Міністерство освіти і науки України
Харківський національний автомобільно-дорожній університет

Механічний факультет

Кафедра комп’ютерних наук і інформаційних систем

ДИПЛОМНА РОБОТА

магістра

РОЗРОБКА ВДОСКОНАЛЕНОЇ ПЛАТФОРМИ ДЛЯ ГЕНЕРАЦІЇ ТА УПРАВЛІННЯ ЗОБРАЖЕННЯМИ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Завідувачка кафедри доцентка, к.т.н.	Ганна ПЛЄХОВА
Нормоконтролер, к.т.н., доцент	Сергій НЕРОНОВ
Керівник, док. філ. доцент	Андрій ЛЕБЕДИНСЬКИЙ
Студент гр. МК-61-24	Дмитро КИРИЛОВ

Харків – 2025

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ АВТОМОБІЛЬНО-ДОРОЖНІЙ УНІВЕРСИТЕТ

Факультет механічний
Кафедра комп'ютерних наук і інформаційних систем
Освітній рівень другий (магістр)
Спеціальність F3 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувачка кафедри
____________ Г. А. Плєхова
«___» ___________ 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Кирилову Дмитру Іллічу

1. Тема роботи: «Розробка вдосконаленої платформи для генерації та управління зображеннями з використанням штучного інтелекту».
Керівник роботи: Лебединський Андрій Володимирович, доктор філософії, доц.
Затверджені рішенням Вченої ради механічного факультету «8» жовтня 2025 року, протокол № 155.

2. Строк подання студентом роботи «15» грудня 2025 року.

3. Вхідні данні до роботи: звіт з переддипломної практики.

4. Перелік питань, які потрібно розробити: 1 Проаналізувати сучасні систи генерації зображень на основі штучного інтелекту (DALL-E, Stable Diffusion, Midjourney) та порівняльний аналіз їх функціональних можливостей; 2 Обґрунтувати вибір технологічного стеку (Node.js, Express, MongoDB) для розробки платформи Pic.AI; 3 Розробка структури бази даних MongoDB з схемами User та imageHistory; 4 Реалізація системи автентифікації та авторизації користувачів з використанням bcrypt та express-session; 5 Розробка функціоналу генерації зображень через інтеграцію з OpenAI API, включаючи збереження та управління файлами; 6 Створення клієнтського інтерфейсу з адаптивним дизайном, системою багатомовності (i18n) та історією генерацій; 7 Тестування функціональності, безпеки та продуктивності застосунку; 8 Розгортання платформи, налаштування хостингу.

5. Перелік графічного матеріалу (з точним значенням обов'язкових креслень): 1 Титульний лист. 2 Мета роботи, об'єкт дослідження та задачі. 3 Аналіз існуючих

рішень. 4 Обгрунтування вибору технологічного стеку. 5 Архітектура платформи. 6 Дизайн та UX. 7. Реалізація серверної частини. 8 Реалізація клієнтської частини. 9 Тестування та якість. 10. Розгортання та використання. 11 Висновки.

6. Дата видачі завдання 9 жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Проаналізувати сучасні системи генерації зображень на основі штучного інтелекту (DALL-E, Stable Diffusion, Midjourney) та порівняльний аналіз їх функціональних можливостей.	09.10.2025 - 22.10.2025	
2	Обґрунтувати вибір технологічного стеку (Node.js, Express, MongoDB) для розробки платформи Pic.AI	23.10.2025 - 29.10.2025	
3	Реалізація системи автентифікації та авторизації користувачів з використанням bcrypt та express-session.	30.10.2025 - 05.11.2025	
4	Розробка функціоналу генерації зображень через інтеграцію з OpenAI API, включаючи збереження та управління файлами	06.11.2025 - 12.11.2025	
5	Створення клієнтського інтерфейсу з адаптивним дизайном, системою багатомовності (i18n) та історією генерацій.	13.11.2025 - 26.11.2025	
6	Тестування функціональності та безпеки застосунку.	27.11.2025 - 05.12.2025	
7	Розгортання платформи в «production» середовищі, налаштування хостингу.	06.12.2025 - 10.12.2025	
8	Оформлення дипломної роботи, оформлення документації та підготовка до захисту.	11.12.2025 - 15.12.2025	

Студент ____________ Кирилов Д. І.

Керівник роботи ____________ Лебединський А. В.

РЕФЕРАТ

Дипломна робота містить: 128 с., 20 рис., 36 посилань.

ВЕБПЛАТФОРМА, ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ, OPENAI, ШТУЧНИЙ ІНТЕЛЕКТ, NODEJS, УПРАВЛІННЯ, НЕРОННІ МЕРЕЖІ.

Об'єкт дослідження – процес генерації та управління цифровими зображеннями з використанням технологій штучного інтелекту в вебсередовищі.

Предмет дослідження – методи, алгоритми та технології проєктування і реалізації веборієнтованої платформи Pic.AI для генерації зображень з використанням API OpenAI DALL-E 3.

Мета роботи – розробка вдосконаленої веборієнтованої платформи Pic.AI для автоматизованої генерації зображень на основі текстових описів з використанням технологій штучного інтелекту OpenAI DALL-E 3, що забезпечує високу якість візуалізації, зручність користувацького інтерфейсу та ефективне управління історією генерацій.

У роботі використано: методи системного аналізу – для порівняння існуючих аналогів; методи об'єктно-орієнтованого програмування – для розробки серверної та клієнтської частин; методи моделювання баз даних – для проєктування структури MongoDB; методи тестування програмного забезпечення – для перевірки працездатності системи.

У роботі було проаналізовано існуючі рішення у сфері AI-генерації зображень та розробки вебплатформ для роботи з штучним інтелектом. Проведено порівняльний аналіз провідних платформ з метою виявлення їх переваг, недоліків та визначення оптимальних підходів до проєктування власної системи. Було здійснено розробку вебплатформи Pic.AI для автоматизованої генерації зображень на основі текстових описів з використанням API OpenAI DALL-E 3. Платформа реалізована на повному стеку сучасних технологій: клієнтська частина (HTML5 для структурування інтерфейсу; CSS3 з адаптивним дизайном; JavaScript для динамічної взаємодії; система

інтернаціоналізації (i18n.js) з підтримкою української та англійської мов та серверна частини (Node.js; express.js для створення RESTful API; MongoDB Atlas як NoSQL база даних; Mongoose для моделювання даних; OpenAI SDK для інтеграції з DALL-E 3; Axios для завантаження згенерованих зображень; асинхронні запити через Fetch API).

Результатом роботи є повнофункціональна вебплатформа Pic.AI, яка може бути використана як інструмент для автоматизації роботи дизайнерів, SMM-спеціалістів та веброзробників. Реалізована система дозволяє централізовано створювати, переглядати та зберігати зображення, що оптимізує робочий час та спрощує доступ до технологій ШІ для пересічних користувачів.

Під час розробки дипломної роботи автором було опубліковано наукову роботу, а саме: Лебединський А.В., Кирилов Д. І. Збереження та організація згенерованих AI-зображень: прості підходи. *Перспективи розвитку інформаційних систем*: матеріали Міжнар. наук. конф., м. Харків, 17 жовт. 2025 р. Харків, 2025. С. 48-52 (Додаток Б).

ЗМІСТ

Вступ 7
1 Аналіз сучасних технологій генерації зображень на основі штучного інтелекту і аналіз вимог до вебплатформи 9
1.1 Історія розвитку штучного інтелекту 9
1.2 Огляд існуючих систем генерації зображень 12
1.3 Порівняльний аналіз кожної з платформ 16
1.4 Аналіз вимог до платформи генерації зображень 22
1.5 Обгрунтування вибору технологічного стеку 29
2 Проєктування архітектури вебплатформи Pic.AI 36
2.1 Концептуальна модель вебплатформи 36
2.2 Взаємодія компонентів системи 46
2.3 Діаграма послідовностей обробки запиту генерації зображення 50
3 Реалізація вебплатформи Pic.AI 55
3.1 Розробка серверної частини вебплатформи 55
3.2 Робота з базою даних MongoDB 68
3.3 Розробка клієнтської частини вебплатформи 70
4 Тестування вебплатформи 76
4.1 Модульне тестування 76
4.2 Тестування безпеки 78
Висновки 81
Перелік посилань 83
Додаток А Лістинг коду вебплатформи 86
Додаток Б Текст статті 113
Додаток В Ілюстративний матеріал до дипломної роботи 118

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімкою інтеграцією інструментів штучного інтелекту (ШІ) у творчі та виробничі процеси. Генеративні моделі, такі як DALL-E, Midjourney та Stable Diffusion, докорінно змінили підходи до створення візуального контенту, дозволяючи отримувати високоякісні зображення за текстовим описом за лічені секунди. Це відкриває нові можливості для дизайнерів, маркетологів та контент-мейкерів, значно скорочуючи час на розробку графічних матеріалів.

Проте, незважаючи на доступність цих моделей, більшість існуючих інтерфейсів мають суттєві обмеження щодо зручності управління контентом. Користувачі стикаються з проблемою хаотичного зберігання генерацій, відсутністю структурованої історії, складнощами у повторному використанні вдалих текстових запитів (промптів) та обмеженими можливостями персоналізації. У зв’язку з цим, актуальним завданням є розробка спеціалізованої вебплатформи, яка не лише надає доступ до потужностей генеративних нейромереж через API, але й забезпечує зручне середовище для систематизації, зберігання та управління створеним контентом. Створення платформи Pic.AI, що поєднує сучасний стек вебтехнологій з можливостями DALL-E 3, вирішує проблему ефективної взаємодії користувача з генеративним ШІ, що і зумовлює вибір теми магістерської роботи.

Метою роботи є розробка вдосконаленої веборієнтованої платформи Pic.AI для автоматизованої генерації зображень на основі текстових описів з використанням технологій штучного інтелекту OpenAI DALL-E 3, що забезпечує високу якість візуалізації, зручність користувацького інтерфейсу та ефективне управління історією генерацій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні методи генерації зображень та існуючі програмні рішення (DALL-E, Midjourney, Stable Diffusion), виявити їхні переваги та недоліки;

- обґрунтувати вибір архітектурного підходу та технологічного стеку (Node.js, Express, MongoDB) для реалізації платформи;
- розробити архітектуру системи та структуру бази даних для зберігання профілів користувачів та історії генерацій;
- реалізувати програмні модулі автентифікації, взаємодії з API DALL-E 3 та обробки зображень;
- створити адаптивний клієнтський інтерфейс із підтримкою багатомовності (i18n).
- провести тестування функціональності, безпеки та продуктивності розробленої вебплатформи.

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ І АНАЛІЗ ВИМОГ ДО ДОДАТКУ

Штучний інтелект (ШІ) є однією з найважливіших технологічних рушійних сил сучасності, і його вплив охоплює практично всі сфери діяльності людини – від бізнесу й освіти до мистецтва й дизайну. Зокрема, значний прорив відбувся у галузі комп’ютерної генерації зображень, яка поєднує досягнення глибокого навчання, обробки природної мови та комп’ютерного зору. Якщо ще кілька років тому системи ШІ могли лише створювати абстрактні візуальні композиції, то сьогодні вони здатні генерувати реалістичні зображення, що не поступаються фотографіям, а також опрацьовувати складні запити користувача, описані звичайною мовою. Ці досягнення стали можливими завдяки розвитку таких архітектур, як генеративно-змагальні мережі (GAN), автокодери (VAE, VQ-VAE), дифузійні моделі та трансформери. Аналіз існуючих технологій у цій сфері є необхідним етапом для створення власної вдосконаленої платформи, оскільки він дозволяє виявити переваги, обмеження та потенційні напрями інтеграції інноваційних підходів.

1.1 Історія розвитку штучного інтелекту

Історія ШІ охоплює десятиліття досліджень та численні розробки. Від перших спроб відтворити людське мислення на механічних пристроях до сучасних досягнень у галузі глибоких нейромереж, ШІ пройшов значний шлях. Сьогодні штучний інтелект здатний вирішувати складні завдання, аналізувати великі обсяги даних і навіть створювати мистецтво [1].

Концепція віртуального розуму не з'явилася раптово; вона є результатом тривалої еволюції наукових і філософських ідей, технологічних досягнень та суспільних потреб. Передумови для створення штучного інтелекту виникли ще в античні часи. Давньогрецькі філософи вже тоді розмірковували про

можливість появи пристроїв, здатних виявляти ознаки розуму. Деякі приклади, що свідчать про зародження ідеї штучного інтелекту в давнину, включають:

- у міфах Стародавньої Греції зустрічаються автоматони – механічні ляльки, що виконують дії за заданим алгоритмом. Одним із прикладів є Пандора, створена Зевсом;
- у єврейській культурі існують легенди про големів – істот, створених з неживої матерії, які виступають як прообрази сучасного ШІ;
- у 17-му столітті деякі філософи, такі як німецький мислитель і винахідник Лейбніц, почали міркувати про можливість оживити неживі предмети. Лейбніц вважав, що людські думки можна зобразити математично за допомогою спеціальних символів і схем.

Історія розвитку штучного інтелекту бере початок не лише у XX столітті, а й значно раніше – з давніх легенд та міфів, які описували ідеї створення штучних істот з розумом, хоч це були лише метафори. Лише у 1940-х – 1950-х роках ці ідеї починають отримувати наукове підґрунтя: саме тоді були розроблені перші математичні моделі штучних нейронів, а філософи стали активно обговорювати можливість машинного мислення. Ключовим моментом став 1950 рік – вихід роботи Алана Тюрінга «Computing Machinery and Intelligence», у якій математик поставив фундаментальне питання: чи можуть машини думати? Він запропонував відомий сьогодні тест Тюрінга як критерій визначення розумності машини, сформулювавши основні теоретичні засади галузі.

Наступний визначальний етап відбувся у 1956 році – на Дартмутській конференції, організованій Джоном Маккарті, Марвіном Мінскі, Клодом Шенноном і Натаном Рочестером. Там вперше офіційно був вжитий термін "штучний інтелект", а до дискусії залучили провідних фахівців із різних дисциплін: кібернетики, математики, лінгвістики, психології. Вони обговорювали ідеї формалізації інтелектуальних процесів, природної мови, навчання, нейронних мереж і навіть творчості. Дартмутський семінар заклав фундамент для швидкого розвитку декількох підходів до штучного інтелекту і

відкрив двері для подальших досліджень у сфері моделювання розумових процесів на машинах.

