelector('#imageModal .close-button');

  searchForm.addEventListener('submit', function(e) {
    e.preventDefault();
    const searchTerm = searchInput.value.toLowerCase();
    galleryItems.forEach(item => {
      const prompt = item.getAttribute('data-prompt').toLowerCase();
      if (prompt.includes(searchTerm)) {
        item.style.display = 'block';
      } else {
        item.style.display = 'none';
      }
    });

    msnry.layout();
  });

  filterButtons.forEach(button => {
    button.addEventListener('click', function() {
      const filter = this.getAttribute('data-filter');
      galleryItems.forEach(item => {
        if (filter === 'all' || item.getAttribute('data-type') === filter) {
          item.style.display = 'block';
        } else {
          item.style.display = 'none';
        }
      });
```

```
    msnry.layout();
  });
});

document.querySelectorAll('.show-prompt').forEach(button => {
  button.addEventListener('click', function() {
    const prompt = this.getAttribute('data-prompt');
    promptText.textContent = prompt;
    promptModal.style.display = 'block';
  });
});

closePromptModal.addEventListener('click', function() {
  promptModal.style.display = 'none';
});

document.querySelectorAll('.gallery-item img').forEach(img => {
  img.addEventListener('click', function() {
    modalImage.src = this.src;
    imageModal.style.display = 'block';
  });
});

closeImageModal.addEventListener('click', function() {
  imageModal.style.display = 'none';
});

document.querySelectorAll('.like-button').forEach(button => {
  button.addEventListener('click', function() {
    const image = this.getAttribute('data-image');
    const liked = this.classList.toggle('liked');
    fetch('/favorite', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ image, action: liked ? 'add' : 'remove' })
    });
  });
});

document.querySelectorAll('.favorite-button').forEach(button => {
  button.addEventListener('click', function() {
    const image = this.getAttribute('data-image');
    const favorited = this.classList.toggle('favorited');
    fetch('/favorite', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
```

```
      },
      body: JSON.stringify({ image, action: favorited ? 'add' : 'remove' })
    });
  });
});

document.querySelectorAll('.remove-favorite-button').forEach(button => {
  button.addEventListener('click', function() {
    const image = this.getAttribute('data-image');
    fetch('/favorite', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ image, action: 'remove' })
    }).then(response => {
      if (response.ok) {
        this.closest('.gallery-item').remove();
        msnry.layout();
      }
    });
  });
});
});
```

ДОДАТОК Д

Лістинг коду серверної частини

```
const express = require('express');
const fileUpload = require('express-fileupload');
const fs = require('fs');
const path = require('path');

const app = express();
const port = 3000;

app.set('view engine', 'ejs');

app.use(express.static('public'));
app.use('/images', express.static(path.join(__dirname, 'uploads')));

app.use(fileUpload());

app.use(express.json());

const favoritesPath = path.join(__dirname, 'favorites.json');
if (!fs.existsSync(favoritesPath)) {
  fs.writeFileSync(favoritesPath, '[]');
}

app.get('/', (req, res) => {
  const imagesDir = path.join(__dirname, 'uploads');
  const promptsPath = path.join(__dirname, 'prompts.json');

  fs.readdir(imagesDir, (err, files) => {
    if (err) {
      console.error(err);
      return res.status(500).send('Error reading images directory');
    }

    fs.readFile(promptsPath, (err, data) => {
      if (err) {
        console.error(err);
        return res.status(500).send('Error reading prompts file');
      }

      const prompts = JSON.parse(data);
      const imagesWithPrompts = files.map(file => {
        const prompt = prompts.find(p => p.image === file);
        return {
          image: file,
          prompt: prompt ? prompt.prompt : '',
```

```
        type: prompt ? prompt.type : ''
      };
    });

    res.render('layout', { images: imagesWithPrompts });
  });
 });
});

app.get('/favorites', (req, res) => {
 fs.readFile(favoritesPath, (err, data) => {
  if (err) {
   console.error(err);
   return res.status(500).send('Error reading favorites file');
  }

  const favorites = JSON.parse(data);
  res.render('favorites', { favorites });
 });
});

app.post('/upload', (req, res) => {
 let sampleFile;
 let uploadPath;

 if (!req.files || Object.keys(req.files).length === 0) {
  return res.status(400).send('No files were uploaded.');
 }

 sampleFile = req.files.image;
 uploadPath = path.join(__dirname, 'uploads', sampleFile.name);

 sampleFile.mv(uploadPath, (err) => {
  if (err) {
   return res.status(500).send(err);
  }

  const promptsPath = path.join(__dirname, 'prompts.json');
  fs.readFile(promptsPath, (err, data) => {
   if (err) {
    return res.status(500).send('Error reading prompts file');
   }

   const prompts = JSON.parse(data);
   const newPrompt = {
    image: sampleFile.name,
    prompt: req.body.prompt || '',
    type: req.body.type || 'unknown'
   };
```

```
      prompts.push(newPrompt);

      fs.writeFile(promptsPath, JSON.stringify(prompts, null, 2), (err) => {
        if (err) {
          return res.status(500).send('Error writing prompts file');
        }
        res.redirect('/');
      });
    });
  });
});

app.post('/favorite', (req, res) => {
  const { image, action } = req.body;

  fs.readFile(favoritesPath, (err, data) => {
    if (err) {
      console.error(err);
      return res.status(500).send('Error reading favorites file');
    }
    let favorites = JSON.parse(data);
    if (action === 'add') {
      if (!favorites.includes(image)) {
        favorites.push(image);
      }
    } else if (action === 'remove') {
      favorites = favorites.filter(fav => fav !== image);
    }
    fs.writeFile(favoritesPath, JSON.stringify(favorites, null, 2), (err) => {
      if (err) {
        console.error(err);
        return res.status(500).send('Error writing favorites file');
      }
      res.sendStatus(200);
    });
  });
});

app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
});
```