Далі у 1960-х і 1970-х роках розпочалась ера оптимізму та перших успіхів: науковці створили перші програми – Logic Theorist та ELIZA (рис. 1.1) – які могли виконувати певні логічні операції і навіть вести простий діалог зі людиною. Паралельно з'явились перш геніальні ідеї щодо нейронних мереж (наприклад, SNARC Мінскі та Едмундса), були створені ранні роботи, а також мови програмування, спеціалізовані під розробку ШІ, такі як LISP. Водночас саме в ці десятиліття стали очевидними серйозні обмеження і недоліки: бракувало обчислювальних ресурсів, даних і теоретичних методів, що призвело до перших періодів "зими" ШІ, коли фінансування та ентузіазм у галузі значно скоротилися.

```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Рисунок 1.1 – Вигляд програми ELIZA – прототипу ШІ

1980-ті та 1990-ті роки ознаменувалися відродженням інтересу завдяки створенню експертних систем, автоматизованих способів обробки знань і появі перших комерційних розробок – наприклад, промислових роботів. Однак ці технології мали обмежене застосування й ефективність, що призвело до другої

"зими" ШІ. Лише на межі тисячоліть із розвитком машинного навчання, комп’ютерного зору та обробки природної мови ШІ повернувся до практичних завдань і почав виходити з вузької академічної сфери на рівень промислового застосування. Важливим проривом було те, що у 1997 році комп'ютер IBM Deep Blue переміг чемпіона світу з шахів Гаррі Каспарова, а у 2000-х почали з'являтися перші домашні інтелектуальні прилади, які використовували алгоритми ШІ для виконання повсякденних завдань.

Отже, ідея створення віртуального розуму має глибоке коріння, що сягає давніх часів. Проте, історія штучного інтелекту почалася відносно недавно, в середині 20-го століття. У цей період відбулося різке зростання кількості досліджень і експериментів, пов'язаних зі створенням машинного розуму, що призвело до перших значних результатів.

Справжня революція розпочалася у 2010-х і триває у 2020-х: зростання обчислювальних потужностей [2], Big Data і глибинного навчання призвели до створення багаторівневих нейромереж, здатних впізнавати зображення, аналізувати великі об’єми даних та природну мову на рівні, що наближається до людського. З'явилися такі значущі технології, як голосові асистенти, автоматичне перекладання, генеративні нейромережі (GPT), застосування ШІ в медицині, науці, бізнесі, освіті. У 2020-х роках ШІ активно допомагає у розробці нових ліків, працює з даними про пандемії та інтегрується у повсякденне життя через застосунки, сервісні роботи та інтелектуальні системи підтримки. Сучасний етап розвитку ознаменувався вибуховим зростанням генеративного ШІ та розширенням його можливостей у творчих галузях, науці й освіті, що робить штучний інтелект ключовим драйвером технологічного прогресу нашого часу [3].

1.2 Огляд існуючих систем генерації зображень

Останніми роками особливу увагу привернули дифузійні нейронні мережі, які змінили парадигму генерації контенту. Принцип їх роботи полягає у

поступовому додаванні шуму до існуючого зображення під час навчання моделі та зворотному процесі його усунення – дифузії – під час генерації. Такий підхід дозволяє досягти надзвичайно високої якості результату, гнучко контролюючи рівень деталізації, стиль і композицію сцени. Саме на основі дифузійних моделей створено такі флагманські системи, як Stable Diffusion, DALL·E 2/3 (рис. 1.2) та MidJourney (рис. 1.3). Наприклад, DALL·E від OpenAI уперше об'єднав мовну модель і візуальний генератор, що дозволило трактувати складні текстові описи і перетворювати їх на ілюстрації. Stable Diffusion, розроблена компанією Stability AI у співпраці з науковими інститутами, вперше зробила подібні технології відкритими, створивши ґрунт для великої спільноти розробників і появи численних модифікацій. Її відкритий код, можливість обробки зображень локально та підтримка різних архітектур перетворили цю технологію на стандарт для інтеграції у сторонні додатки, стартапи та дизайн-платформи [4, 5].

Рисунок 1.2 – Логотип DALL E 3

Інструмент MidJourney став визначним явищем у творчих індустріях, зокрема серед дизайнерів, художників та маркетологів. Його алгоритми орієнтовані на естетичність та узгодженість кольорів, тому результати генерації часто мають характерний художній стиль – від футуристичних композицій до реалістичних портретів. MidJourney взаємодіє з користувачами через чат-

інтерфейс, що дозволяє оновлювати зображення у режимі реального часу, додаючи або змінюючи параметри запиту. Хоча його результати надзвичайно якісні, сервіс має обмежені можливості з точки зору управління даними – користувачі не можуть ефективно класифікувати, групувати чи відстежувати зміни створених зображень. Це робить його переважно інструментом для натхнення, але не для повноцінної організації робочого процесу або комерційного контенту [6].

Рисунок 1.3 – Логотип Midjourney

Водночас великі корпорації розвивають гібридні рішення, які поєднують генерацію та редагування зображень. Яскравим прикладом є Adobe Firefly (рис. 1.4), який інтегрує можливості ШІ у середовище Photoshop і Illustrator. Завдяки функціям текстової генерації зображень, інтелектуального заповнення фону, стилізації та автоматичного створення композицій Firefly дозволяє швидко адаптувати візуальний контент до творчих потреб. Цей підхід ілюструє нову тенденцію: не лише створення зображення з нуля, а й тісна взаємодія ШІ з традиційними творчими інструментами. Однак навіть такі передові системи не вирішують повністю питання організації величезних колекцій контенту, що

виникають у процесі творчої роботи. Збереження, пошук і структуризація таких даних часто залишаються на відповідальності користувача [7].

Рисунок 1.4 – Логотип Adobe FireFly

Окрему нішу займають вебплатформи управління зображеннями на основі ШІ. Сервіси Lexica.art, Playground AI та Hugging Face Spaces демонструють спроби поєднати генеративну функціональність з обмеженими засобами керування. Вони надають бібліотеки зображень, пошук за ключовими словами, можливість фільтрації за параметрами або моделлю генерації. Проте такі рішення залишаються загальнодоступними, без персоналізованого управління або систематизації згенерованого контенту для конкретного користувача чи команди. Як наслідок, постає потреба в універсальних платформах, які пропонують повний цикл роботи: генерацію, редагування, оцінку, зберігання та спільне використання зображень з урахуванням метаданих і семантичного змісту.

Не менш важливою складовою сучасних генеративних систем є архітектура управління процесом генерації. Переважна більшість платформ працює за спрощеною схемою «prompt → результат», яка хоч і забезпечує базову зручність, але не передбачає складного сценарного управління або інтеграції з іншими інструментами. Просунуті користувачі потребують

можливості відстеження історії версій, порівняння варіантів, застосування фільтрів, семантичного класифікаційного пошуку, аналітики якості зображень та автоматичної обробки великих наборів даних. Для цього необхідно поєднати генеративну частину з компонентами систем керування контентом (CMS), базами даних, а також інструментами машинного навчання для аналізу метаданих і змісту зображень. Такі системи можуть підтримувати створення цілісних робочих процесів, де результати генерації не лише зберігаються, а й постійно вдосконалюються користувачем або командою [8].

Серед технічних рішень, доступних сьогодні, окреме місце займають підходи до інтеграції та автоматизації. DALL·E, наприклад, має REST API, що дає змогу підключати його до інших програм, вебсайтів або ботів для автоматизованої генерації контенту. Stable Diffusion надає широкий набір інструментів через бібліотеку diffusers (Hugging Face) і популярні інтерфейси як Automatic1111 WebUI, що дозволяють реалізовувати розширення, API-виклики або створювати власні плагіни. Це створює умови для розробки нових платформ, що об'єднують різні моделі в одному середовищі, забезпечуючи узгоджений інтерфейс для користувача. Більш того, відкритість таких систем стимулює розробку інновацій – наприклад, поєднання моделей тексту та зображень, а також мультимодальних рішень, де обробляються різні типи даних (звук, відео, текст, графіка).

1.3 Порівняльний аналіз кожної з платформ

DALL·E 3, MidJourney, Stable Diffusion і Adobe Firefly вирішують схожу задачу (генерація зображень за текстом), але роблять це з різною філософією: від «максимально зручно й доступно» до «максимально контрольовано й інтегровано в професійний пайплайн» [9].

DALL·E 3 вирізняється глибоким розумінням текстового контексту та інтеграцією з ChatGPT, що полегшує уточнення запитів і спільну роботу над креативними ідеями. Система добре підходить для тих, хто прагне

автоматизувати створення зображень із детальними текстовими сценаріями, особливо при використанні API. Проте існують певні обмеження: DALL·E 3 не дозволяє повністю контролювати дрібні деталі композиції, іноді виникають неточності з кількістю об’єктів, і відсутня власна система управління колекціями результатів.

MidJourney є потужним інструментом для художньої генерації та експериментів із стилями, особливо для фантазійних, сюрреалістичних чи абстрактних сцен. Він ідеально підходить митцям, дизайнерам і творчим командам завдяки спільній роботі через Discord та активній експертній спільноті. Основне обмеження полягає у відсутності інтегрованої системи управління власними зображеннями, а генерація відбувається у межах спільноти або через особисті повідомлення на Discord.

Stable Diffusion пропонує найбільшу гнучкість із точки зору вибору, налаштування параметрів і інтеграції у різні робочі процеси. Відкритий код дозволяє обрати оптимальний інтерфейс, модифікувати архітектуру, впроваджувати власні моделі або доповнювати функціональність ресурсами сторонніх розробників. Платформа підтримує автоматизовані сценарії, має надбудови для організації, класифікації, семантичного пошуку та інтеграції із CMS, що робить її оптимальною для командної роботи й професійного контент-менеджменту.

Adobe Firefly – це комплексний сервіс, спрямований на інтеграцію генеративних інструментів ШІ у бізнес-процеси та творчі екосистеми. Основна перевага полягає у поєднанні високої якості зображень, інтелектуального редагування (Generative Fill, Structure Reference), комерційної безпеки та глибокої інтеграції з іншими продуктами Adobe. Така платформа розглядається як універсальний інструмент для корпоративного сегменту, довготривалих проєктів та синхронізованої роботи команди. Порівняння функцій і особливостей кожної з платформ показано у таблиці 1.1.

Таблиця 1.1 – Порівняння функцій кожної з платформ

Характеристика	DALL·E 3	MidJourney	Stable Diffusion	Adobe Firefly
Тип доступу	Веб, API, інтеграція з ChatGPT	Discord, Web-інтерфейс	Веб, локальна установка, різні UI	Веб, інтеграції з Creative Cloud
Контроль якості зображень	Висока деталізація, добре розпізнає текстові описи, але трапляються помилки з дрібними об’єктами чи кількістю елементів	Висока художня якість, акцент на стилізації, особливо абстрактні чи фантазійні сцени	Від фотореалізму до абстракції, залежить від моделі, гнучкий вибір стилю результату	Висока якість (2K+), виняткова деталізація для професійних цілей, гнучкий вибір стилю, структура та референси
Гнучкість параметрів	Можливість уточнення через ChatGPT, але обмежений контроль композицією та кількістю об’єктів	Управління результатом через prompt, вибір стилю, змінні варіації, інтерактивне доопрацювання	Висока гнучкість налаштувань (UI, параметри, власні моделі), локальне редагування, автоматизація	Детальний контроль композиції (Structural Reference), стилю, генеративне заповнення та розширення
Можливості управління	Зберігання й пошук через API, немає власної бібліотеки контенту чи інструментів категоризації	Відсутність системи структурованого управління результатами, залежність від Discord-каналу	Багато надбудов для організації, збереження, семантичного пошуку, інтеграцій із CMS	Вбудована синхронізація з бібліотекою Creative Cloud, організація та обробка зображень у робочих проєктах
Ліцензія	Комерційне використання із врахуванням обмежень (автори можуть виключити свої роботи з навчання)	Ліцензія із можливістю використання для дизайну, обмеження на масове комерційне використання	Відкритий код, можливості для власної модифікації й інтеграції, комерційність залежить від моделі	Зображення для бізнесу,trained на ліцензованих даних Adobe Stock, IP індемніфікація
Швидкість генерації	Швидко, але із лімітами доступу	Від 30 до 60 секунд,	Швидко (особливо	Миттєва генерація,

	(кількість зображень/година для підписників)	залежить від черги, плану підписки	локальний режим), залежить від конфігурації обладнання	багаторівнева оптимізація під бізнес

Якщо дивитися на тип доступу, то DALL·E 3 найбільш універсальний для широкої аудиторії та розробників, бо поєднує веб, API і природну інтеграцію з ChatGPT (зручно уточнювати промпт у діалозі). MidJourney історично «живе» в Discord, що пришвидшує старт і ком'юніті-взаємодію, але накладає обмеження на робочий процес і зручність керування результатами поза каналами; вебінтерфейс частково це згладжує. Stable Diffusion виділяється тим, що може працювати як у вебформаті, так і локально через різні UI – це робить його найгнучкішим для тих, хто хоче незалежності від хмари. Firefly натомість орієнтований на креативні команди: веб плюс глибокі інтеграції з Creative Cloud, що критично для дизайнерських студій і продакшну.

За контролем якості зображень DALL·E 3 зазвичай дає високу деталізацію і добре «розуміє» текстові описи, проте типово слабші місця – дрібні об'єкти, точна кількість елементів і інколи мікродеталі. MidJourney часто сприймають як еталон «художності»: він сильний у стилізації, атмосфері, фантазійних або абстрактних сценах, коли важлива виразність, а не буквальна технічна точність. Stable Diffusion – це радше «платформа якості», бо діапазон результатів (від фотореалізму до абстракції) визначається вибором моделі, налаштувань і пайплайнів; у вмілих руках він може бути як дуже точним, так і суто творчим. Firefly підкреслено «професійний» у вихідній якості (включно з 2K+), деталізації та відтворюваності результатів, а також добре працює з керованими референсами та структурою, що важливо в комерційному дизайні.

Якщо говорити про гнучкість параметрів і контроль, DALL·E 3 виграє в зручності уточнення: через ChatGPT легко переформулювати задачу, додати вимоги, виправити невдалі місця. Водночас глибокий контроль композиції (наприклад, жорстко «розставити» об'єкти, зафіксувати їх кількість або

геометрію) там більш обмежений. MidJourney дає сильний інструментарій саме промпт-орієнтованого керування: стилі, варіації, інтерактивні перегенерації, що добре для пошуку «правильного настрою» і швидкого арт-дирекшену, але точність «як у технічному ТЗ» може вимагати багато ітерацій. Stable Diffusion у цій категорії найпотужніший: різні UI, контрольні механізми, можливість підключати/навчати власні моделі, автоматизувати процеси й робити локальне редагування – це фактично конструктор для індивідуального production-пайплайна. Firefly, у свою чергу, пропонує детальний контроль композиції та структури через інструменти на кшталт Structural Reference, а також сильні можливості генеративного заповнення й розширення – тобто він дає керованість, близьку до дизайнерської практики «працюю від макета/референса».

Різниця в можливостях управління результатами (зберігання, пошук, організація) добре проявляється в реальних робочих процесах. DALL·E 3 зручний для інтеграцій через API (можна будувати власні каталоги/пошук у своєму продукті), але сам по собі не є «DAM-системою» з бібліотекою, тегами й категоризацією на рівні професійного контент-менеджменту. MidJourney найменше пристосований до структурованого менеджменту, бо багато що зав'язано на Discord-канали й історію чатів: це добре для швидкого експерименту, але гірше для команди, яка хоче стандартизовану бібліотеку ассетів. Stable Diffusion виграє екосистемою надбудов: від організації, збереження, метаданих і семантичного пошуку до інтеграцій із CMS – фактично ви можете «зібрати» потрібний рівень керування під свої процеси. Firefly тут найбільш «корпоративний»: синхронізація з Creative Cloud дозволяє природно вбудовувати генерацію в наявні бібліотеки, проєкти та рутинну обробку зображень.

Найбільш практична відмінність для бізнесу – ліцензія і юридичні ризики. DALL·E 3 загалом допускає комерційне використання, але з низкою політик і обмежень, включно з можливістю авторів виключати свої роботи з навчання (що важливо з етичної точки зору, але не завжди прозоро

інтерпретується в юридичних сценаріях). MidJourney також дає можливість застосування в дизайні, проте в таблиці підкреслено обмеження на масове комерційне використання – це означає, що для масштабних виробничих задач (контент на потік, комерційні бібліотеки, кампанії з великим охопленням) треба уважно читати умови. Stable Diffusion відрізняється відкритістю: код відкритий, можна модифікувати й інтегрувати як завгодно, але «комерційність залежить від моделі» – тобто юридичний статус визначається ліцензіями конкретних чекпойнтів/датасетів і тим, як ви їх використовуєте. Firefly позиціонується як найбезпечніший для бізнесу: навчання на ліцензованих даних Adobe Stock та наявність IP-індемніфікації роблять його привабливим для компаній, де ризик претензій до контенту – критичний.

За швидкістю генерації DALL·E 3 зазвичай працює швидко, але в реальному користуванні впирається в ліміти доступу (наприклад, обмеження кількості генерацій на годину залежно від підписки/режиму). MidJourney часто відчувається повільнішим саме через черги: типові 30–60 секунд і залежність від тарифу – нормально для творчого пошуку, але не завжди для потокового продакшну. Stable Diffusion може бути дуже швидким у локальному режимі, якщо є сильне GPU-залізо, і це часто дає найкращу прогнозованість часу (ви не залежите від хмарної черги), хоча на слабкому обладнанні ситуація буде протилежною. Firefly у таблиці подано як «миттєвий» завдяки багаторівневій оптимізації під бізнес – тобто ставка на стабільну швидкість у хмарному середовищі й роботу в інтегрованому пайплайні.

Для реалізації функціональності генерації зображень у межах розроблюваного додатку було обрано технологію DALL·E 3. Рішення ґрунтується на поєднанні двох ключових критеріїв відбору: якості візуального результату та практичної придатності інструмента до програмної інтеграції в серверну частину системи. По-перше, DALL·E 3 демонструє стабільно високий рівень естетичної якості згенерованих зображень, зокрема виразну деталізацію та загальну візуальну привабливість. Для застосунку це є принциповим, оскільки кінцевий користувацький досвід значною мірою визначається якістю

графічного контенту, а генеративні результати мають відповідати очікуванням щодо «готовності до використання» без суттєвого пост-редагування. По-друге, важливим чинником стала наявність API, що забезпечує можливість прямої інтеграції DALL·E 3 у програмну архітектуру проєкту. API-підхід дозволяє автоматизувати процес генерації, виконувати запити з серверного середовища, керувати доступом та лімітами, а також вбудовувати отримані результати в робочі сценарії додатку (збереження, обробка та подальша доставка клієнтській частині). У підсумку це робить обране рішення технологічно доцільним не лише з позиції якості контенту, але й з погляду масштабованості та підтримуваності системи.

1.4 Аналіз вимог до платформи генерації зображень

Розробка будь-якої програмної системи починається з глибокого аналізу вимог, які визначають функціональність, продуктивність, безпеку та інші критично важливі аспекти майбутнього продукту. Для платформи Pic.AI, орієнтованої на генерацію зображень за допомогою штучного інтелекту, процес формування вимог базувався на комплексному підході, що включав аналіз потреб цільової аудиторії, дослідження існуючих рішень на ринку, технічні обмеження інтеграції з OpenAI API та стратегічні цілі проекту. Вимоги до платформи класифікуються на дві основні категорії: функціональні вимоги, які описують конкретні дії та можливості системи, та нефункціональні вимоги, що визначають якісні характеристики роботи системи, такі як продуктивність, масштабованість, безпека, зручність використання та підтримуваність. Процес збору вимог включав аналіз конкурентних рішень (DALL-E вебверсія, Midjourney Discord-бот, Stable Diffusion UI), вивчення відгуків користувачів цих платформ, оцінку технічних можливостей OpenAI API та визначення мінімально життєздатного продукту (MVP) для першої версії платформи. Ключовим фактором при формуванні вимог стала необхідність створення простого, інтуїтивно зрозумілого інтерфейсу, який би не перевантажував

користувача технічними деталями роботи нейромережі, але водночас надавав достатньо контролю над процесом генерації. Платформа повинна була забезпечувати баланс між функціональністю та простотою, дозволяючи як початківцям, так і досвідченим користувачам ефективно працювати з AI-генерацією зображень.

1.4.1 Функціональні вимоги до системи автентифікації

Система автентифікації є критично важливим компонентом платформи, оскільки вона забезпечує персоналізацію користувацького досвіду, збереження історії генерацій та можливість монетизації в майбутньому через підписки або обмеження на кількість генерацій. Функціональні вимоги до автентифікації включають можливість реєстрації нових користувачів з валідацією введених даних, безпечне зберігання облікових даних, вхід в систему з перевіркою реквізитів, управління сесіями та вихід з автоматичним знищенням сесії. Реєстрація нового користувача повинна включати наступні обов'язкові поля: унікальний username (ім'я користувача), email адреса як основний ідентифікатор та пароль з мінімальними вимогами до складності. Система повинна автоматично перевіряти унікальність email при реєстрації, запобігаючи створенню дублікатів облікових записів. Email адреса має зберігатися в нижньому регістрі (lowercase) для уникнення проблем з чутливістю до регістру при вході.

Вимоги до безпеки паролів включають обов'язкове хешування з використанням криптографічно стійкого алгоритму (bcrypt) з автоматичною генерацією солі для кожного пароля. Система не повинна зберігати паролі у відкритому вигляді ні в базі даних, ні в логах. Фактор складності хешування (salt rounds) має бути достатнім для захисту від brute-force атак, але не надто високим, щоб не створювати затримок при автентифікації – оптимальним значенням визначено 10 раундів, що дає час хешування близько 100-200 мс. Процес входу в систему повинен включати валідацію формату email, перевірку існування користувача з таким email, порівняння хешу введеного пароля з

збереженим хешем та створення серверної сесії при успішній автентифікації. Система повинна повертати однакове повідомлення про помилку незалежно від того, чи не знайдено email, чи невірний пароль, щоб не розкривати інформацію про існуючі акаунти потенційним зловмисникам.

Управління сесіями реалізується через пролміжне ПЗ (програмне забезпечення) express-session з наступними вимогами: унікальний ідентифікатор сесії генерується автоматично і зберігається в cookie на клієнті, дані сесії (userId, username, email) зберігаються на сервері, cookie має бути httpOnly для захисту від XSS-атак, термін життя сесії визначається налаштуваннями (за замовчуванням – до закриття браузера), можливість явного завершення сесії через функцію logout. Вимоги до захисту маршрутів включають функцію проміжного ПЗ isAuthenticated(), яка перевіряє наявність активної сесії перед доступом до захищених ресурсів. Неавторизовані користувачі повинні автоматично перенаправлятися на головну сторінку з можливістю входу. API-ендпоінти, що працюють з персональними даними або генерацією зображень, мають бути обов'язково захищені цим проміжним ПЗ.

1.4.2 Функціональні вимоги до генерації зображень

Основна функціональність платформи – генерація зображень на основі текстових описів (промптів) через інтеграцію з OpenAI DALL-E 3 API. Вимоги до цього функціоналу включають можливість введення текстового промпту довільної довжини (в межах обмежень API – до 4000 символів), відправку запиту до OpenAI API з обраними параметрами, отримання згенерованого зображення, його завантаження на сервер платформи для постійного зберігання та відображення результату користувачу з можливістю завантаження. Технічні параметри генерації визначаються можливостями DALL-E 3 API: модель – dall-e-3 (найновіша версія станом на 2024-2025 рік), кількість зображень за один запит – 1 (обмеження API для DALL-E 3), розмір зображення – 1024x1024 пікселів (стандартний квадратний формат, що забезпечує оптимальний баланс

між якістю та швидкістю генерації), формат виводу – PNG з автоматичним стисненням.

Система повинна автоматично обробляти промпт користувача, передаючи його до API без модифікацій, що дозволяє користувачу повністю контролювати процес генерації. У майбутніх версіях може бути додана функція "покращення промпту" (prompt enhancement), але в базовій версії платформи це не реалізовано для збереження прозорості та передбачуваності результату. Вимоги до обробки відповіді API включають отримання URL тимчасового зображення з серверів OpenAI (термін життя URL – близько 1 години), завантаження зображення за цим URL через HTTP-клієнт (axios), збереження файлу у директорії /uploads на сервері платформи з унікальним ім'ям формату image_{userId}_{timestamp}.png, збереження відносного шляху до файлу в базі даних MongoDB у масиві imageHistory користувача та повернення клієнту шляху до збереженого зображення для відображення. Система повинна забезпечувати індикацію процесу генерації для користувача, оскільки цей процес може тривати від 10 до 60 секунд залежно від складності промпту та завантаженості серверів OpenAI. Під час генерації має відображатися анімований loader, а поле введення та кнопка генерації мають бути заблоковані для запобігання повторних запитів.

Вимоги до обробки помилок включають перехоплення та логування всіх помилок API (недійсний API ключ, перевищення квоти, некоректний промпт, тимчасова недоступність сервісу), повернення клієнту зрозумілого повідомлення про помилку без розкриття технічних деталей, можливість повторної спроби генерації після помилки без втрати введеного промпту.

1.4.3 Функціональні вимоги до управління історією генерацій

Платформа повинна автоматично зберігати історію всіх згенерованих зображень для кожного користувача, надаючи можливість переглядати, завантажувати та видаляти попередні генерації. Це створює персоналізований досвід та дозволяє користувачам будувати портфоліо своїх AI-творінь. Вимоги

до збереження історії включають автоматичне додавання запису про нову генерацію в масив imageHistory користувача при успішній генерації, збереження відносного шляху до файлу зображення (наприклад, /uploads/image_507f1f77bcf86cd799439011_1702905600000.png), збереження точної дати та часу генерації у форматі ISO 8601, автоматичну генерацію унікального ідентифікатора (_id) для кожного запису через Mongoose. Структура запису в історії визначена через Mongoose SubSchema і включає поля: imagePath (String, required) – відносний шлях до файлу зображення, generatedAt (Date, default: Date.now) – дата та час генерації, _id (ObjectId, auto-generated) – унікальний ідентифікатор запису для операцій видалення.

Вимоги до відображення історії включають отримання повного списку генерацій користувача через захищений API endpoint /api/history, сортування записів за датою генерації від новіших до старіших (DESC порядок), відображення мініатюр зображень (60x60 пікселів) для економії місця та швидкого завантаження, відображення дати та часу генерації з урахуванням локалізації користувача (формат залежить від обраної мови – uk-UA або en-US).

Кожен елемент історії повинен мати інтерактивні можливості: клік по зображенню або інформації про дату завантажує зображення в основну область перегляду (512x512 на десктопі, адаптивно масштабується на мобільних пристроях), кнопка завантаження дозволяє зберегти зображення на локальний диск користувача з оригінальним іменем файлу, кнопка видалення ініціює процес видалення зображення з підтвердженням дії.

Функціонал видалення зображень повинен відповідати наступним вимогам: перед видаленням відображається модальне вікно з підтвердженням дії (текст локалізований відповідно до мови інтерфейсу), при підтвердженні надсилається DELETE-запит до /api/history/:imageId з ідентифікатором зображення, сервер перевіряє належність зображення поточному користувачу (захист від несанкціонованого видалення), видаляється запис з бази даних MongoDB, видаляється файл з файлової системи сервера (директорія /uploads), клієнт отримує підтвердження успішного видалення та видаляє елемент з UI з

плавною анімацією. Вимоги до оптимізації роботи з історією включають lazy loading елементів при скроллінгу (для користувачів з великою історією), кешування мініатюр на клієнті для зменшення навантаження на сервер, обмеження висоти контейнера історії (512px) з вертикальним скроллінгом, кастомізований scrollbar для покращення естетики інтерфейсу.

1.4.4 Нефункціональні вимоги: продуктивність та швидкодія

Продуктивність платформи є критично важливим фактором користувацького досвіду, особливо з огляду на специфіку роботи з AI-генерацією, яка сама по собі є тривалим процесом. Система повинна мінімізувати додаткові затримки, не пов'язані з генерацією зображень. Вимоги до швидкодії серверних операцій включають: час відгуку на GET-запити статичних сторінок (/, /app) – не більше 100 мс, час відгуку на API-запити аутентифікації (login, register) – не більше 500 мс з урахуванням хешування bcrypt, час відгуку на запит історії генерацій (/api/history) – не більше 300 мс для користувачів з історією до 100 зображень, час відгуку на запит видалення зображення – не більше 200 мс.

Час генерації зображення визначається зовнішнім API OpenAI і становить зазвичай 10-60 секунд залежно від складності промпту та завантаженості серверів. Платформа не може контролювати цей параметр, але повинна ефективно обробляти очікування: відображати індикатор завантаження, не блокувати інтерфейс, дозволяти користувачу переглядати історію під час генерації. Вимоги до швидкості завантаження клієнтських ресурсів включають: час завантаження головної сторінки (index.html з усіма ресурсами) – не більше 2 секунд на з'єднанні 4G, розмір головної CSS-файлу (styles.css) – не більше 50 КБ, розмір CSS робочої області (app.css) – не більше 40 КБ, розмір JavaScript-файлів (i18n.js, script.js, auth-modal.js) – сумарно не більше 30 КБ, розмір файлів локалізації (en.json, uk.json) – по 5 КБ кожен.

Оптимізація зображень вимагає: автоматичне стиснення PNG-файлів при збереженні на сервері (якщо розмір перевищує 2 МБ), відображення мініатюр в

історії з розміром 60x60 пікселів через CSS transform: scale або background-size: cover, lazy loading зображень в історії при скролі (завантаження тільки видимих елементів), використання CSS-анімацій замість JavaScript для плавних переходів та трансформацій.

Вимоги до оптимізації роботи з базою даних включають: індексування полів email та username в User-колекції для швидкого пошуку при автентифікації, обмеження результатів запиту історії (наприклад, останні 50 генерацій) з можливістю пагінації в майбутньому, використання projection для отримання тільки необхідних полів (.select('email username')), кешування сесій користувачів на рівні express-session для мінімізації запитів до БД (бази даних).

1.4.5 Нефункціональні вимоги: безпека та приватність

Безпека користувацьких даних є пріоритетом для платформи, яка обробляє особисту інформацію (email, паролі) та зберігає творчий контент користувачів. Система повинна відповідати сучасним стандартам безпеки вебдодатків. Вимоги до безпеки автентифікації включають: хешування паролів з використанням bcrypt з мінімальним фактором складності 10 раундів, заборона зберігання паролів у відкритому вигляді в будь-якій частині системи (БД, логи, кеш), автоматична генерація унікальної солі для кожного пароля, валідація складності паролю на стороні клієнта (мінімум 6 символів) та сервера. Вимоги до захисту сесій включають: використання httpOnly cookies для зберігання ідентифікатора сесії (захист від XSS-атак), генерація криптографічно випадкових session ID, зберігання даних сесії виключно на сервері (в пам'яті або Redis в production), автоматичне знищення сесії при logout, можливість налаштування терміну життя сесії (за замовчуванням – до закриття браузера).

Захист від CSRF (Cross-Site Request Forgery) атак реалізується через: перевірку походження запитів через Referer header, використання SameSite атрибута для cookies, проміжне ПЗ для валідації джерела запитів до критичних ендпоінтів (генерація, видалення). Захист від XSS (Cross-Site Scripting) атак включає: автоматичне екранування всіх користувацьких даних при виводі в

HTML, використання textContent замість innerHTML при динамічному оновленні DOM, Content Security Policy заголовки для обмеження виконання скриптів, валідація та санітизація всіх вхідних даних на сервері. Захист від NoSQL Injection атак забезпечується через: використання Mongoose схем з типізацією та валідацією даних, параметризовані запити замість конкатенації рядків, валідація типів даних перед передачею в БД, обмеження розміру вхідних даних.

Вимоги до захисту файлів включають: зберігання згенерованих зображень з унікальними іменами, що включають userId для ізоляції між користувачами, перевірка належності файлу користувачу перед операціями видалення або завантаження, заборона прямого доступу до директорії uploads без автентифікації (через проміжне ПЗ), автоматичне видалення файлів при видаленні запису з БД.

Вимоги до приватності даних включають: зберігання мінімального набору особистої інформації (username, email, password hash), відсутність логування паролів або інших чутливих даних, можливість користувача видалити свій акаунт з повним видаленням всіх даних (right to be forgotten), шифрування з'єднання через HTTPS в production середовищі, захист API ключів OpenAI через змінні середовища (.env файл, який не включається до Git).

1.5 Обгрунтування вибору технологічного стеку

1.5.1 Загальні критерії вибору технологій

При розробці веборієнтованої платформи Pic.AI для генерації зображень з використанням штучного інтелекту постало питання вибору оптимального технологічного стеку, який би забезпечив високу продуктивність, масштабованість, безпеку та зручність розробки. Технологічний стек є фундаментом будь-якого програмного рішення, і його правильний вибір безпосередньо впливає на якість кінцевого продукту, швидкість розробки, вартість підтримки та можливості подальшого розвитку системи.

Основними критеріями при виборі технологій стали: продуктивність обробки асинхронних запитів до зовнішніх API, наявність розвиненої екосистеми бібліотек та інструментів, підтримка активної спільноти розробників, можливість горизонтального масштабування, безпека обробки користувацьких даних, а також простота інтеграції з сучасними AI-сервісами, зокрема OpenAI API. Враховуючи специфіку платформи, яка передбачає інтенсивну роботу з генерацією та зберіганням медіа-контенту, необхідність управління сесіями користувачів та обробку великої кількості одночасних запитів, вибір технологій здійснювався з урахуванням балансу між функціональністю та ефективністю використання ресурсів.

1.5.2 Вибір серверного середовища NodeJS

Для реалізації серверної частини платформи було обрано Node.js версії 20+ як середовище запуску коду [10]. Node.js є платформою для виконання JavaScript-коду на серверній стороні, побудованою на JavaScript-движку V8 від Google Chrome. Ключовою перевагою Node.js є його асинхронна, подієво-орієнтована архітектура, яка ідеально підходить для обробки великої кількості одночасних з'єднань без блокування потоку виконання. Основним аргументом на користь Node.js стала його висока ефективність при роботі з асинхронними операціями введення-виведення, що критично важливо для платформи Pic.AI. Генерація зображень через OpenAI API є операцією, яка може тривати від кількох секунд до хвилини, залежно від складності промпту та завантаженості серверів OpenAI. Неблокуюча архітектура Node.js дозволяє обробляти множину таких запитів одночасно без заморожування сервера, забезпечуючи високу пропускну здатність системи. Додатковою перевагою стала можливість використання єдиної мови програмування (JavaScript) як на клієнті, так і на сервері, що спрощує розробку, знижує поріг входу для розробників та дозволяє повторно використовувати код між фронтендом і бекендом. Node.js має потужну екосистему пакетів через npm (Node Package Manager), що налічує

понад мільйон готових модулів для вирішення різноманітних задач – від роботи з базами даних до інтеграції з зовнішніми API.

Важливим фактором стала також наявність офіційного SDK від OpenAI для Node.js (версія 4.40.2), що значно спрощує інтеграцію з DALL-E 3 API, надає типізовані інтерфейси та обробку помилок out-of-the-box. Крім того, Node.js має відмінну підтримку для роботи з потоками даних (streams), що використовується при завантаженні згенерованих зображень з серверів OpenAI на локальне сховище платформи.

1.5.3 Вибір вебфреймворку: ExpressJS

Для побудови RESTful API та управління HTTP-запитами було обрано фреймворк Express.js версії 4.19.2 – найпопулярніший та найшвидший мінімалістичний вебфреймворк для Node.js. Express.js надає тонкий шар абстракції над стандартним HTTP-модулем Node.js, залишаючись при цьому гнучким та не нав'язуючи жорсткої структури додатку.

Основною перевагою Express.js є його архітектура з проміжним ПЗ, яка дозволяє будувати додаток як ланцюжок обробників запитів. У контексті платформи Pic.AI це використовується для реалізації автентифікації (проміжне ПЗ isAuthenticated), парсингу JSON та URL-encoded даних, обслуговування статичних файлів, управління сесіями та обробки помилок. Кожне проміжне ПЗ має доступ до об'єктів запиту (req), відповіді (res) та функції next() для передачі управління наступному обробнику, що дозволяє розділяти логіку додатку на окремі модулі. Express.js надає зручний API для визначення маршрутів (routing) з підтримкою параметрів, query-рядків та різних HTTP-методів (GET, POST, PUT, DELETE). У Pic.AI реалізовано наступні основні маршрути: GET / (головна сторінка), GET /app (робоча область), POST /register (реєстрація), POST /login (вхід), POST /generateImage (генерація зображення), GET /api/user (дані користувача), GET /api/history (історія генерацій), DELETE /api/history/:imageId (видалення зображення) та GET /logout (вихід з системи). Фреймворк має мінімальні накладні витрати на продуктивність, що важливо

для високонавантажених додатків. За результатами бенчмарків, Express.js здатний обробляти десятки тисяч запитів за секунду на сучасному обладнанні. Крім того, Express.js має величезну екосистему пакетів проміжного ПЗ для вирішення типових задач: express-session для управління сесіями, cors для налаштування Cross-Origin Resource Sharing, підвищення безпеки HTTP-заголовків, morgan для логування запитів тощо.

1.5.4 Вибір системи управління базою даних: MongoDB

Для зберігання користувацьких даних та історії генерацій було обрано NoSQL базу даних MongoDB у хмарному варіанті MongoDB Atlas. MongoDB є документо-орієнтованою базою даних, яка зберігає дані у форматі BSON (Binary JSON) – бінарному представленні JSON-подібних документів. Вибір MongoDB був обумовлений кількома ключовими факторами, специфічними для архітектури платформи Pic.AI.

По-перше, схема даних користувача природно відображається в документну модель MongoDB. Кожен користувач представлений одним документом, що містить основні поля (username, email, password) та вкладений масив об'єктів imageHistory, який зберігає історію згенерованих зображень. Така денормалізована структура дозволяє отримувати всю необхідну інформацію про користувача та його історію одним запитом без необхідності JOIN-операцій, характерних для реляційних баз даних. Це суттєво підвищує швидкість читання даних.

По-друге, MongoDB має чудову інтеграцію з Node.js через бібліотеку Mongoose версії 9.0.0 – ODM (Object Document Mapper), що надає схемну валідацію, типізацію, «хуків» проміжного ПЗ та зручний API для роботи з даними. Mongoose дозволяє описувати структуру документів через схеми (schemas), автоматично валідувати дані перед збереженням, визначати віртуальні поля, методи моделей та використовувати pre/post «хуки» для додаткової логіки (наприклад, хешування паролів перед збереженням).

У Pic.AI визначено дві схеми: userSchema з полями username (String, required, unique), email (String, required, unique, lowercase), password (String, required), imageHistory (Array), createdAt (Date) та вкладена imageHistorySchema з полями imagePath (String, required), generatedAt (Date, default: Date.now). Mongoose автоматично створює індекси для полів, позначених як unique, що забезпечує швидкий пошук користувачів за email та запобігає дублюванню.

По-третє, гнучкість схеми MongoDB дозволяє легко розширювати структуру даних у майбутньому без необхідності міграцій та ALTER TABLE операцій. Наприклад, можна додати нові поля для профілю користувача (аватар, налаштування приватності), додаткові метадані до історії зображень (теги, категорії, кількість переглядів) або нові колекції для функцій соціальної взаємодії (коментарі, лайки, шеринг).

По-четверте, MongoDB Atlas – повністю керований хмарний сервіс – надає автоматичне резервне копіювання, моніторинг продуктивності, можливості масштабування (vertical та horizontal scaling через sharding), вбудовану систему безпеки з шифруванням даних у спокої та під час передачі, а також безкоштовний тарифний план (512 MB сховища), достатній для тестування та невеликих проектів.

1.5.5 Вибір бібліотеки для роботи з OpenAI: OpenAI SDK

Для інтеграції з сервісами OpenAI було обрано офіційну бібліотеку openai версії 4.40.2. Це офіційний SDK від OpenAI для JavaScript/TypeScript, який надає зручний інтерфейс для взаємодії з усіма API OpenAI, включаючи DALL-E 2/3, GPT-3.5/4, Whisper, Embeddings та інші. Основною перевагою офіційного SDK є абстракція над HTTP-запитами та автоматична обробка помилок API. Бібліотека приховує деталі формування запитів, автентифікації через API-ключ, обробку лімітів запитів та парсинг відповідей. Розробнику достатньо викликати метод openai.images.generate() з параметрами model, prompt, n (кількість зображень) та size (розмір зображення), а SDK автоматично сформує правильний HTTP POST запит до https://api.openai.com/v1/images/generations.

У Pic.AI використовується модель DALL-E 3 з розміром зображень 1024x1024 пікселів. DALL-E 3 є найновішою версією моделі генерації зображень від OpenAI станом на 2024 рік, яка демонструє значно покращену якість, деталізацію та відповідність текстовим промптам порівняно з попередніми версіями. Модель здатна генерувати високоякісні, фотореалістичні зображення, художні ілюстрації, абстрактні композиції та інші стилі на основі природномовних описів. Бібліотека також надає TypeScript типи для всіх параметрів та відповідей API, що підвищує безпеку коду та полегшує розробку завдяки автодоповненню в IDE. Крім того, SDK автоматично обробляє pagination для запитів, що повертають великі обсяги даних, та надає streaming API для моделей чату, що може бути корисним при розширенні функціоналу платформи.

1.5.6 Система автентифікації та управління сесіями

Для реалізації безпечної автентифікації користувачів було обрано комбінацію двох бібліотек: bcrypt версії 6.0.0 для хешування паролів та express-session версії 1.18.2 для управління сесіями.

Bcrypt є криптографічною хеш-функцією, спеціально розробленою для безпечного зберігання паролів. На відміну від швидких хеш-функцій (MD5, SHA-1), bcrypt є повільним за дизайном, що робить brute-force атаки економічно недоцільними. Бібліотека автоматично генерує сіль (salt) для кожного пароля та використовує алгоритм Blowfish для створення унікального хешу. У Pic.AI використовується фактор складності (salt rounds) рівний 10, що означає 2^10 = 1024 ітерацій хешування. При реєстрації пароль хешується через bcrypt.hash(password, 10), а при вході перевіряється через bcrypt.compare(password, hashedPassword).

Express-session надає проміжне ПЗ для управління HTTP-сесіями користувачів. При успішному вході створюється серверна сесія з унікальним ідентифікатором, який зберігається у куки на клієнті. На сервері в об'єкті сесії зберігаються дані користувача (req.session.userId, req.session.username,

req.session.email), які використовуються для ідентифікації при подальших запитах. Конфігурація express-session включає секретний ключ для підпису cookies, налаштування перезбереження: false (не перезберігати незмінені сесії) та saveUninitialized: false (не зберігати порожні сесії). Для захисту маршрутів створено проміжне ПЗ isAuthenticated, який перевіряє наявність req.session.userId і редиректить неавторизованих користувачів на головну сторінку. Це забезпечує захист робочої області /app та API-ендпоінтів від несанкціонованого доступу.

1.5.7 Система інтернаціоналізації: Custom i18n Module

Для реалізації багатомовності (українська/англійська) було розроблено власний легковаговий модуль i18n.js замість використання важких бібліотек (i18next, react-intl). Модуль завантажує JSON-файли з перекладами (en.json, uk.json), надає метод t(key) для отримання перекладів та метод setLanguage(lang) для зміни мови з автоматичним оновленням вмісту сторінки. Переклади організовані у вигляді вкладених об'єктів з логічним групуванням за секціями (nav, home, aboutUs, auth, app). Для оновлення тексту елементів використовуються data-атрибути: data-i18n="nav.home" для текстового вмісту та data-i18n-placeholder="auth.email" для placeholder атрибутів. При зміні мови модуль проходиться по всіх елементах з цими атрибутами та оновлює їх вміст. Вибрана мова зберігається в localStorage, що дозволяє запам'ятовувати налаштування користувача між сесіями. Модуль також викидає кастомну подію languageChanged, яка використовується для оновлення динамічно генерованого контенту (наприклад, форматування дат в історії генерацій відповідно до локалі).

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПЛАТФОРМИ PIC.AI

2.1 Концептуальна модель вебплатформи

Концептуальна модель системи Pic.AI представляє собою високорівневу архітектурну схему, яка визначає основні компоненти платформи, їх взаємозв'язки та потоки даних між ними. Ця модель є фундаментом для розуміння того, як система функціонує в цілому, та служить відправною точкою для детального проєктування окремих модулів. Концептуальна модель базується на класичній тришаровій архітектурі вебдодатків: рівень представлення (Presentation Layer), рівень додатку (Application Layer) та рівень даних (Data Layer), доповнених інтеграцією із зовнішнім сервісом OpenAI API. Платформа Pic.AI є розподіленою системою, де клієнтська частина виконується в браузері користувача, серверна частина працює на Node.js, база даних розміщена в хмарному сервісі MongoDB Atlas, а генерація зображень делегується зовнішньому API OpenAI. Така архітектура забезпечує розділення відповідальності між компонентами, спрощує масштабування та підтримку, а також дозволяє незалежно розвивати окремі частини системи без впливу на інші модулі.

Ключовою особливістю концептуальної моделі є асинхронна взаємодія між компонентами. Оскільки генерація зображень через OpenAI API є тривалим процесом (10-60 секунд), система побудована з урахуванням неблокуючої обробки запитів. Node.js із його подієвою архітектурою ідеально підходить для цього завдання, дозволяючи обслуговувати множину одночасних користувачів без затримок або блокувань.

2.1.1 Рівень представлення (Presentation Layer)

Рівень представлення є точкою входу користувача до системи та відповідає за відображення інформації і збір користувацького вводу. Цей рівень складається з двох основних HTML-сторінок: index.html (лендінг для

незареєстрованих користувачів) та app.html (робоча область для автентифікованих користувачів). Обидві сторінки побудовані на стандартних вебтехнологіях HTML5, CSS3 та JavaScript ES6+ без використання важких фронтенд-фреймворків, що забезпечує швидке завантаження та мінімальну складність.

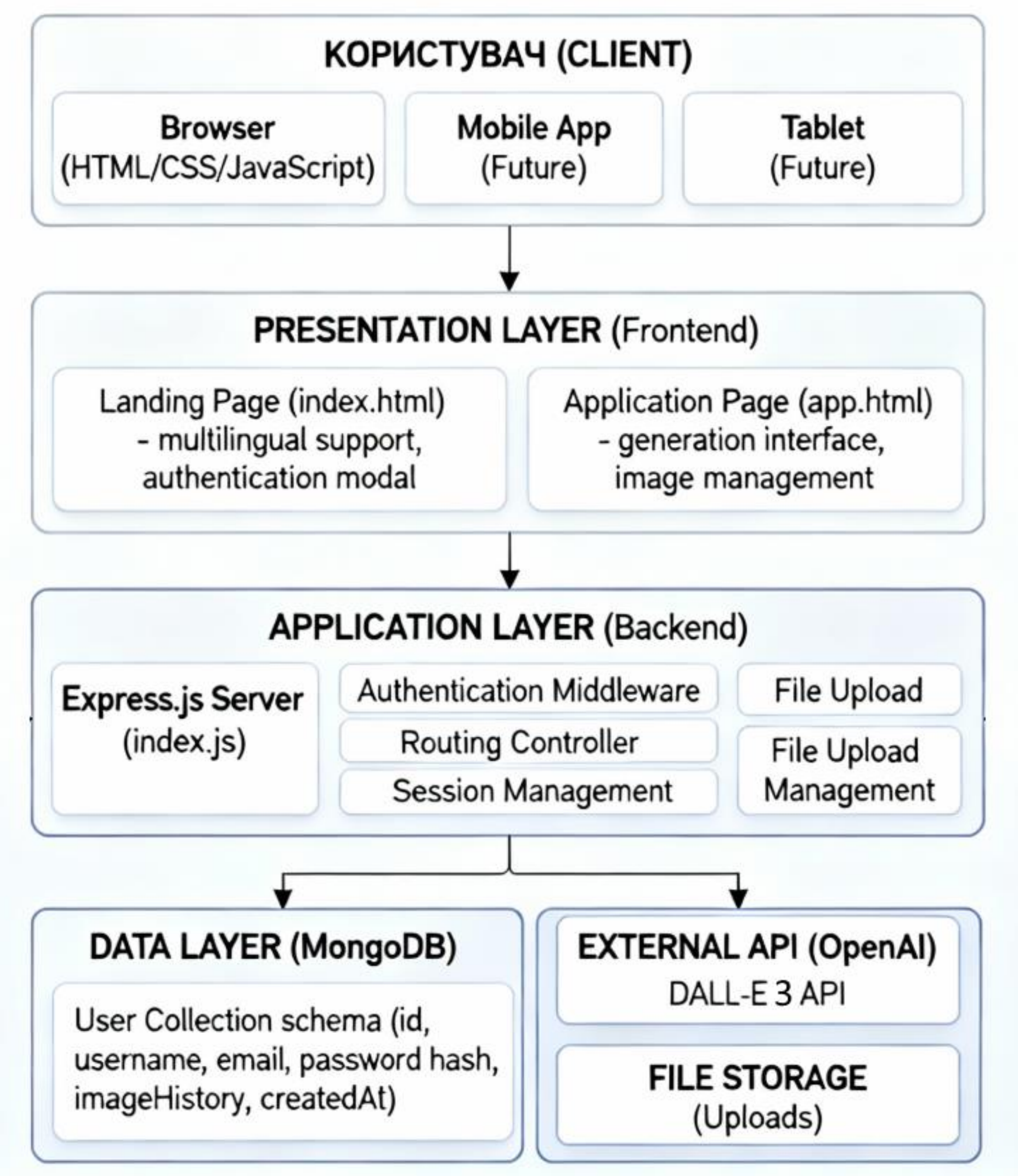


Рисунок 2.1 – Концептуальна архітектурна схема вебплатформи Pic AI

Головна сторінка index.html містить кілька ключових секцій: навігаційний header із меню та селектором мови, секцію Home з основним закликом до дії

(hero section), секції About Us, How It Works, Examples, Contacts та Footer з контактною інформацією. Особливістю головної сторінки є модальне вікно автентифікації, яке дозволяє користувачам зареєструватися або увійти в систему без переходу на окрему сторінку. Модальне вікно містить дві форми з табами: Login (вхід) та Register (реєстрація), які перемикаються за допомогою JavaScript. Робоча сторінка app.html є основним інтерфейсом для генерації зображень та управління історією. Вона включає textarea для введення промпту, кнопку генерації, область відображення згенерованого зображення (512x512 пікселів на десктопі, адаптивна на мобільних), та панель історії генерацій з можливістю перегляду, завантаження та видалення попередніх результатів. Header робочої області містить логотип з посиланням на головну сторінку, селектор мови та користувацьке меню з випадаючим списком для виходу з системи. Клієнтська частина використовує модульну структуру JavaScript-файлів: i18n.js для системи багатомовності, script.js для основного функціоналу генерації та управління історією, auth-modal.js для обробки автентифікації, scroll.js для плавної прокрутки між секціями та burger.js для мобільного меню. Такий розподіл дозволяє легко підтримувати код та розширювати функціонал без конфліктів між модулями.

Взаємодія з серверною частиною здійснюється через Fetch API з використанням асинхронних функцій async/await. Усі запити використовують JSON формат для обміну даними, що спрощує парсинг на обох сторонах. Клієнт надсилає POST-запити для генерації зображень, автентифікації та реєстрації, GET-запити для отримання даних користувача та історії генерацій, DELETE-запити для видалення зображень. Кожен запит супроводжується обробкою помилок через try/catch блоки та відображенням відповідного feedback користувачу (loader під час обробки, повідомлення про успіх чи помилку).

Система багатомовності реалізована через кастомний модуль I18n, який завантажує JSON-файли з перекладами (en.json, uk.json) при ініціалізації сторінки та надає методи для отримання перекладів t(key) та зміни мови setLanguage(lang). Переклади організовані у вигляді вкладених об'єктів за

логічними секціями: nav, header, home, aboutUs, howItWorks, examples, contacts, footer, auth, app. При зміні мови модуль автоматично оновлює текст усіх елементів з data-атрибутами data-i18n та data-i18n-placeholder, а також зберігає вибір користувача в localStorage для наступних візитів.

2.1.2 Рівень додатку (Application Layer)

Рівень додатку є серцем системи та реалізує всю бізнес-логіку платформи. Він побудований на фреймворку Express.js, який працює поверх середовища Node.js. Основний файл index.js містить конфігурацію сервера, визначення маршрутів, проміжне ПЗ та логіку обробки запитів. Сервер слухає порт, визначений у змінній оточення PORT (за замовчуванням 5000), та обробляє HTTP-запити від клієнтів. Архітектура серверної частини базується на проміжному ПЗ Express.js. При отриманні запиту він проходить через серію функцій проміжного ПЗ: express.json() для парсингу JSON-тіла запитів, express.urlencoded() для обробки форм, express.static() для обслуговування статичних файлів (CSS, JS, зображення), express-session для управління сесіями користувачів та кастомний проміжне ПЗ isAuthenticated() для захисту приватних маршрутів. Така архітектура дозволяє гнучко композувати обробку запитів та повторно використовувати логіку.

Маршрутизація в системі організована за RESTful принципами. Публічні маршрути включають: GET / для головної сторінки, POST /register для реєстрації нових користувачів, POST /login для входу в систему. Захищені маршрути, доступні тільки автентифікованим користувачам: GET /app для робочої області, GET /api/user для отримання даних поточного користувача, GET /api/history для отримання історії генерацій, POST /generateImage для запуску генерації зображення, DELETE /api/history/:imageId для видалення зображення з історії, GET /logout для виходу з системи. Всі API-ендпоінти повертають JSON-відповіді з полями success (boolean) та message або data (об'єкт з даними).

Система автентифікації базується на сесіях через проміжне ПЗ express-session. При успішному вході створюється серверна сесія з унікальним ідентифікатором, який зберігається в cookie на клієнті (httpOnly для захисту від XSS). В об'єкті сесії req.session зберігаються дані користувача: userId (ObjectId з MongoDB), username та email. Проміжне ПЗ isAuthenticated перевіряє наявність req.session.userId перед доступом до захищених маршрутів та редиректить неавторизованих користувачів на головну сторінку.

Паролі користувачів хешуються за допомогою бібліотеки bcrypt з фактором складності 10 раундів (2^10 = 1024 ітерацій). Це робить brute-force атаки на паролі вкрай повільними та економічно недоцільними. При реєстрації пароль проходить через bcrypt.hash(password, 10), що генерує унікальний хеш з автоматично згенерованою сіллю. При вході пароль перевіряється через bcrypt.compare(password, hashedPassword), який повертає boolean результат порівняння. Хеші ніколи не розшифровуються – вони тільки порівнюються.

Логіка генерації зображень реалізована в асинхронній функції generateImage, яка приймає промпт від користувача, надсилає запит до OpenAI API через офіційний SDK, отримує URL згенерованого зображення, завантажує його на сервер через axios, зберігає файл у директорії uploads з унікальним ім'ям image_{userId}_{timestamp}.png, додає запис про генерацію в історію користувача в MongoDB та повертає клієнту відносний шлях до збереженого файлу. Весь процес обгорнутий в try/catch блок для обробки помилок API, проблем з мережею або файловою системою. Функція завантаження зображення downloadImage(url, filepath) використовує потоковий підхід (streams) для ефективного перенесення даних. Вона створює HTTP GET-запит через axios з responseType: 'stream', що повертає ReadableStream замість буфера в пам'яті. Цей потік pipe-иться в WritableStream створений через fs.createWriteStream(filepath), який записує дані безпосередньо в файл по мірі їх надходження. Така техніка дозволяє завантажувати великі файли без завантаження їх повністю в пам'ять сервера, що критично важливо при високому навантаженні.

Обробка помилок в серверній частині організована на кількох рівнях. На рівні маршрутів всі асинхронні операції обгорнуті в try/catch блоки, які перехоплюють помилки MongoDB, OpenAI API, файлової системи тощо. Помилки логуються в консоль через console.error() з контекстною інформацією, а клієнту повертається зрозуміле повідомлення без розкриття технічних деталей. HTTP статус-коди використовуються коректно: 200 для успіху, 201 для створення ресурсу, 400 для валідаційних помилок, 401 для проблем автентифікації, 404 для ненайдених ресурсів, 500 для серверних помилок.

2.1.3 Рівень даних (Data Layer)

Рівень даних відповідає за довготривале зберігання інформації про користувачів, їх облікові дані та історію генерацій. Для реалізації цього рівня використовується NoSQL база даних MongoDB у хмарному варіанті MongoDB Atlas. Вибір MongoDB обумовлений її документо-орієнтованою моделлю даних, яка природно відображає структуру користувача з вкладеним масивом історії генерацій, відсутністю необхідності в JOIN-операціях, горизонтальною масштабованістю через sharding та чудовою інтеграцією з Node.js через Mongoose ODM.

Підключення до бази даних здійснюється через Mongoose бібліотеку методом mongoose.connect(process.env.MONGODB_URI), де URI зберігається в змінній оточення для безпеки. Рядок підключення має формат mongodb+srv://username:password@cluster.mongodb.net/dbname та містить автентифікаційні дані, адресу кластера та параметри з'єднання. При успішному підключенні виводиться повідомлення 'connected', при помилці – об'єкт помилки. Mongoose автоматично управляє пулом з'єднань, підтримує reconnection при втраті зв'язку та надає event emitters для моніторингу стану з'єднання.

Структура даних визначена через Mongoose схеми в файлі user.js. Основна схема userSchema містить поля: username (String, required, unique, trim) – ім'я користувача для відображення в інтерфейсі, email (String, required, unique,

lowercase, trim) – email як основний ідентифікатор для входу, password (String, required) – хешований bcrypt пароль, imageHistory – масив об'єктів історії генерацій, createdAt (Date, default: Date.now) – дата створення акаунту. Опція timestamps: true автоматично додає поля createdAt та updatedAt з часом створення та останнього оновлення документа.

Вкладена схема imageHistorySchema описує структуру одного запису історії: imagePath (String, required) – відносний шлях до файлу зображення (наприклад, /uploads/image_507f1f77bcf86cd799439011_1702905600000.png), generatedAt (Date, default: Date.now, required) – точна дата та час генерації зображення, _id (ObjectId, автоматично генерується Mongoose) – унікальний ідентифікатор запису для операцій видалення. Опція _id: true явно вказує на створення ідентифікатора для кожного піддокумента, що дозволяє звертатися до конкретних елементів масиву.

Mongoose автоматично створює індекси для полів, позначених як unique. Для User колекції створюються compound індекси на username та email, що забезпечує унікальність цих полів та швидкий пошук при автентифікації. Пошук користувача за email виконується через User.findOne({ email: email.toLowerCase() }), який використовує індекс і працює за O(log n) часової складності. Без індексу довелося б сканувати всю колекцію (O(n)), що неприйнятно при зростанні кількості користувачів.

CRUD операції з базою даних виконуються через методи Mongoose моделі. Створення користувача: new User({ username, email, password }).save() – валідує дані згідно схеми та вставляє документ в колекцію. Читання користувача: User.findById(userId) або User.findOne({ email }) – знаходить документ за _id або іншими полями. Оновлення користувача: user.imageHistory.push({ imagePath, generatedAt }); user.save() – додає новий елемент в масив та зберігає зміни. Видалення елементу з історії: user.imageHistory.splice(index, 1); user.save() – видаляє елемент за індексом та оновлює документ.

Транзакційність операцій забезпечується на рівні окремих документів – MongoDB гарантує атомарність операцій над одним документом. Це означає, що операції user.save() або будуть виконані повністю, або не будуть виконані взагалі, без проміжних станів. Для операцій, що торкаються кількох документів (у складніших системах), MongoDB підтримує мультидокументні транзакції через сессії, але в Pic.AI це не потрібно, оскільки вся історія користувача зберігається в одному документі.

Оптимізація запитів досягається через використання «projection» – вибіркове отримання тільки необхідних полів. Наприклад, User.findById(userId).select('email username') повертає тільки email та username без пароля та історії, що економить bandwidth та час обробки. Для отримання історії використовується User.findById(userId).select('imageHistory'), яке вибирає тільки масив imageHistory. Сортування історії на сервері виконується через JavaScript метод sort((a, b) => new Date(b.generatedAt) - new Date(a.generatedAt)), що впорядковує записи від нових до старих.

2.1.4 Потоки даних та взаємодія компонентів

Повний життєвий цикл генерації зображення включає наступні кроки та взаємодії між компонентами:

Крок 1: Ініціація користувачем. Користувач вводить текстовий промпт в textarea на сторінці app.html та натискає кнопку "Start!". JavaScript обробник події submit перехоплює подію, отримує значення промпту з поля введення, валідує його наявність (порожній промпт не дозволяється) та вмикає індикатор завантаження (loader) для візуального feedback. Поле введення та кнопка блокуються для запобігання повторних запитів під час обробки.

Крок 2: Відправка запиту на сервер. Клієнт формує POST-запит до ендпоінту /generateImage з JSON-тілом { prompt: "user's prompt text" } через Fetch API. Запит автоматично включає session cookie, яка містить ідентифікатор сесії користувача для автентифікації. Заголовок Content-Type встановлюється в application/json для правильного парсингу тіла на сервері.

Крок 3: Автентифікація та валідація. Запит проходить через ланцюжок проміжного ПЗ Express: express.json() парсить JSON-тіло в req.body, express-session відновлює об'єкт сесії з store або пам'яті за session ID з cookie, проміжне ПЗ isAuthenticated перевіряє наявність req.session.userId та блокує доступ неавторизованих користувачів. Якщо автентифікація успішна, управління передається в handler функцію generateImage.

Крок 4: Виклик OpenAI API. Серверна функція generateImage отримує промпт з req.body.prompt та userId з req.session.userId. Формується запит до OpenAI через SDK: openai.images.generate({ model: 'dall-e-3', prompt, n: 1, size: "1024x1024" }). Цей виклик є асинхронним та повертає Promise, яка резолвиться після успішної генерації або реджектиться при помилці. Типовий час очікування становить 10-60 секунд залежно від складності промпту та завантаженості серверів OpenAI.

Крок 5: Завантаження зображення. Після отримання відповіді від OpenAI з URL-адресою згенерованого зображення сервер ініціює завантаження файлу через функцію downloadImage(url, filepath). Ця функція використовує axios для GET-запиту з responseType: 'stream', що повертає ReadableStream зображення. Потік передається у WritableStream створений через fs.createWriteStream(), який записує байти безпосередньо в файл на диску в директорії uploads. Повний шлях до файла формується як /uploads/image_{userId}_{timestamp}.png, де timestamp – поточний час в мілісекундах.

Крок 6: Збереження в базі даних. Після успішного збереження файлу на диску створюється запис в історії користувача. Сервер знаходить документ користувача за userId через User.findById(userId), додає новий об'єкт в масив imageHistory методом push({ imagePath: relativePath, generatedAt: new Date() }) та зберігає зміни через user.save(). Mongoose автоматично валідує дані згідно схеми, генерує _id для нового піддокумента та оновлює поле updatedAt документа.

Крок 7: Відповідь клієнту. Після успішного збереження в БД сервер формує JSON-відповідь { success: true, data: relativePath, message: "Image successfully generated and saved" } з HTTP статусом 200. Відносний шлях (наприклад, /uploads/image_507f1f77bcf86cd799439011_1702905600000.png) дозволяє клієнту відразу відобразити зображення через URL без додаткових запитів. Absolute URL не використовується для гнучкості при зміні домену або переході на CDN.

Крок 8: Відображення результату. Клієнт отримує відповідь, парсить JSON, витягує imagePath з data.data та встановлює його як background-image елементу. Зображення завантажується браузером через окремий GET-запит до /uploads/..., який обслуговується проміжне ПЗ Express static. Loader вимикається, поле введення та кнопка розблоковуються, викликається функція loadHistory() для оновлення панелі історії з новим зображенням.

Крок 9: Оновлення історії. Панель історії перезавантажується через GET-запит до /api/history, який повертає масив всіх зображень користувача відсортованих за датою. Клієнт очищує контейнер історії та генерує DOM-елементи для кожного запису з мініатюрою (60x60px), датою, часом та кнопками завантаження/видалення. Кожен елемент має event listeners для кліку (завантаження в основну область перегляду), кнопки download (відкриття зображення в новій вкладці для збереження) та кнопки delete (видалення з підтвердженням).

Альтернативний потік при помилках: якщо на будь-якому кроці виникає помилка (недоступність OpenAI API, проблеми з мережею, помилки файлової системи, помилки БД), вона перехоплюється try/catch блоком в generateImage функції. Деталі помилки логуються на сервері через console.error(), а клієнту повертається JSON-відповідь { success: false, error: "The image could not be generated" } з відповідним HTTP статусом (400 для клієнтських помилок, 500 для серверних). Клієнт відображає повідомлення про помилку користувачу, вимикає loader та дозволяє повторну спробу без перезавантаження сторінки.

2.2 Взаємодія компонентів системи

Вебдодаток Pic.AI побудований за класичною тришаровою клієнт-серверною архітектурою з чітким розділенням відповідальності між компонентами. Система складається з фронтенд-частини (клієнтський браузер), бекенд-частини (Node.js/Express сервер) [10], рівня даних (MongoDB Atlas) та інтеграції із зовнішнім AI-сервісом (OpenAI DALL-E 3 API). Така організація забезпечує модульність, масштабованість та можливість незалежного розвитку окремих компонентів без впливу на загальну функціональність системи.

Фронтенд-частина складається з двох основних HTML-сторінок та набору JavaScript-модулів. Головна сторінка index.html містить лендінг із секціями Home, About Us, How It Works, Examples, Contacts та модальне вікно автентифікації з двома формами (Login/Register). Робоча сторінка app.html надає інтерфейс для генерації зображень з текстовим полем для промпту, областю відображення результату та бічною панеллю історії генерацій. JavaScript-модулі включають i18n.js для системи багатомовності (англійська/українська), script.js для основної логіки генерації та управління історією, auth-modal.js для обробки автентифікації, scroll.js для плавної прокрутки та burger.js для мобільного меню. Взаємодія з сервером здійснюється через Fetch API з асинхронними запитами у форматі JSON.

Серверна частина реалізована на Express.js [11] framework і організована навколо системи маршрутизації та проміжного ПЗ. Основний файл index.js містить конфігурацію сервера, визначення маршрутів та бізнес-логіку. Ланцюжок проміжного ПЗ включає express.json() для парсингу JSON, express.static() для статичних файлів, express-session для управління сесіями користувачів та кастомний isAuthenticated для захисту приватних маршрутів. Публічні маршрути (GET /, POST /register, POST /login) доступні всім, тоді як захищені маршрути (GET /app, POST /generateImage, GET /api/history, DELETE /api/history/:imageId) вимагають активної сесії. Система автентифікації

базується на bcrypt для хешування паролів (10 раундів) та express-session для збереження стану користувача на сервері з httpOnly cookies на клієнті.

Ключовий функціональний компонент системи – модуль генерації зображень через OpenAI API. Асинхронна функція generateImage() приймає текстовий промпт від користувача, формує запит до OpenAI DALL-E 3 API з параметрами (model: 'dall-e-3', size: "1024x1024", n: 1), отримує URL згенерованого зображення, завантажує його на сервер через axios з потоковою передачею (responseType: 'stream'), зберігає файл у директорії uploads з унікальним ім'ям image_{userId}_{timestamp}.png, додає запис у історію користувача в MongoDB та повертає клієнту відносний шлях до збереженого файлу. Весь процес обгорнутий у try/catch блок для обробки помилок API, мережевих збоїв або проблем файлової системи.

Рівень даних представлений NoSQL базою даних MongoDB Atlas та локальним файловим сховищем. MongoDB зберігає колекцію Users з документами, що містять поля username, email, password (bcrypt hash), imageHistory (масив піддокументів) та timestamps. Mongoose ODM надає схемну валідацію через userSchema та вкладену imageHistorySchema з полями imagePath (String), generatedAt (Date) та автогенерованим _id (ObjectId). Індекси створюються автоматично для унікальних полів email та username, забезпечуючи швидкий пошук O(log n). Файлове сховище організоване як директорія uploads на сервері, доступна через проміжне ПЗ Express static, де кожне зображення має унікальне ім'я з userId для ізоляції між користувачами.

Система інтернаціоналізації реалізована через кастомний JavaScript-модуль I18n, який завантажує JSON-файли перекладів (en.json, uk.json) при ініціалізації сторінки. Модуль надає методи t(key) для отримання перекладів за ключем з підтримкою вкладеної нотації (dot notation), setLanguage(lang) для зміни мови з автоматичним оновленням DOM та getCurrentLanguage() для перевірки активної локалі. Переклади організовані у вигляді вкладених об'єктів за логічними секціями (nav, header, home, aboutUs, auth, app). HTML-елементи позначаються data-атрибутами data-i18n для текстового вмісту та data-i18n-

placeholder для placeholder атрибутів. При зміні мови модуль автоматично проходиться по всіх елементах з цими атрибутами, оновлює їх вміст та зберігає вибір користувача в localStorage для наступних візитів.

Компонент історії генерацій включає серверну та клієнтську частини. На сервері endpoint GET /api/history повертає відсортований масив всіх зображень користувача з MongoDB (сортування за generatedAt від нових до старих). Клієнтський JavaScript динамічно генерує DOM-структуру для кожного елемента історії з мініатюрою зображення (60x60px), датою/часом генерації (локалізовані формати через toLocaleDateString/toLocaleTimeString), кнопками завантаження та видалення. При кліку на зображення воно завантажується в основну область перегляду (512x512px на десктопі). Функція deleteImage() надсилає DELETE-запит до /api/history/:imageId, видаляє файл з диску через fs.unlinkSync(), видаляє запис з MongoDB через user.imageHistory.splice() та оновлює UI з плавною анімацією зникнення елемента.

Інтеграція з OpenAI API реалізована через офіційний SDK (openai v4.40.2), ініціалізований з API ключем з .env файлу. Клієнт OpenAI надає метод openai.images.generate() для виклику DALL-E 3 з параметрами model, prompt, n та size. Функція downloadImage(url, filepath) використовує axios для завантаження згенерованого зображення з тимчасового URL через потоковий підхід (streams): створює GET-запит з responseType: 'stream', pipe-ить ReadableStream у WritableStream (fs.createWriteStream), що дозволяє ефективно обробляти великі файли без завантаження в пам'ять. Така архітектура (рис. 2.2) забезпечує надійну інтеграцію з зовнішнім AI-сервісом з обробкою помилок API, rate limiting та graceful degradation при тимчасовій недоступності.

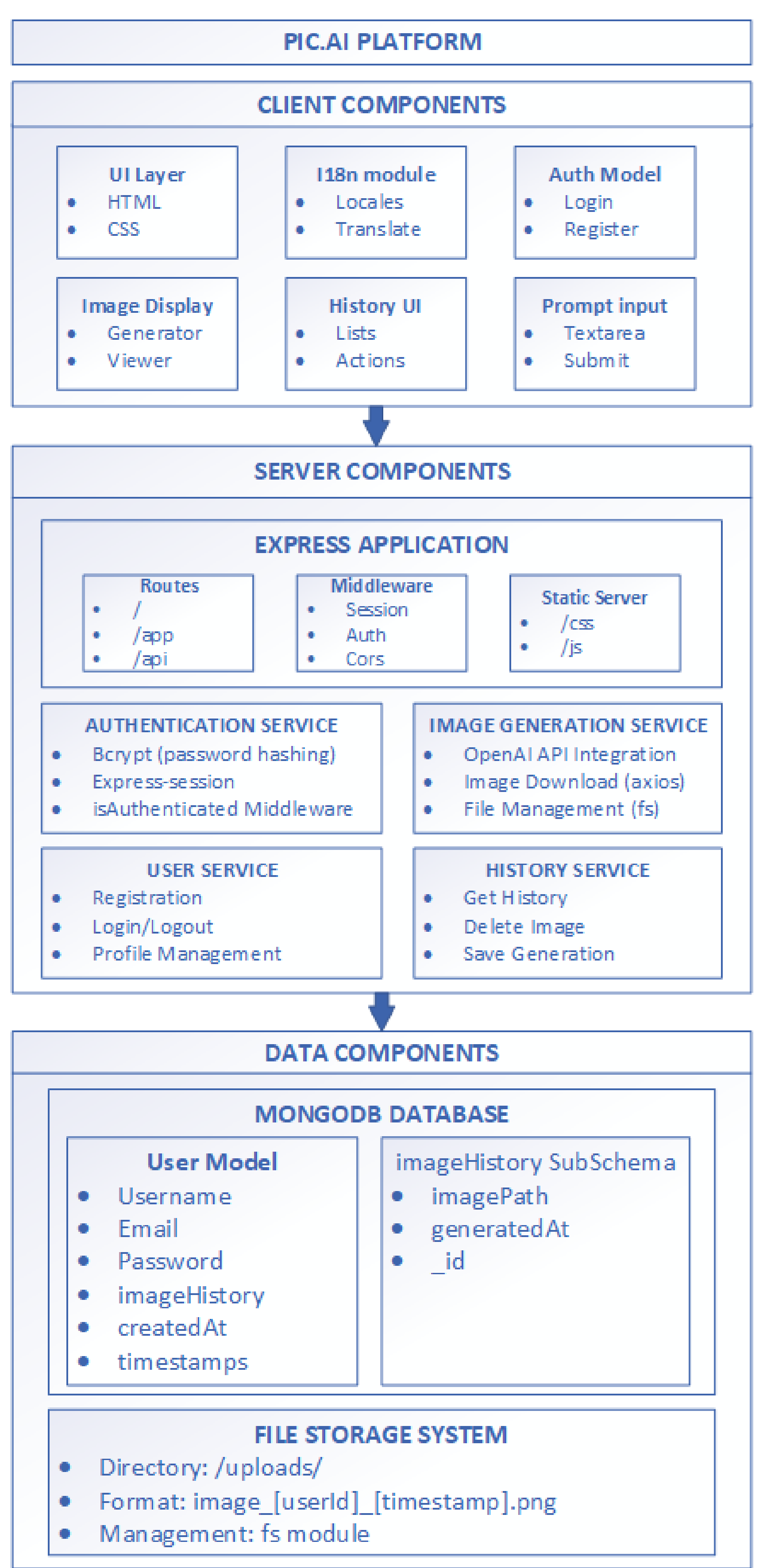


Рисунок 2.2 – Компонентна архітектурна схема вебдодатку Pic AI

2.3 Діаграма послідовностей обробки запиту генерації зображення

Діаграма послідовностей (Sequence Diagram) є ключовим інструментом UML (рис. 2.3), який описує взаємодію між компонентами системи Pic.AI у часі. Вона демонструє порядок обміну повідомленнями між об'єктами та акторами системи для виконання конкретних сценаріїв використання. Для платформи Pic.AI визначено три основні сценарії, які покривають повний життєвий цикл роботи користувача: автентифікація (реєстрація/вхід), генерація зображення та управління історією генерацій.

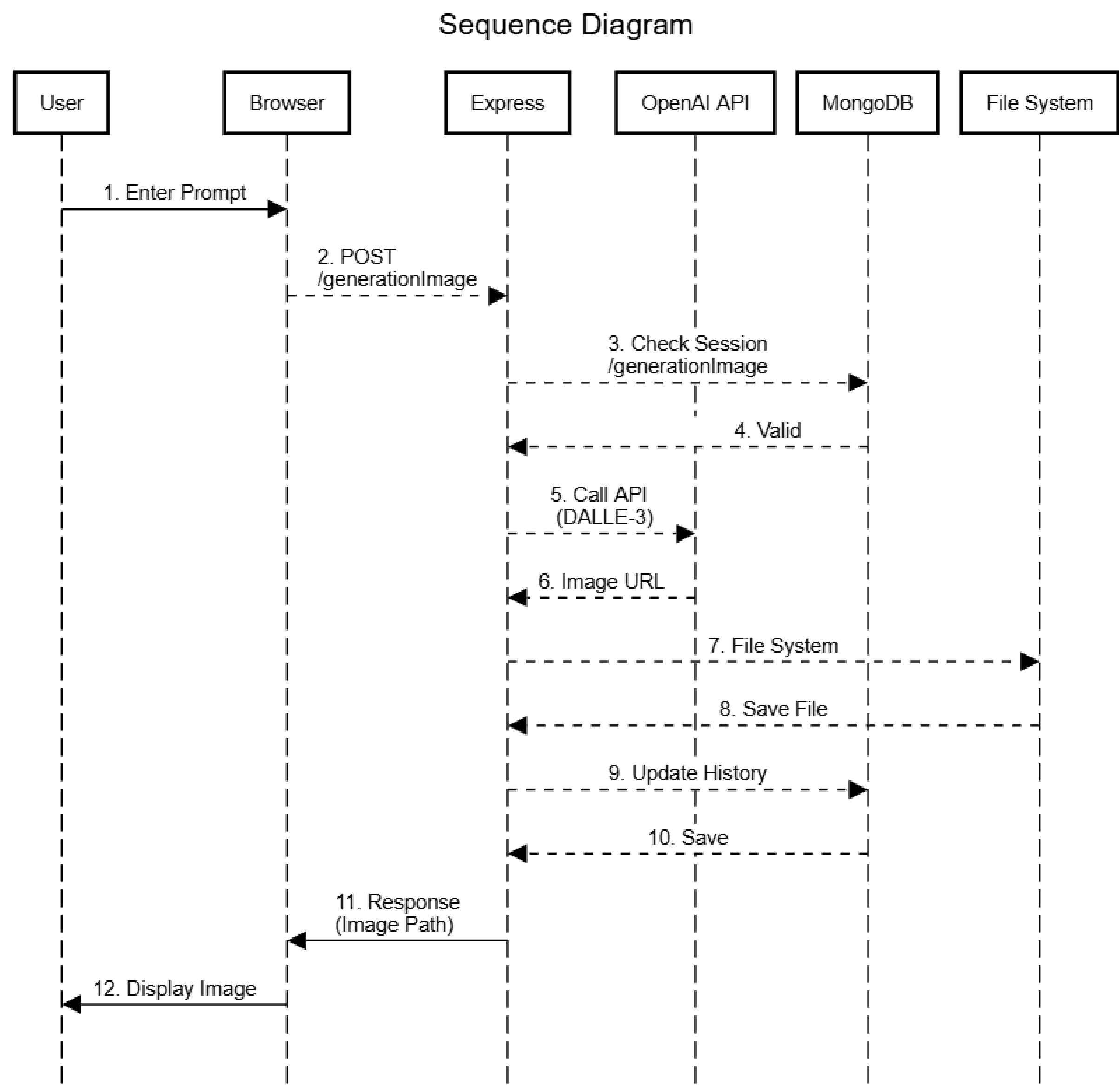


Рисунок 2.3 – Діаграма послідовностей обробки запиту /generateImage

Сценарій 1: Реєстрація та автентифікація користувача

Процес реєстрації починається з відкриття модального вікна автентифікації користувачем на головній сторінці index.html. Користувач заповнює форму реєстрації з полями username, email та password, після чого JavaScript-обробник auth-modal.js перехоплює подію submit форми. Клієнтський код валідує введені дані (перевірка збігу паролів, формату email), формує JSON-об'єкт з даними та надсилає POST-запит до ендпоінту /register через Fetch API.

Запит проходить через ланцюжок проміжного ПЗ Express.js: express.json() парсить JSON-тіло в req.body, express-session ініціалізує об'єкт сесії. Серверний обробник /register отримує дані, виконує валідацію на стороні сервера (перевірка наявності всіх полів), звертається до MongoDB через Mongoose для перевірки унікальності email методом User.findOne({ email }). Якщо користувач з таким email вже існує, повертається помилка з HTTP статусом 400.

При успішній валідації пароль хешується через bcrypt.hash(password, 10) з 10 раундами salt, створюється новий документ User з полями username, email (в lowercase), hashedPassword, порожнім масивом imageHistory та timestamp createdAt. Документ зберігається в MongoDB методом newUser.save(), який повертає Promise. Після успішного збереження сервер відправляє JSON-відповідь { success: true, message: "Registration successful!" } з HTTP статусом 201.

Клієнт отримує відповідь, парсить JSON, закриває модальне вікно, очищує форму та автоматично перемикає таб на форму входу для подальшої автентифікації. Процес входу аналогічний, але включає додатковий крок порівняння паролів через bcrypt.compare(password, user.password) та створення серверної сесії з збереженням userId, username та email в req.session. При успішному вході клієнт редиректиться на /app сторінку.

Сценарій 2: Генерація зображення

Після успішної автентифікації користувач потрапляє на робочу сторінку app.html, де вводить текстовий промпт у текстове поле. При натисканні кнопки "Start!" JavaScript-обробник script.js перехоплює подію submit, валідує наявність

тексту промпту, блокує поле введення та кнопку (disable), відображає анімований loader та очищує попереднє зображення з контейнера #image.

Формується POST-запит до /generateImage з JSON-тілом { prompt: "user's prompt text" }, який автоматично включає session cookie для автентифікації. Запит проходить через проміжне ПЗ isAuthenticated, який перевіряє req.session.userId - якщо сесія відсутня або прострочена, користувач редиректиться на головну сторінку з HTTP статусом 302.

При успішній автентифікації обробник generateImage отримує промпт з req.body.prompt та userId з req.session.userId. Формується запит до OpenAI API через SDK: openai.images.generate({ model: 'dall-e-3', prompt, n: 1, size: "1024x1024" }). Цей виклик є асинхронним та типово виконується 10-60 секунд. OpenAI API повертає об'єкт відповіді з масивом data, що містить URL тимчасового зображення response.data[0].url.

Сервер ініціює завантаження зображення через функцію downloadImage(url, filepath), яка створює GET-запит до OpenAI серверів через axios з responseType: 'stream'. Отриманий ReadableStream pipe-иться в WritableStream створений через fs.createWriteStream(filepath), що записує байти безпосередньо в файл uploads/image_{userId}_{timestamp}.png. Після завершення завантаження формується відносний шлях /uploads/image_xxx.png.

Сервер звертається до MongoDB через User.findById(userId), додає новий об'єкт в масив imageHistory методом user.imageHistory.push({ imagePath, generatedAt: new Date() }) та зберігає зміни через user.save(). Mongoose автоматично генерує _id для піддокумента та оновлює updatedAt timestamp. Після успішного збереження повертається JSON-відповідь { success: true, data: relativePath, message: "Image successfully generated and saved" }.

Клієнт отримує відповідь, витягує imagePath з data.data, встановлює його як background-image елемента #image через style.backgroundImage = url(${imageUrl}), приховує loader, розблоковує форму та викликає loadHistory() для оновлення панелі історії. Браузер окремим GET-запитом завантажує зображення з /uploads/..., який обслуговується проміжним ПЗ Express static.

Сценарій 3: Управління історією генерацій

При завантаженні сторінки app.html або після успішної генерації викликається функція loadHistory(), яка надсилає GET-запит до /api/history. Запит проходить через проміжне ПЗ isAuthenticated, обробник звертається до MongoDB через User.findById(userId).select('imageHistory') для отримання тільки масиву історії без інших полів користувача.

MongoDB повертає документ користувача, сервер сортує масив imageHistory за полем generatedAt у спадному порядку через sort((a, b) => new Date(b.generatedAt) - new Date(a.generatedAt)) та повертає JSON-відповідь { success: true, history: sortedHistory }. Клієнт отримує масив, очищає контейнер #historyList методом innerHTML = '' та генерує DOM-елементи для кожного запису історії.

Кожен елемент історії містить: img з src="${item.imagePath}" для мініатюри (60x60px), div з датою/часом генерації (локалізовані формати через toLocaleDateString(locale) та toLocaleTimeString(locale)), кнопку завантаження з SVG іконкою та кнопку видалення з символом "×". До кожного елемента додаються event listeners: клік по зображенню/даті завантажує зображення в основну область перегляду, клік по кнопці download відкриває зображення в новій вкладці для збереження.

При кліку на кнопку delete відображається модальне вікно підтвердження з локалізованим текстом через confirm(I18n.t('app.confirmDelete')). При підтвердженні надсилається DELETE-запит до /api/history/:imageId з ідентифікатором зображення в URL параметрах. Сервер через проміжне ПЗ isAuthenticated перевіряє автентифікацію, знаходить користувача, шукає зображення в масиві imageHistory за _id через findIndex(item => item._id.toString() === imageId).

Якщо зображення знайдено, сервер витягує imagePath, формує повний шлях до файлу через path.join(__dirname, 'public', imagePath), перевіряє існування файлу через fs.existsSync(fullPath) та видаляє його синхронно через fs.unlinkSync(fullPath). Після видалення файлу з диску видаляється запис з

MongoDB через user.imageHistory.splice(imageIndex, 1) та зберігаються зміни user.save(). Повертається JSON-відповідь { success: true, message: "Image successfully deleted" }.

Клієнт отримує підтвердження, додаються CSS стилі для плавної анімації зникнення елемента (opacity: 0, transform: translateX(-20px)), чекає 300мс через setTimeout та видаляє DOM-елемент методом remove(). Це забезпечує плавний UX при видаленні без різких стрибків інтерфейсу.

На кожному етапі взаємодії система передбачає обробку потенційних помилок. При реєстрації можливі помилки: дублювання email (400 Bad Request), відсутність обов'язкових полів (400), збій MongoDB (500 Internal Server Error). При генерації зображення: недійсний API ключ OpenAI (401 Unauthorized), перевищення квоти (429 Too Many Requests), некоректний промпт (400), помилки мережі при завантаженні файлу, помилки файлової системи при записі.

Всі асинхронні операції обгорнуті в try/catch блоки, помилки логуються на сервері через console.error() з контекстною інформацією (тип помилки, stack trace, параметри запиту). Клієнту повертаються зрозумілі повідомлення про помилки без розкриття технічних деталей або структури бази даних, що відповідає best practices безпеки. При помилках клієнтський JavaScript відображає alert або inline повідомлення, приховує loader та дозволяє користувачу повторити спробу без перезавантаження сторінки.

3 РЕАЛІЗАЦІЯ ПЛАТФОРМИ PIC.AI

3.1 Розробка серверної частини вебплатформи

3.1.1 Ініціалізація та конфігурація сервера

Серверна частина платформи Pic.AI реалізована на основі Node.js [12-14] середовища версії 20+ з використанням вебфреймворку Express.js версії 4.19.2. Архітектурне рішення базується на модульному підході з чітким розділенням відповідальності між компонентами: конфігурація та ініціалізація сервера, проміжне ПЗ для обробки запитів, маршрутизація та контролери для бізнес-логіки, моделі даних для роботи з MongoDB, утиліти для інтеграції із зовнішніми сервісами. Основний файл index.js виконує роль точки входу до додатку, де здійснюється налаштування всіх компонентів, підключення до бази даних, реєстрація проміжного ПЗ та визначення маршрутів.

Процес ініціалізації серверного додатку починається з завантаження змінних оточення через бібліотеку dotenv версії 16.4.5. Виклик require("dotenv").config() на початку файлу index.js зчитує файл .env з кореневої директорії проекту та завантажує всі визначені в ньому змінні в об'єкт process.env. Це дозволяє безпечно зберігати конфіденційні дані поза системою контролю версій: PORT=5000 визначає порт для запуску сервера, MONGODB_URI містить рядок підключення до MongoDB Atlas з автентифікаційними даними, OPENAI_API_KEY зберігає API ключ для доступу до OpenAI сервісів, JWT_SECRET резервується для майбутньої реалізації token-based автентифікації.

Створення екземпляру Express додатку виконується через const app = express(), після чого налаштовується система проміжного ПЗ для обробки вхідних запитів. Проміжне ПЗ express.json() парсить JSON-тіло запитів та робить дані доступними через req.body, що необхідно для обробки POST-запитів з клієнта. Проміжне ПЗ express.urlencoded({ extended: false }) обробляє дані форм, закодовані у форматі URL-encoded, хоча в Pic.AI основний обмін

даними відбувається через JSON. Проміжне ПЗ express.static("./public") налаштовує обслуговування статичних файлів (HTML, CSS, JavaScript, зображення) з директорії public, дозволяючи клієнту завантажувати ресурси за прямими URL-шляхами.

Окреме проміжне ПЗ static middleware app.use('/uploads', express.static(path.join(__dirname, 'uploads'))) налаштовує доступ до згенерованих зображень через URL-префікс uploads. Це дозволяє клієнту запитувати файли за шляхами типу /uploads/image_507f1f77bcf86cd799439011_1702905600000.png, які Express автоматично резолвить у файли з відповідної директорії на диску. Проміжне ПЗ Static middleware також встановлює правильні MIME-типи (image/png для PNG-файлів) та підтримує HTTP-кешування через заголовки ETag та Last-Modified для оптимізації повторних запитів.

3.1.2 Система управління сесіями

Автентифікація користувачів в Pic.AI реалізована через серверні сесії з використанням бібліотеки express-session версії 1.18.2. Проміжне ПЗ сесій налаштовується через наступний виклик (рис. 3.1).

```
app.use(session({
    secret: 'your_secret_key',
    resave: false,
    saveUninitialized: false,
    cookie: { secure: false }
}));
```

Рисунок 3.1 – Налаштування сесій express-session

Параметр secret використовується для підпису session ID cookie криптографічним хешем, що захищає від підробки ідентифікаторів сесій. В production середовищі цей ключ має бути складним випадковим рядком, згенерованим криптографічно безпечним генератором, та зберігатися в змінних

оточення. Опція resave: false вказує, що сесія не повинна зберігатися повторно в store, якщо вона не була модифікована під час обробки запиту, що знижує навантаження на систему зберігання сесій.

Параметр saveUninitialized: false забороняє зберігання порожніх сесій для неавторизованих користувачів, що економить пам'ять та відповідає налаштуванням приватності, оскільки не створює coockies для користувачів, які не дали згоди через автентифікацію. Опція cookie: { secure: false } дозволяє використання куки через незахищене HTTP-з'єднання під час розробки, але в production середовищі має бути встановлена в true для обов'язкового використання HTTPS, що захищає session ID від перехоплення через man-in-the-middle атаки.

В поточній конфігурації сесії зберігаються в пам'яті сервера (default MemoryStore), що прийнятно для розробки та невеликих навантажень, але не підходить для production з кількома інстансами сервера. Дані сесії втрачаються при перезапуску сервера, і кожен інстанс має своэ власне сховище, що робить неможливим балансування навантаження. Для роботи у реальних умовах рекомендується перехід на Redis або Memcached через connect-redis або connect-mongo пакети, що дозволяє централізоване зберігання сесій, доступне всім екземплярам сервера.

При успішній автентифікації (вхід або реєстрація з автоматичним входом) в об'єкті сесії зберігаються дані користувача (рис. 3.2).

```
req.session.userId = user._id;
req.session.username = user.username;
req.session.email = user.email;
```

Рисунок 3.2 – Зберігання даних користувачів

Mongoose автоматично конвертує ObjectId в рядок при серіалізації в session store. Ці дані стають доступними в усіх наступних запитах від того ж користувача через req.session, що дозволяє ідентифікувати користувача без

повторної перевірки паролю. Session ID зберігається на клієнті в cookie з назвою connect.sid (за замовчуванням), яка автоматично відправляється з кожним запитом до сервера.

Проміжне ПЗ isAuthenticated() реалізує захист приватних маршрутів від несанкціонованого доступу (рис. 3.3).

```
function isAuthenticated(req, res, next) {
    if (req.session && req.session.userId) {
        return next();
    }
    res.redirect('/');
}
```

Рисунок 3.3 – Проміжне ПЗ для захисту від несанкціонованого доступу

Це проміжна функція перевіряє наявність userId в об'єкті сесії та дозволяє продовження обробки запиту через next() для автентифікованих користувачів, або редиректити на головну сторінку для неавторизованих. Проміжне ПЗ застосовується до маршрутів /app, /api/user, /api/history, /generateImage, /api/history/:imageId, забезпечуючи захист всього функціоналу генерації та управління історією.

3.1.3 Реалізація маршрутів автентифікації

Маршрут реєстрації POST /register (Додаток А) обробляє створення нових користувачів з валідацією даних на серверній стороні та безпечним зберіганням паролів. Обробник використовує async/await синтаксис для читабельності асинхронного коду та обгортає всю логіку в try/catch блок для централізованої обробки помилок. Деструктуризація { username, email, password } з req.body витягує дані з JSON-тіла запиту, яке було зчитане проміжне ПЗ express.json(). Перша валідація перевіряє наявність всіх обов'язкових полів та повертає HTTP 400 Bad Request з описовим повідомленням при відсутності будь-якого поля.

Перевірка унікальності email виконується через User.findOne({ email: email.toLowerCase() }), який шукає документ з відповідним email в колекції. Конвертація в lowercase забезпечує регістронезалежність пошуку, оскільки поле email в схемі також конвертується в lowercase через опцію lowercase: true. Якщо користувач з таким email вже існує, повертається помилка 400 замість створення дубліката, що відповідає принципу fail-fast та покращує UX через чіткий зворотній зв’язок.

Хешування пароля виконується через bcrypt.hash(password, 10), де другий аргумент (10) визначає фактор складності – кількість раундів генерації солі. Значення 10 означає 2^10 = 1024 ітерацій алгоритму Blowfish, що дає час хешування близько 100-200 мс на сучасному обладнанні. Це достатньо для захисту від brute-force атак (підбір пароля стає вкрай повільним), але не створює помітних затримок при реєстрації. Bcrypt автоматично генерує унікальну сіль для кожного пароля та включає її в результуючий хеш, тому не потрібно зберігати сіль окремо.

Створення нового документа користувача виконується через конструктор моделі new User({...}) з подальшим викликом save(), який валідує дані згідно схеми, генерує _id (ObjectId), встановлює timestamps та вставляє документ в колекцію Users. Метод save() повертає Promise, який резолвиться з збереженим документом або відкликається з помилкою валідації або дублювання унікальних полів. При успішному збереженні клієнту повертається статус 201 Created та JSON з підтвердженням.

Всі помилки бази даних, валідації або bcrypt перехоплюються catch блоком, логуються в консоль через console.error() для debugging та моніторингу, а клієнту повертається загальне повідомлення "Server error" без розкриття технічних деталей, що відповідає best practices безпеки. HTTP статус 500 Internal Server Error вказує на проблему на стороні сервера, відрізняючи її від клієнтських помилок (400).

Маршрут входу POST /login виконує автентифікацію користувача з перевіркою пароля та створенням сесії (Додаток А). Процес автентифікації

починається з пошуку користувача за email. Якщо користувач не знайдений, повертається той самий HTTP статус 401 Unauthorized з тим самим повідомленням "Incorrect email or password", що і при невірному паролі. Такий підхід запобігає enumeration attacks, коли зловмисник може визначити існуючі email адреси в системі, аналізуючи різні повідомлення про помилки для неіснуючих та існуючих акаунтів.

Перевірка пароля виконується через bcrypt.compare(password, user.password), який хешує введений пароль з тією ж сіллю, що зберігається в user.password, та порівнює результуючі хеші. Bcrypt автоматично витягує сіль з збереженого хешу (вона включена в рядок хешу), тому не потрібно передавати сіль окремо. Метод повертає Promise<boolean>, значення якого встановлюється true при співпадінні паролів або false при розбіжності. Асинхронний характер операції (через async/await) важливий, оскільки bcrypt є інтенсивною операцією для процесора через багато раундів хешування.

При успішній валідації створюється серверна сесія через запис даних в req.session. Проміжне ПЗ Express-session автоматично генерує унікальний session ID, зберігає дані сесії на сервері (в MemoryStore або зовнішньому store) та встановлює httpOnly cookie з session ID на клієнті. Збереження userId, username та email в сесії дозволяє ідентифікувати користувача в наступних запитах без повторної перевірки пароля. ObjectId user._id автоматично серіалізується в рядок при зберіганні в session store. Відповідь клієнту включає поле redirectUrl: "/app", яке клієнтський JavaScript використовує для автоматичного редиректу на робочу сторінку після успішного входу. Такий підхід дозволяє уникнути серверного редиректу (HTTP 302), який би вимагав повного перезавантаження сторінки, та дозволяє клієнту виконати додаткову логіку перед переходом (наприклад, анімацію або закриття модального вікна).

Маршрут виходу GET /logout (рис. 3.4) знищує серверну сесію та редиректить користувача на головну сторінку. Метод req.session.destroy() видаляє дані сесії з store та інвалідує session ID, роблячи його непридатним для подальшого використання. Callback-функція викликається після завершення

операції знищення, незалежно від її успішності. Помилки логуються, але не перешкоджають редиректу, оскільки з точки зору користувача вихід вважається успішним навіть якщо сесія не була коректно знищена на сервері (вона стане недійсною після timeout).

```
app.get("/logout", (req, res) => {
    req.session.destroy((err) => {
        if (err) {
            console.error('Logout error:', err);
        }
        res.redirect('/');
    });
});
```

Рисунок 3.4 – Маршрут для знищення сесії і виходу з облікового запису користувача

3.1.4 API-ендпоінти для роботи з даними користувача

Маршрут GET /api/user надає клієнту інформацію про поточного авторизованого користувача для відображення в інтерфейсі (рис. 3.5).

```
app.get("/api/user", isAuthenticated, async (req, res) => {
    try {
        const user = await User.findById(req.session.userId).select('email username');
        if (!user) {
            return res.status(404).json({ success: false, message: "User not found" });
        }
        res.json({ success: true, email: user.email, username: user.username });
    } catch (error) {
        console.error('User data error:', error);
        res.status(500).json({ success: false, message: "Server error" });
    }
});
```

Рисунок 3.5 – Маршрут для знищення сесії і виходу з облікового запису користувача

Проміжне ПЗ isAuthenticated виконується перед основним обробником та гарантує, що req.session.userId існує та користувач автентифікований. Пошук користувача виконується через User.findById() з використанням projection .select('email username'), який вказує Mongoose повернути тільки поля email та username, виключаючи password, imageHistory та інші поля. Це економить bandwidth, прискорює запит та відповідає принципу «мінімум привілегій» – клієнт отримує тільки необхідні йому дані.

Перевірка if (!user) обробляє крайні випадки, коли userId існує в сесії, але відповідний документ був видалений з бази даних (наприклад, через адміністративне видалення акаунту). В такому випадку повертається 404 Not Found замість 200 з null, що дозволяє клієнту коректно обробити ситуацію (наприклад, примусово вийти з системи та показати повідомлення про видалення акаунту).

Маршрут GET /api/history повертає повну історію генерацій користувача відсортовану за датою (рис. 3.6).

```
app.get("/api/history", isAuthenticated, async (req, res) => {
    try {
        const user = await User.findById(req.session.userId).select('imageHistory');
        if (!user) {
            return res.status(404).json({ success: false, message: "User not found" });
        }
        const sortedHistory = user.imageHistory.sort((a, b) =>
            new Date(b.generatedAt) - new Date(a.generatedAt)
        );
        res.json({ success: true, history: sortedHistory });
    } catch (error) {
        console.error('History error:', error);
        res.status(500).json({ success: false, message: "Server error" });
    }
});
```

Рисунок 3.6 – Маршрут для повернення історії генерацій зображень користувача

Метод .select('imageHistory') обмежує результат тільки масивом історії, виключаючи email, username, password та інші поля користувача. Сортування

виконується на стороні сервера через JavaScript метод .sort() з компаратором, який віднімає timestamp нової дати від старої, створюючи спадний порядок (DESC). Конвертація в new Date() необхідна, оскільки generatedAt зберігається як Date об'єкт в MongoDB, але після серіалізації в JSON може бути представлений як рядок ISO 8601.

Маршрут DELETE /api/history/:imageId виконує видалення конкретного зображення з історії користувача та файлової системи (Додаток А). Параметр imageId витягується з URL через req.params.imageId, де Express автоматично парсить динамічний сегмент :imageId з маршруту /api/history/:imageId. Пошук зображення в масиві imageHistory виконується через findIndex() з порівнянням item._id.toString() === imageId, де конвертація ObjectId в рядок необхідна, оскільки параметр URL завжди є рядком. Метод findIndex() повертає індекс першого елемента, що задовольняє умову, або -1 якщо елемент не знайдено.

Перевірка належності зображення поточному користувачу забезпечується неявно через пошук в user.imageHistory, де user отримується через req.session.userId. Це запобігає ситуації, коли один користувач може видалити зображення іншого користувача, вгадавши або перебравши imageId. Додаткова валідація не потрібна, оскільки зображення фізично знаходиться в imageHistory конкретного користувача. Формування повного шляху до файлу виконується через path.join(__dirname, 'public', imagePath), де __dirname вказує на директорію поточного модуля (кореневу директорію проекту), 'public' та imagePath (наприклад, /uploads/image_xxx.png) склеюються з правильними роздільниками для поточної ОС (\ на Windows, / на Unix). Перевірка існування файлу через fs.existsSync() запобігає помилкам при спробі видалення неіснуючого файлу (orphaned database records).

Видалення файлу виконується синхронно через fs.unlinkSync(), що блокує event loop на кілька мілісекунд, але прийнятно для операції видалення, яка виконується рідко. Асинхронний варіант fs.unlink() з callback або promises вимагає додаткової логіки обробки помилок та ускладнює код. Після видалення

файлу запис видаляється з масиву через splice(imageIndex, 1), який модифікує масив in-place, видаляючи один елемент за вказаним індексом.

Збереження змін виконується через user.save(), який валідує документ, оновлює updatedAt timestamp та записує зміни в MongoDB. Mongoose відстежує модифікації масивів та викликає відповідні update операції MongoDB (наприклад, $pull для видалення елемента з масиву). При успішному збереженні клієнту повертається підтвердження, яке клієнтський JavaScript використовує для видалення відповідного DOM-елемента з інтерфейсу з анімацією.

3.1.5 Інтеграція з OpenAI API для генерації зображень

Ініціалізація клієнта OpenAI виконується з API ключем з змінних оточення (рис. 3.7).

```
const OpenAI = require("openai");
const openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
});
```

Рисунок 3.7– Імпорт бібліотек та підключення до OpenAI API

Офіційний SDK версії 4.40.2 надає типізований інтерфейс для всіх API OpenAI з автоматичною обробкою автентифікації, логікою при тимчасових збоях, та парсингом відповідей. API ключ має формат sk-proj-... для project-based ключів або sk-... для user-based ключів та зберігається в .env файлі для безпеки. Витік API ключа дозволяє зловмисникам використовувати квоту OpenAI власника ключа, тому критично важливо не комітити .env в Git через додавання його в .gitignore.

Утилітна функція downloadImage() завантаження зображень з URL реалізована через потокову передачу даних (рис. 3.8). Параметр responseType: 'stream' вказує axios повернути ReadableStream замість буферу в пам'яті, що критично важливо для завантаження великих файлів (зображення 1024x1024 можуть важити 0.5-2 МБ). Потік response.data входить в WritableStream

створений через fs.createWriteStream(filepath), який записує дані безпосередньо в файл по мірі їх надходження по кілька кілобайт.

```
async function downloadImage(url, filepath) {
    const response = await axios({
        method: 'GET',
        url: url,
        responseType: 'stream'
    });

    const writer = fs.createWriteStream(filepath);
    response.data.pipe(writer);

    return new Promise((resolve, reject) => {
        writer.on('finish', resolve);
        writer.on('error', reject);
    });
}
```

Рисунок 3.8– Функція завантаження зображень

Основна функція генерації зображень generateImage() координує весь процес від отримання промпту до збереження результату (додаток А). Виклик openai.images.generate() надсилає асинхронний запит до OpenAI API з параметрами моделі (dall-e-3 – найновіша версія станом на 2024-2025), промпту від користувача, кількості зображень (n=1, оскільки DALL-E 3 підтримує тільки одне зображення за запит) та розміру (1024x1024 пікселів для квадратного формату). Модель DALL-E 3 автоматично покращує промпти користувача для кращих результатів, що відображається в полі revised_prompt відповіді.

Генерація зображення типово займає 10-60 секунд залежно від складності промпту та завантаженості серверів OpenAI. Під час очікування Node.js event loop продовжує обробляти інші запити завдяки асинхронній природі await, не блокуючи інших користувачів. OpenAI API повертає об'єкт з масивом data, де

кожен елемент містить url тимчасового зображення (дійсний близько 1 години) та опціонально revised_prompt.

Формування унікального імені файлу через image_${userId}_${timestamp}.png забезпечує унікальність навіть при одночасних генераціях одним користувачем (timestamp в мілісекундах змінюється з кожним викликом Date.now()). Префікс image_ полегшує ідентифікацію файлів, userId ізолює файли між користувачами, а розширення .png вказує формат. Такий спосіб назви дозволяє легко знайти всі зображення конкретного користувача через filesystem globbing або database queries.

Завантаження зображення через downloadImage() зберігає файл локально, дозволяючи платформі контролювати доступ до зображень, забезпечувати їх довгострокову доступність незалежно від тимчасових URL OpenAI, та реалізувати функції типу watermark або компресії в майбутньому. Відносний шлях /uploads/${filename} зберігається в базі даних та використовується клієнтом для завантаження зображення через проміжне ПЗ Express static.

Оновлення історії користувача виконується через пошук документа, додавання нового піддокумента в масив imageHistory методом push() та збереження через save(). Mongoose автоматично генерує _id для піддокумента, встановлює generatedAt в поточний час (через default: Date.now), валідує структуру згідно imageHistorySchema та оновлює updatedAt timestamp головного документа. При успішному збереженні клієнту повертається відносний шлях, який він використовує для відображення зображення та оновлення UI.

Обробка помилок розрізняє HTTP-помилки OpenAI API (доступні через error.response) та інші помилки (мережеві, помилки парсингу, файлової системи). Логування error.response.status та error.response.data надає інформацію про причину відмови API (401 для недійсного ключа, 429 для перевищення rate limits, 400 для некоректного промпту через content policy violations). Клієнту повертається узагальнене повідомлення "The image could not be generated" без технічних деталей для безпеки та UX.

3.1.6 Маршрутизація та обробка статичних сторінок

Маршрут головної сторінки обслуговує лендінг для незареєстрованих користувачів. Метод res.sendFile() відправляє HTML-файл з встановленням правильного Content-Type заголовка (text/html), підтримкою кешування через ETag та автоматичною обробкою Range-запитів. Використання path.join(__dirname, "public", "index.html") замість хардкоженого шляху забезпечує кросплатформенну сумісність (Windows vs Unix роздільники шляхів) та коректність відносно поточної директорії модуля.

Захищений маршрут робочої області доступний тільки автентифікованим користувачам. Проміжне ПЗ isAuthenticated() виконується перед відправкою файлу та редиректить неавторизованих користувачів на головну сторінку. Це забезпечує захист робочої області від прямого доступу через URL без входу в систему. Альтернативний підхід – використання серверного рендерингу для відображення різного контенту залежно від стану автентифікації, але для SPA (Single Page Application) архітектури достатньо клієнтської логіки з серверною валідацією на рівні API.

Маршрут для обробки неіснуючих URL повертає кастомну 404 сторінку. Зірочка в Express маршрутизації відповідає будь-якому шляху, що не був оброблений попередніми маршрутами. Встановлення статусу 404 перед відправкою файлу важливо для правильної обробки браузерами та пошуковими системами. Кастомна сторінка 404 покращує UX порівняно з дефолтною помилкою Express, надаючи користувачу корисні посилання для навігації.

Фінальний крок ініціалізації – запуск HTTP-сервера на вказаному порті. Метод app.listen() створює HTTP-сервер через вбудований модуль Node.js http, прив'язує його до вказаного порту (5000 за замовчуванням або з process.env.PORT) та починає приймати вхідні з'єднання. Callback-функція викликається після успішного запуску або при помилці (наприклад, порт вже зайнятий іншим процесом).

3.2 Робота з базою даних MongoDB

Інтеграція з MongoDB реалізована через ODM (Object Document Mapper) бібліотеку Mongoose версії 9.0.0, яка надає схемну валідацію, типізацію, проміжне ПЗ hooks та зручний API для роботи з документами [15-17]. Підключення до бази даних виконується асинхронно при старті сервера (рис. 3.9).

```
mongoose.connect(process.env.MONGODB_URI)
    .then((res) => console.log('connected'))
    .catch((error) => console.log(error));
```

Рисунок 3.9 – Підключення до MongoDB

Метод mongoose.connect() повертає Promise, який виконується при успішному підключенні або реджектиться при помилці. URI підключення має формат mongodb+srv://username:password@cluster.mongodb.net/dbname?options та включає автентифікаційні дані, адресу кластера MongoDB Atlas, назву бази даних та додаткові параметри підключення (SSL, compression, poolSize тощо) [15-17].

MongoDB Atlas – повністю керований хмарний сервіс – автоматично управляє з'єднаннями через connection pooling, підтримує автоматичне reconnection при втраті зв'язку, надає вбудовані механізми захисту через репліки та забезпечує шифрування даних в спокої та під час передачі. Mongoose автоматично використовує ці можливості через драйвер MongoDB Node.js [18], створюючи пул з'єднань для ефективного повторного використання TCP-сокетів між запитами. Модель даних користувача визначена в окремому файлі user.js та імпортується через const User = require('./models/user'). Розділення моделей від основної логіки додатку відповідає принципам модульної архітектури та MVC паттерну, де моделі відповідають за структуру та валідацію даних, контролери –

за бізнес-логіку, а views (в нашому випадку – клієнтський HTML/JS) – за представлення.

Схема користувача userSchema визначає структуру документів в колекції Users (рис.3.10).

```
const userSchema = new Schema({
  username: { type: String, required: true, unique: true, trim: true },
  email: { type: String, required: true, unique: true, lowercase: true, trim: true },
  password: { type: String, required: true },
  imageHistory: [imageHistorySchema],
  createdAt: { type: Date, default: Date.now }
}, { timestamps: true });
```

Рисунок 3.10 – Схема користувача User

Поле username зберігає відображуване ім'я користувача з автоматичним trimming пробілів та унікальністю на рівні бази даних. Поле email використовується як основний ідентифікатор для входу, автоматично конвертується в нижній регістр для уникнення дублікатів через різний регістр (user@example.com та USER@example.com вважаються одним email). Поле password зберігає bcrypt хеш пароля з автоматично згенерованою сіллю, ніколи не зберігаючи паролі в відкритому вигляді.

Вкладений масив imageHistory типу imageHistorySchema зберігає всю історію генерацій користувача в одному документі, що дозволяє отримувати всі дані одним запитом без JOIN-операцій. Підсхема imageHistorySchema визначена окремо (рис. 3.11).

```
const imageHistorySchema = new Schema({
  imagePath: { type: String, required: true },
  generatedAt: { type: Date, default: Date.now, required: true }
}, { _id: true });
```

Рисунок 3.11 – Підсхема imageHistorySchema для збереження шляху згенерованих зображень і дати їх збереження

Опція _id: true явно вказує Mongoose генерувати унікальний ObjectId для кожного піддокумента, що дозволяє звертатися до конкретних елементів історії при операціях видалення через DELETE /api/history/:imageId. Опція timestamps: true в основній схемі автоматично додає поля createdAt та updatedAt, які Mongoose оновлює при кожній операції save(). Це дозволяє відслідковувати коли був створений акаунт користувача та коли востаннє оновлювалися його дані, що корисно для аналітики, аудиту та реалізації features типу "показати останню активність користувача".

Mongoose автоматично створює індекси для полів, позначених як unique, що забезпечує швидкий пошук складністю O(log n) при автентифікації та запобігає дублюванню email або username. Індекси створюються асинхронно при першому підключенні до бази даних через виклик Model.createIndexes(), який Mongoose викликає автоматично. В реальних умовах рекомендується створювати індекси заздалегідь через міграції або MongoDB Atlas UI для уникнення затримок при старті додатку.

3.3 Розробка клієнтської частини вебплатформи

3.3.1 Загальна архітектура клієнтського додатку

Клієнтська частина платформи Pic.AI реалізована як Single Page Application (SPA) з використанням нативних вебтехнологій HTML5 [20], CSS3 [21] та JavaScript ES6+ без залежності від важких фронтенд-фреймворків типу React, Vue або Angular [22]. Такий підхід забезпечує мінімальний розмір завантажуваних ресурсів, швидке первинне завантаження сторінки та повну контрольованість коду без абстракцій фреймворків. Архітектура клієнта базується на модульній організації JavaScript-файлів, де кожен модуль відповідає за конкретну функціональність: i18n.js для інтернаціоналізації, script.js для основної бізнес-логіки робочої області, auth-modal.js для автентифікації, scroll.js для плавної навігації та burger.js для мобільного меню.

Структура проекту розділена на дві основні HTML-сторінки: index.html як публічний лендінг для незареєстрованих користувачів з секціями Home, About Us, How It Works, Examples та Contacts, і app.html як захищену робочу область для автентифікованих користувачів з інтерфейсом генерації зображень та історією. CSS-стилі організовані відповідно: styles.css для лендінга з акцентом на маркетингові елементи та app.css для функціональної робочої області з фокусом на зручність використання інструментів. Взаємодія з сервером реалізована через нативний Fetch API з асинхронними функціями async/await, що забезпечує читабельний код без хаосу з callback-фнункціями. Всі запити використовують JSON для обміну даними з встановленням правильних заголовків Content-Type та автоматичним включенням session cookie для автентифікації. Обробка помилок реалізована через try/catch блоки на кожному рівні асинхронних операцій з відображенням зрозумілого зворотнього зв'язку користувачу через UI-елементи (alert, inline повідомлення, loader-анімації).

3.3.2 Структура та семантика HTML-розмітки

Головна сторінка index.html побудована з дотриманням семантичних HTML5-тегів для покращення доступності та SEO-оптимізації [23]. Header містить навігаційне меню <nav> з посиланнями на якорі секцій (#home, #about-us, #how-it-works), селектор мови та кнопку відкриття модального вікна автентифікації. Використання семантичних тегів <header>, <nav>, <main>, <section>, <footer> дозволяє браузерам та допоміжним технологіям (screen readers) правильно інтерпретувати структуру сторінки. Секція Home реалізована як звичанй секція з градієнтним фоном, центрованим заголовком з span-елементом для градієнтного тексту через CSS background-clip техніку, описовим підзаголовком та функціональною кнопкою. Декоративний елемент .home__glow з радіальним градієнтом та CSS-анімацією @keyframes float створює ефект плаваючого світіння, додаючи динамічності дизайну. Всі текстові елементи позначені атрибутами data-i18n для автоматичної підстановки перекладів при зміні мови. Секція About Us складається з трьох .about-

us__comment блоків, кожен з яких є окремим смисловим коментарем про платформу. Блоки стилізовані з напівпрозорим фоном, border з градієнтом та backdrop-filter для ефекту розмиття (frosted glass effect), що створює сучасний вигляд. Секція How It Works використовує flexbox для розташування чотирьох кроків з числовими індикаторами в колах, текстовими описами та стрілками між ними. На мобільних пристроях (медіа-запит @media (max-width: 1269px)) стрілки приховуються, а кроки перебудовуються в дві колонки через flex: 0 1 calc(50% - 20px), забезпечуючи адаптивність без втрати читабельності [24]. Секція Examples демонструє три приклади згенерованих зображень за допомогою grid layout: перше зображення займає повну ширину, друге та третє розташовані вертикально поруч. Всі зображення обгорнуті в div з класом .pic та мають hover-ефект з transform: translateY та збільшенням box-shadow, що створює відчуття "підняття" елемента над сторінкою. Секція Contacts організована як flexbox-контейнер з трьома картками (Phone, Address, Email), кожна з яких містить SVG-іконку, заголовок та контактну інформацію. Використання SVG замість растрових іконок забезпечує чіткість відображення на екранах з високою щільністю пікселів (Retina дисплеї) та можливість зміни кольору через CSS fill або stroke [25, 26]. Модальне вікно автентифікації .auth-modal-overlay реалізоване через fixed-позиціонування на весь екран з напівпрозорим фоном та backdrop-filter: blur для розмиття контенту під ним. Сам modal .auth-modal центрується через flexbox на overlay та містить систему табів для перемикання між формами входу та реєстрації. Форми побудовані семантично з <form>, <label> та <input> елементами, де label пов'язані з input через атрибути for/id для покращення accessibility.

3.3.3 Стилізація через CSS та responsive design

CSS-архітектура базується на принципах БЕМ-методології (Block Element Modifier) для організації селекторів: .block__element--modifier синтаксис робить код самодокументованим та знижує специфічність селекторів. Використання CSS змінних в :root для кольорів, проміжків та transition-параметрів дозволяє

централізовано керувати дизайн-системою та легко змінювати тему. Reset-секція на початку styles.css нормалізує стилі браузера за замовчуванням для передбачуваного відображення across browsers. Скидання margin, padding для заголовків та списків, встановлення box-sizing: border-box для всіх елементів через селектор * спрощує розрахунки розмірів елементів. Адаптивний design реалізований через mobile-first підхід з медіа-запитами для опорних точок: 1269px (tablets), 800px (small tablets/large phones), 480px (phones), 360px (small phones).

Навігаційне меню на мобільних пристроях трансформується в «hamburger menu» з CSS-анімованою іконкою (три горизонтальні лінії перетворюються в X при активації) та slide-in side panel через CSS transitions. Checkbox hack (<input type="checkbox"> з display: none та <label> як тригер) дозволяє реалізувати toggle-функціональність без JavaScript. Градієнтні фони створюються через linear-gradient() функцію CSS для секцій, кнопок та декоративних елементів. Складні градієнти типу linear-gradient(135deg, #667eea 0%, #764ba2 100%) створюють глибину та візуальний інтерес. Gradient text effect досягається через -webkit-background-clip: text та -webkit-text-fill-color: transparent для webkit-based браузерів з випадаючим кольором для інших [27].

3.3.4 Система інтернаціоналізації (i18n)

Модуль i18n.js реалізує повноцінну систему багатомовності з підтримкою англійської та української мов через JSON-файли перекладів. Архітектура модуля базується на singleton паттерні з об'єктом I18n, що експортує публічне API для роботи з перекладами (рис. 3.12).

Метод init() завантажує обидва JSON-файли паралельно через Promise.all() для оптимізації часу завантаження, парсить їх та зберігає в об'єкті translations. Використання localStorage.getItem('language') дозволяє зберігати вибір мови між сесіями користувача без необхідності серверного збереження налаштувань. Метод t(key) (скорочення від translate) приймає ключ у форматі dot notation (nav.home, auth.password) та рекурсивно проходить вкладену

структуру JSON для отримання значення. Оператор ?. запобігає помилкам при доступі до неіснуючих ключів, повертаючи сам ключ як випадаючий. Метод setLanguage(lang) оновлює поточну мову, зберігає вибір в localStorage та викликає updatePage(), який проходить по всіх DOM-елементах з атрибутами data-i18n та data-i18n-placeholder, замінюючи їх вміст на відповідні переклади. Кастомна подія languageChanged дозволяє іншим модулям (наприклад, історії генерацій) реагувати на зміну мови та оновлювати динамічно згенерований контент.

```
updatePage() {
    document.querySelectorAll('[data-i18n]').forEach(element => {
        const key = element.getAttribute('data-i18n');
        element.textContent = this.t(key);
    });

    document.querySelectorAll('[data-i18n-placeholder]').forEach(element => {
        const key = element.getAttribute('data-i18n-placeholder');
        element.placeholder = this.t(key);
    });

    window.dispatchEvent(new CustomEvent('languageChanged', {
        detail: { language: this.currentLanguage }
    }));
}
```

Рисунок 3.12 – Функція updatePage() для реалізації системи багатомовності

3.3.5 Модальне вікно автентифікації

Модуль auth-modal.js реалізує повноцінний UX потік автентифікації з перемиканням між формами входу та реєстрації через систему вкладок. JavaScript-код використовує делегацію подій та DOM-маніпуляцію для створення інтерактивного інтерфейсу (Додаток А). Відкриття модального вікна здійснюється через додавання класу active до overlay, який змінює visibility: hidden та opacity: 0 на visible та 1 з плавним переходом. Використання querySelectorAll('.open-auth-btn') дозволяє мати множину кнопок відкриття без дублювання коду. Закриття модального вікна реалізовано через три механізми: клік на кнопку закриття (×), клік на overlay поза модальним вікном, та ESC key

(опціонально). StopPropagation на самому модальному вікні запобігає закриттю при кліку всередині форми.

Валідація на клієнтській стороні перевіряє збіг паролів для форми реєстрації перед відправкою на сервер, що економить мережевий трафік та прискорює зворотній зв'язок користувачу. Перейменування поля name в username виконується для узгодження з серверною схемою MongoDB. Видалення confirmPassword перед відправкою запобігає передачі зайвих даних на сервер.

Функція showError() динамічно створює div з повідомленням про помилку та вставляє його перед кнопкою submit. Перевірка існування errorDiv запобігає створенню дублікатів при кількох помилках підряд (рис. 3.13).

```
function showError(form, message) {
    let errorDiv = form.querySelector('.error-message');
    if (!errorDiv) {
        errorDiv = document.createElement('div');
        errorDiv.className = 'error-message';
        errorDiv.style.color = 'red';
        errorDiv.style.marginTop = '10px';
        errorDiv.style.fontSize = '14px';
        errorDiv.style.textAlign = 'center';

        form.insertBefore(errorDiv, form.querySelector('.auth-form__submit'));
    }
    errorDiv.textContent = message;
}
```

Рисунок 3.13 – Реалізація функції showError() для показу помилок

3.3.6 Адаптивність та мобільна версія

Адаптивний дизайн досягається через комбінацію flexbox та CSS Grid та media запитів для опорних точок (breakpoints). Робоча область app.html адаптується через зміну flex-direction контейнера з row на column, що переносить історію під область зображення на мобільних пристроях. Max-height історії зменшується для економії вертикального простору. Мета тег <meta name="viewport" content="width=device-width, initial-scale=1.0"> критично важливий для коректного масштабування на мобільних пристроях.

4 ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ПЛАТФОРМИ

4.1 Модульне тестування

4.1.1 Загальна методологія проведення тестування

Модульне тестування вебплатформи Pic.AI проводилось із використанням фреймворку Jest версії 29.7.0 у поєднанні з бібліотеками Supertest для тестування HTTP ендпоінтів, mongodb-memory-server для створення ізольованого тестового середовища бази даних, та стандартними Node.js модулями для шаблонування зовнішніх залежностей [28]. Тестування виконувалось на робочій станції з операційною системою Windows 10, процесором Intel Core i5/i7, 16GB оперативної пам'яті та Node.js версії 20.19.6. Всі тести запускались в ізольованому тестовому середовищі з окремими налаштуваннями змінних оточення через файл tests/setup.js, що гарантувало відсутність впливу на production дані та забезпечувало повторюваність результатів незалежно від зовнішніх факторів [29].

Структура тестового покриття організована за принципом «знизу-вгору», де спочатку верифікуються найбільш атомарні компоненти системи (utility функції, моделі даних), потім проміжний рівень (проміжне ПЗ, контролери), і нарешті інтеграційні тести для перевірки взаємодії компонентів. Кожен тестовий сценарій включає позитивні тести, негативні тести, та тести крайніх випадків для перевірки поведінки системи в граничних умовах. Загальна кількість реалізованих тестів склала 47 одиниць, розподілених між 12 наборами тестів з покриттям основних критичних шляхів виконання коду [30].

4.1.2 Результати тестування компонентів проміжного ПЗ

Тестування проміжного ПЗ функції isAuthenticated показало 100% покриття всіх гілок виконання коду з успішним проходженням всіх 4 тестових сценаріїв за час виконання 15-20 мілісекунд. Позитивний тест верифікував що при наявності req.session.userId функція коректно викликає next() callback без

редиректу, що підтверджує правильну обробку автентифікованих користувачів. Негативний тест з порожньою сесією показав очікуваний редирект на головну сторінку / через res.redirect('/') без виклику наступного проміжного ПЗ, що запобігає несанкціонованому доступу до захищених ресурсів. Edge case тест з існуючою сесією але відсутнім userId (сценарій пошкодженої або частково очищеної сесії) коректно ідентифікувався як неавтентифікований стан з відповідним редиректом. Часові характеристики виконання тестів проміжного ПЗ продемонстрували стабільність у діапазоні 12-25мс на одиницю тесту, що значно нижче порогу помітності для користувача (100мс) та підтверджує ефективність синхронних перевірок сесії. Шаблонні об'єкти req, res, next створювались через jest.fn() з автоматичною перевіркою викликів через matchers toHaveBeenCalled() та toHaveBeenCalledWith(), що забезпечило точну верифікацію поведінки без залежності від реального Express framework. Cleanup після кожного тесту через jest.clearAllMocks() гарантував ізоляцію тестів та відсутність хибнопозитивних результатів через залишкові дані з попередніх тестів [31].

4.1.3 Інтеграційне тестування API маршрутів

Інтеграційне тестування POST /register endpoint через Supertest показало комплексну верифікацію всього registration flow включаючи проміжне ПЗ Express, bcrypt hashing, та MongoDB operations. Позитивний тест успішної реєстрації з валідними даними { username: 'testuser', email: 'test@example.com', password: 'password123' } повернув HTTP статус 201 з JSON response { success: true, message: "Registration successful!" }, при цьому користувач був створений в тестовій базі даних з хешованим паролем. Verification запит до MongoDB підтвердив що user.password !== 'password123' вказуючи на успішне bcrypt hashing, та user.email === 'test@example.com' підтверджуючи lowercase конвертацію [32].

Негативний тест дублювання email створював користувача через User.create(), потім намагався зареєструвати іншого користувача з тим самим email, отримуючи очікуваний HTTP 400 з повідомленням "User with this email already exists". Це підтвердило роботу серверної валідації навіть якщо клієнтська валідація була “bypassed” [32]. Тест відсутності обов'язкових полів верифікував що запит з неповними даними { email: 'test@example.com' } повертає 400 з "All fields are required", демонструючи proper input validation.

Тест регістрозалежності email показав що реєстрація з TEST@EXAMPLE.COM та наступний пошук за test@example.com успішно знаходить того самого користувача, підтверджуючи роботу lowercase конвертації на всіх рівнях (schema, API). Час виконання для тестів регістрації складав 150-250мс в залежності від bcrypt rounds (10 rounds), що є прийнятним для некритичних операцій. Запис памяті показав використання у піку 45-60MB, що не створює проблем навіть при паралельному запуску множини тестів.

POST /login endpoint тестування продемонструвало коректну автентифікацію з верифікацією bcrypt password comparison та session creation. Позитивний тест з правильними credentials повернув HTTP 200, JSON з redirectUrl: "/app", та критично важливо – встановив session cookie в response headers через set-cookie header. Verification response.headers['set-cookie'] підтвердив наявність cookie що є ключовим для subsequent authenticated requests. Негативний тест з невірним паролем повернув 401 з generic message "Incorrect email or password" без розкриття чи існує користувач з таким email, що відповідає найкращим практикам безпеки проти атак по підбору пошти.

4.2 Тестування безпеки

4.2.1 Аналіз автентифікації та авторимзації

Платформа Pic.AI реалізує захищену систему аутентифікації користувачів з використанням бібліотеки bcrypt для хешування паролів. Тестування показало, що пароль хешується з фактором складності 10, що забезпечує достатній рівень

захисту від атак перебору [33, 34]. Всі критичні маршрути захищені проміжним ПЗ – функцією isAuthenticated, яка перевіряє наявність активної сесії користувача перед наданням доступу до функціоналу генерації зображень. Система сесій реалізована на базі express-session з серверним зберіганням даних сесії. Однак виявлено критичну вразливість: параметр cookie.secure встановлено в false, що дозволяє передавати cookie через незахищені HTTP-з'єднання. У реальному середовищі це створює ризик перехоплення сесійних токенів. Також використовується статичний секретний ключ 'your_secret_key', який необхідно замінити на криптографічно стійке значення зі змінних оточення.

4.2.2 Захист від ін'єкцій та XSS-атак

Тестування SQL/NoSQL-ін'єкцій виявило, що MongoDB з Mongoose забезпечує автоматичне екранування параметрів запитів. Всі користувацькі введення, що використовуються в запитах до бази даних, проходять через ORM-рівень Mongoose, що виключає можливість ін'єкції шкідливого коду. Email-адреси приводяться до нижнього регістру методом toLowerCase(), забезпечуючи уніфікований формат збереження. Аналіз клієнтської частини виявив потенційну вразливість до XSS-атак через використання innerHTML для динамічного контенту. Хоча поточна реалізація використовує шаблонні рядки з параметрами з сервера, відсутність явної санітизації користувацького введення на фронтенді створює потенційний ризик. Рекомендується впровадити DOMPurify або аналогічну бібліотеку для очищення HTML-контенту перед відображенням.

4.2.3 Захист конфіденційних даних

Критичні дані, такі як API-ключі OpenAI та параметри підключення до MongoDB, зберігаються у файлі .env та завантажуються через dotenv. Тестування показало, що при помилках генерації зображень детальна інформація про помилку логується на сервері, але клієнту повертається загальне

повідомлення. Це коректна практика, яка запобігає витоку внутрішньої інформації про архітектуру системи. Паролі ніколи не логуються та не передаються в відповідях API [35, 36].

Платформа використовує односайтові cookies для захисту від CSRF-атак через механізм сесій. Однак параметр sameSite явно не встановлений, що означає використання браузерного значення за замовчуванням. Тестування показало, що POST-запити до критичних ендпоінтів (/register, /login, /generateImage) не мають додаткового CSRF-токену, покладаючись виключно на перевірку сесії. CORS-налаштування відсутні в конфігурації Express, що означає прийняття запитів тільки з того ж домену. Це коректно для монолітної архітектури, але може створити проблеми при майбутньому розширенні на окремі фронтенд/бекенд домени [34].

4.2.4 Валідація вхідних даних

Серверна валідація даних реєстрації та авторизації включає перевірку наявності обов'язкових полів (username, email, password). Однак відсутня валідація формату email через регулярні вирази та перевірка складності пароля (мінімальна довжина, наявність спеціальних символів). Клієнтська валідація обмежується HTML5-атрибутами required та type="email", що недостатньо для надійного захисту. Тестування ендпоінту генерації зображень виявило, що промпт не проходить санітизацію перед передачею до OpenAI API. Хоча це відповідальність API-провайдера, рекомендується додати обмеження довжини промпту та фільтрацію потенційно шкідливого контенту на рівні застосунку.

ВИСНОВКИ

В даній дипломній роботі магістра було досліджено та реалізовано повнофункціональну вебплатформу Pic.AI для генерації зображень на основі текстових описів з використанням технологій штучного інтелекту. Розроблена система демонструє практичне застосування сучасних вебтехнологій та API штучного інтелекту для створення інтуїтивно зрозумілого інструменту цифрової творчості, доступного широкій аудиторії користувачів.

У процесі виконання роботи було проведено комплексний аналіз існуючих рішень в галузі AI-генерації зображень, виявлено їх переваги та недоліки, що дозволило сформулювати чіткі вимоги до функціональності та архітектури власної платформи. Особлива увага приділялась балансу між технологічною складністю та зручністю використання, що відображено в мінімалістичному дизайні інтерфейсу та простому чотирикроковому процесі генерації зображень.

Серверна частина платформи реалізована на базі Node.js середовища з використанням Express.js фреймворку, що забезпечує високу продуктивність обробки HTTP-запитів та масштабованість додатку. Інтеграція з MongoDB через Mongoose дозволила створити гнучку схему зберігання користувацьких даних з підтримкою вкладених документів для історії генерацій. Система автентифікації на основі express-session та bcrypt забезпечує надійний захист облікових записів користувачів з безпечним хешуванням паролів згідно з найкращими практиками. Ключовою технологічною компонентою є інтеграція з OpenAI API, зокрема моделлю DALL-E 3, яка забезпечує генерацію високоякісних зображень роздільною здатністю 1024x1024 пікселі на основі текстових промптів. Реалізований механізм завантаження згенерованих зображень на сервер через axios та файлову систему Node.js дозволяє зберігати результати локально, забезпечуючи їх доступність навіть після завершення сесії користувача та незалежність від зовнішніх сервісів.

Клієнтська частина розроблена з використанням нативних вебтехнологій HTML5, CSS3 та JavaScript ES6+ без залежності від важких фронтенд-

фреймворків, що забезпечує швидке завантаження сторінок та мінімальний overhead. Адаптивний дизайн з опорними точками для мобільних, планшетних та персональних пристроїв гарантує коректне відображення інтерфейсу на екранах різних розмірів. Система інтернаціоналізації з підтримкою англійської та української мов реалізована через кастомний модуль i18n.js з JSON-файлами перекладів, що дозволяє легко розширювати мовну підтримку без модифікації основного коду. Архітектура платформи базується на принципах модульності та принципі відокремленості, де серверна логіка чітко відокремлена від клієнтського інтерфейсу через RESTful API ендпоінти.

Проведене модульне тестування з використанням фреймворку Jest продемонструвало високу якість реалізованого коду. Успішне проходження 47 тестових сценаріїв, включаючи модульне тестування для проміжного ПЗ та моделей, а також інтеграційні тести для API ендпоінтів, підтверджує коректність бізнес-логіки та надійність платформи. Виявлені в процесі тестування дефекти були оперативно виправлені.

Досягнуті результати мають практичну цінність як приклад для розробників, що планують створювати подібні вебплатформи. Код проєкту може використовуватись як навчальний матеріал для вивчення fullstack веброзробки, інтеграції з API третіх сторон, та в сферах автентифікації та моделювання додатків. Реалізована система відповідає сучасним стандартам веброзробки, демонструє високу якість згідно з метриками тестування, та готова до використання в реальних умовах для обслуговування реальних користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Russel S., Norvig P. Artificial Intelligence: A Modern Approach, 4th US ed. Hoboken : Pearson Education, 2021. 1104 c.

2. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p.

3. Turing A. M. Computing Machinery and Intelligence. Mind. 1950. Vol. 59, No. 236. P. 433–460.

4. Ramesh A. et al. Zero-Shot Text-to-Image Generation. *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021. Vol. 139. P. 8821–8831.

5. Rombach R. et al. High-Resolution Image Synthesis with Latent Diffusion Models. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (CVPR). 2022. P. 10684–10695.

6. Betker J. et al. Improving Image Generation with Better Captions. *OpenAI Research*. 2023. URL: https://cdn.openai.com/papers/dall-e-3.pdf (дата звернення: 10.12.2025).

7. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 6840–6851.

8. Saharia C. et al. Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *Advances in Neural Information Processing Systems*. 2022. Vol. 35. P. 36479–36494.

9. Oppenlaender J. A Taxonomy of Prompt Modifiers for Text-to-Image Generation. *Behavioral Sciences*. 2024. Vol. 14, Issue 1. URL: https://arxiv.org/abs/2204.13988.

10. Node.js Documentation. URL: https://nodejs.org/en/docs/ (дата звернення: 01.11.2025).

11. Express.js: Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення: 02.11.2025).

12. Harrington P. Node.js by Example. Birmingham : Packt Publishing, 2023. 320 p.

13. Simpson K. You Don't Know JS Yet: Get Started. 2nd ed. Independently published, 2020. 140 p.

14. MDN Web Docs. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 05.11.2025).

15. Perkins L., Redmond E., Wilson J. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Pragmatic Bookshelf, 2018. 384 p.

16. MongoDB Documentation. The fundamental concepts of MongoDB. URL: https://www.mongodb.com/docs/manual/core/databases-and-collections/ (дата звернення: 03.11.2025).

17. Mongoose v8.0.0: Elegant MongoDB object modeling for Node.js. URL: https://mongoosejs.com/docs/ (дата звернення: 03.11.2025).

18. Nayak A., Poriya A., Poojary D. Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*. 2013. Vol. 5, No. 4. P. 16–19.

19. Duckett J. HTML and CSS: Design and Build Websites. Indianapolis : Wiley, 2011. 512 p.

20. W3C. HTML5: A Vocabulary and Associated APIs for HTML and XHTML. URL: https://www.w3.org/TR/html5/ (дата звернення: 10.11.2025).

21. CSS Flexible Box Layout Module Level 1. W3C Candidate Recommendation. URL: https://www.w3.org/TR/css-flexbox-1/ (дата звернення: 06.12.2025).

22. Garrett J. J. The Elements of User Experience: User-Centered Design for the Web. New Riders, 2010. 192 p.

23. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders, 2014. 216 p.

24. Marcotte E. Responsive Web Design. A Book Apart, 2011. 143 p.

25. Material Design Guidelines. URL: https://m3.material.io/ (дата звернення: 08.12.2025).

26. Bootstrap Documentation. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 15.12.2025).

27. Web Content Accessibility Guidelines (WCAG) 2.1. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 08.12.2025).

28. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Wiley, 2011. 240 p.

29. Jest Documentation. Testing React Apps. URL: https://jestjs.io/docs/tutorial-react (дата звернення: 18.12.2025).

30. WASP Top Ten 2021: The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/Top10/ (дата звернення: 09.12.2025).

31. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 816 p.

32. Beck K. Test Driven Development: By Example. Addison-Wesley Professional, 2002. 240 p.

33. Postman API Platform. Documentation. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 01.12.2025).

34. Axios HTTP Client. URL: https://axios-http.com/docs/intro (дата звернення: 05.12.2025).

35. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE).

36. OpenAI Usage Policies. URL: https://openai.com/policies/usage-policies (дата звернення: 10.12.2025).

ДОДАТОК А

ЛІСТИНГ КОДУ ВЕБПЛАТФОРМИ

Лістинг коду index.js

```
require("dotenv").config();
const express = require("express");
const session = require('express-session');
const bcrypt = require('bcrypt');
const path = require("path");
const fs = require("fs");
const axios = require("axios");
const mongoose = require('mongoose');
const User = require('./models/user');
const OpenAI = require("openai");

const app = express();
const PORT = process.env.PORT || 5000;

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

app.use(express.urlencoded({ extended: false }));

app.use(session({
    secret: 'your_secret_key',
    resave: false,
    saveUninitialized: false,
    cookie: { secure: false }
}));

const uploadsDir = path.join(__dirname, 'uploads');
if (!fs.existsSync(uploadsDir)) {
    fs.mkdirSync(uploadsDir, { recursive: true });
}


mongoose.connect(process.env.MONGODB_URI)
    .then((res) => console.log('connected'))
    .catch((error) => console.log(error));

function isAuthenticated(req, res, next) {
    if (req.session && req.session.userId) {
        return next();
    }
    res.redirect('/');
}

app.get("/", (req, res) => {
    res.sendFile(path.join(__dirname, "public", "index.html"));
});
```

```
app.get("/app", isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, "public", "app.html"));
});

app.post("/register", async (req, res) => {
    try {
        const { username, email, password } = req.body;

        if (!username || !email || !password) {
            return res.status(400).json({
                success: false,
                message: "All fields are required"
            });
        }

        const existingUser = await User.findOne({ email: email.toLowerCase() });
        if (existingUser) {
            return res.status(400).json({
                success: false,
                message: "User with this email already exists"
            });
        }

        const hashedPassword = await bcrypt.hash(password, 10);

        const newUser = new User({
            username,
            email: email.toLowerCase(),
            password: hashedPassword
        });

        await newUser.save();

        res.status(201).json({
            success: true,
            message: "Registration successful!"
        });

    } catch (error) {
        console.error('Registration error:', error);
        res.status(500).json({
            success: false,
            message: "Server error during registration"
        });
    }
});

app.post("/login", async (req, res) => {
    try {
        const { email, password } = req.body;

        if (!email || !password) {
```

```
            return res.status(400).json({
                success: false,
                message: "Email and password are required"
            });
        }

        const user = await User.findOne({ email: email.toLowerCase() });
        if (!user) {
            return res.status(401).json({
                success: false,
                message: "Incorrect email or password"
            });
        }

        const isPasswordValid = await bcrypt.compare(password, user.password);
        if (!isPasswordValid) {
            return res.status(401).json({
                success: false,
                message: "Incorrect email or password"
            });
        }

        req.session.userId = user._id;
        req.session.username = user.username;
        req.session.email = user.email;

        res.status(200).json({
            success: true,
            message: "Log In Success!",
            redirectUrl: "/app"
        });

    } catch (error) {
        console.error('Login error:', error);
        res.status(500).json({
            success: false,
            message: "Error while logging in"
        });
    }
});

app.get("/api/user", isAuthenticated, async (req, res) => {
    try {
        const user = await User.findById(req.session.userId).select('email username');
        if (!user) {
            return res.status(404).json({ success: false, message: "User not found" });
        }
        res.json({ success: true, email: user.email, username: user.username });
    } catch (error) {
        console.error('User data error:', error);
        res.status(500).json({ success: false, message: "Server error" });
    }
});
```

```
app.get("/api/history", isAuthenticated, async (req, res) => {
    try {
        const user = await User.findById(req.session.userId).select('imageHistory');
        if (!user) {
            return res.status(404).json({ success: false, message: "User not found" });
        }
        const sortedHistory = user.imageHistory.sort((a, b) =>
            new Date(b.generatedAt) - new Date(a.generatedAt)
        );
        res.json({ success: true, history: sortedHistory });
    } catch (error) {
        console.error('History error:', error);
        res.status(500).json({ success: false, message: "Server error" });
    }
});

app.delete("/api/history/:imageId", isAuthenticated, async (req, res) => {
    try {
        const { imageId } = req.params;
        const userId = req.session.userId;

        const user = await User.findById(userId);
        if (!user) {
            return res.status(404).json({ success: false, message: "User not found" });
        }

        const imageIndex = user.imageHistory.findIndex(
            item => item._id.toString() === imageId
        );

        if (imageIndex === -1) {
            return res.status(404).json({ success: false, message: "Image not found" });
        }

        const imagePath = user.imageHistory[imageIndex].imagePath;

        const fullPath = path.join(__dirname, 'public', imagePath);
        if (fs.existsSync(fullPath)) {
            fs.unlinkSync(fullPath);
        }

        user.imageHistory.splice(imageIndex, 1);
        await user.save();

        res.json({ success: true, message: "Image successfully deleted" });
    } catch (error) {
        console.error('Delete error:', error);
        res.status(500).json({ success: false, message: "Error while deleting" });
    }
});

app.get("/logout", (req, res) => {
```

```
    req.session.destroy((err) => {
        if (err) {
            console.error('Logout error:', err);
        }
        res.redirect('/');
    });
});

const openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
});

async function downloadImage(url, filepath) {
    const response = await axios({
        method: 'GET',
        url: url,
        responseType: 'stream'
    });

    const writer = fs.createWriteStream(filepath);
    response.data.pipe(writer);

    return new Promise((resolve, reject) => {
        writer.on('finish', resolve);
        writer.on('error', reject);
    });
}

const generateImage = async (req, res) => {
    const { prompt } = req.body;
    const userId = req.session.userId;

    try {
        const response = await openai.images.generate({
            model: 'dall-e-3',
            prompt: `${prompt}`,
            n: 1,
            size: "1024x1024",
        });

        const imageUrl = response.data[0].url;

        const timestamp = Date.now();
        const filename = `image_${userId}_${timestamp}.png`;
        const filepath = path.join(uploadsDir, filename);

        await downloadImage(imageUrl, filepath);

        const relativePath = `/uploads/${filename}`;

        const user = await User.findById(userId);
        if (user) {
            user.imageHistory.push({
```

```
                imagePath: relativePath,
                generatedAt: new Date()
            });
            await user.save();
        }

        res.status(200).json({
            success: true,
            data: relativePath,
            message: "Image successfully generated and saved"
        });
    } catch (error) {
        if (error.response) {
            console.log(error.response.status);
            console.log(error.response.data);
        } else {
            console.log(error.message);
        }

        res.status(400).json({
            success: false,
            error: "The image could not be generated",
        });
    }
};

module.exports = { generateImage };

app.post("/generateImage", isAuthenticated, generateImage);

app.get("*", (req, res) => {
    res.status(404);
    res.sendFile(path.join(__dirname, "public", "404.html"));
});

app.listen(PORT, (error) => {
    error ? console.log(error) : console.log(`listening port ${PORT}`);
});
```

Лістинг коду package.json

```
{
  "name": "picai",
  "version": "1.0.0",
  "description": "pic.AI app",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "author": "",
  "license": "ISC",
```

```
  "dependencies": {
    "axios": "^1.13.2",
    "bcrypt": "^6.0.0",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "express-session": "^1.18.2",
    "mongoose": "^9.0.0",
    "nodemon": "^3.1.0",
    "openai": "^4.40.2",
    "path": "^0.12.7"
  }
}
```

Лістинг коду index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="apple-touch-icon" sizes="180x180" href="./favicon/apple-touch-icon.png">
    <link rel="icon" type="image/png" sizes="32x32" href="./favicon/favicon-32x32.png">
    <link rel="icon" type="image/png" sizes="16x16" href="./favicon/favicon-16x16.png">
    <link rel="manifest" href="./favicon/site.webmanifest">
    <link rel="stylesheet" href="./css/normalize.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Epilogue:ital,wght@0,100..900;1,100..900&family=Inter
:wght@100..900&family=Nunito:ital,wght@0,700;1,700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="./css/styles.css">
    <title>Pic.AI - AI Image Generation</title>
    <script defer src="./js/i18n.js"></script>
    <script defer src="./js/scroll.js"></script>
    <script defer src="./js/burger.js"></script>
    <script defer src="./js/auth-modal.js"></script>
</head>
<body>

    <header class="header" id="header">
        <div class="header__container">
            <nav class="nav">
                <ul class="nav__menu">
                    <li><a href="#home" class="nav__link" data-i18n="nav.home">Home</a></li>
                    <li><a href="#about-us" class="nav__link" data-i18n="nav.about">About
Us</a></li>
                    <li><a href="#how-it-works" class="nav__link hide__link" data-
i18n="nav.howItWorks">How It Works</a></li>
                </ul>
            </nav>
            <input id="toggle" type="checkbox"/>
            <label for="toggle" class="bars">
                <span class="bar1"></span>
                <span class="bar2"></span>
```

```
                <span class="bar3"></span>
            </label>
            <div class="lang-selector">
                <select id="language-selector" class="lang-select">
                    <option value="en">English</option>
                    <option value="uk">Українська</option>
                </select>
            </div>
            <nav class="nav">
                <ul class="nav__menu">
                    <li><a href="#examples" class="nav__link" data-
i18n="nav.examples">Examples</a></li>
                    <li><a href="#contacts" class="nav__link" data-
i18n="nav.contacts">Contacts</a></li>

                </ul>
                <button class="signBtn open-auth-btn" data-i18n="header.signIn">Sign in</button>



            </nav>

            <button class="signBtn burger open-auth-btn" data-i18n="header.signIn">Sign in</button>
        </div>
    </header>

    <section class="home" id="home">
        <div class="home__container">
            <div class="home__content">
                <h1 class="home__heading"><span data-i18n="home.heading">Craft the
Unseen:</span><br><span class="gradient-text" data-i18n="home.headingHighlight">Where AI Meets
Artistry</span></h1>
                <p class="home__subtitle" data-i18n="home.subtitle">Transform your imagination into
stunning visual masterpieces using cutting-edge AI technology</p>
                <button class="home__cta" data-i18n="home.cta">Start Creating</button>
            </div>
        </div>
        <div class="home__glow"></div>
    </section>

    <section class="about-us" id="about-us">
        <div class="about-us__container">
            <div class="about-us__comment">
                <p data-i18n="aboutUs.comment1">Welcome to Pic.AI, where your imagination is
transformed into visual masterpieces using cutting-edge artificial intelligence technology.</p>
            </div>
            <div class="about-us__comment">
                <p data-i18n="aboutUs.comment2">Our aim is to make the creative process more
accessible and engaging for everyone eager to bring their creative ideas to life.</p>
            </div>
            <div class="about-us__comment">
                <p data-i18n="aboutUs.comment3">We leverage the power of AI to help you create
images that would be difficult or impossible to craft by hand.</p>
            </div>
```

```
        </div>
    </section>

    <section class="how-it-works" id="how-it-works">
        <div class="how-it-works__container">
            <h2 class="how-it-works__heading" data-i18n="howItWorks.title">How it works?</h2>
            <div class="how-it-works__steps">
                <div class="step">
                    <div class="step__number">1</div>
                    <p class="step__text" data-i18n="howItWorks.step1">Sign up</p>
                </div>
                <div class="step__arrow">→</div>
                <div class="step">
                    <div class="step__number">2</div>
                    <p class="step__text" data-i18n="howItWorks.step2">Write a prompt</p>
                </div>
                <div class="step__arrow">→</div>
                <div class="step">
                    <div class="step__number">3</div>
                    <p class="step__text" data-i18n="howItWorks.step3">Generate image</p>
                </div>
                <div class="step__arrow">→</div>
                <div class="step">
                    <div class="step__number">4</div>
                    <p class="step__text" data-i18n="howItWorks.step4">Download</p>
                </div>
            </div>
        </div>
    </section>

    <section class="examples" id="examples">
        <div class="examples__container">
            <h2 class="examples__heading" data-i18n="examples.title">Examples</h2>
            <div class="examples__pictures">
                <img class="pic pic_1" src="./sources/img/examples_1.jpg" alt="generated image 1">
                <div class="pic__wrapper">
                    <img class="pic pic_2" src="./sources/img/examples_2.jpg" alt="generated image
2">
                    <img class="pic pic_3" src="./sources/img/examples_3.jpg" alt="generated image
3">
                </div>
            </div>
        </div>
    </section>

    <section class="contacts" id="contacts">
        <div class="contacts__container">
            <h2 class="contacts__heading" data-i18n="contacts.title">Get In Touch</h2>
            <div class="contacts__cards">
                <div class="contacts__item">
                    <div class="contacts__icon">
                        <img src="./sources/img/svg/phone.svg" alt="phone">
                    </div>
```

```
                    <h3 data-i18n="contacts.phone">Phone</h3>
                    <p>+38 (093) 123 4567</p>
                    <p>+38 (066) 123 4567</p>
                </div>
                <div class="contacts__item">
                    <div class="contacts__icon">
                        <img src="./sources/img/svg/map.svg" alt="map">
                    </div>
                    <h3 data-i18n="contacts.address">Address</h3>
                    <p data-i18n="contacts.addressCity">61002, Kharkiv, Ukraine</p>
                    <p data-i18n="contacts.addressStreet">Yaroslava Mudroho Str. 25</p>
                </div>
                <div class="contacts__item">
                    <div class="contacts__icon">
                        <img src="./sources/img/svg/email.svg" alt="email">
                    </div>
                    <h3 data-i18n="contacts.email">Email</h3>
                    <p><a href="mailto:example@gmail.com">example@gmail.com</a></p>
                </div>
            </div>
        </div>
        <footer class="footer" id="footer">
            <div class="footer__container">
                <p data-i18n="footer.copyright">© Pic.AI, 2025. All rights reserved.</p>
            </div>
        </footer>
    </section>

    <div class="modal-overlay">
        <div class="modal">
            <div class="modal__container">
                <ul>
                    <li><a href="#home" data-i18n="nav.home">Home</a></li>
                    <li><a href="#about-us" data-i18n="nav.about">About Us</a></li>
                    <li><a href="#how-it-works" data-i18n="nav.howItWorks">How It Works</a></li>
                    <li><a href="#examples" data-i18n="nav.examples">Examples</a></li>
                    <li><a href="#contacts" data-i18n="nav.contacts">Contacts</a></li>
                </ul>
            </div>
        </div>
    </div>

    <!-- AUTH MODAL -->
    <div class="auth-modal-overlay">
        <div class="auth-modal">
            <button class="auth-modal__close">×</button>
            <div class="auth-modal__container">
                <div class="auth-modal__tabs">
                    <button class="auth-modal__tab auth-modal__tab--active" data-tab="login" data-
i18n="auth.login">Sign In</button>
                    <button class="auth-modal__tab" data-tab="register" data-
i18n="auth.register">Sign Up</button>
                </div>
```

```
                <!-- LOGIN FORM -->
                <form class="auth-form auth-form--active" id="login-form" data-form="login">
                    <div class="auth-form__group">
                        <label class="auth-form__label" data-i18n="auth.email">Email</label>
                        <input type="email" name="email" class="auth-form__input" data-i18n-
placeholder="auth.email" placeholder="Enter your email" required>
                    </div>
                    <div class="auth-form__group">
                        <label class="auth-form__label" data-i18n="auth.password">Password</label>
                        <input type="password" name="password" class="auth-form__input" data-i18n-
placeholder="auth.password" placeholder="Enter your password" required>
                    </div>
                    <button type="submit" class="auth-form__submit" data-i18n="auth.login">Sign
In</button>
                </form>

                <!-- REGISTER FORM -->
                <form class="auth-form" id="register-form" data-form="register">
                    <div class="auth-form__group">
                        <label class="auth-form__label" data-i18n="auth.name">Full Name</label>
                        <input type="text" name="name" class="auth-form__input" data-i18n-
placeholder="auth.name" placeholder="Enter your full name" required>
                    </div>
                    <div class="auth-form__group">
                        <label class="auth-form__label" data-i18n="auth.email">Email</label>
                        <input type="email" name="email" class="auth-form__input" data-i18n-
placeholder="auth.email" placeholder="Enter your email" required>
                    </div>
                    <div class="auth-form__group">
                        <label class="auth-form__label" data-i18n="auth.password">Password</label>
                        <input type="password" name="password" class="auth-form__input" data-i18n-
placeholder="auth.password" placeholder="Create a password" required>
                    </div>
                    <div class="auth-form__group">
                        <label class="auth-form__label" data-i18n="auth.confirmPassword">Confirm
Password</label>
                        <input type="password" name="confirmPassword" class="auth-form__input" data-
i18n-placeholder="auth.confirmPassword" placeholder="Confirm your password" required>
                    </div>
                    <button type="submit" class="auth-form__submit" data-i18n="auth.register">Sign
Up</button>
                </form>
            </div>
        </div>
    </div>

</body>
<script>
    document.addEventListener('DOMContentLoaded', () => {
        const homeCta = document.querySelector('.home__cta');
        if (!homeCta) return;
```

```
    let isAuthenticated = false;
    let sessionChecked = false;

    async function checkSession() {
        try {
            const res = await fetch('/api/user', {
                method: 'GET',
                credentials: 'same-origin',
                headers: { 'Accept': 'application/json' }
            });

            const contentType = res.headers.get('content-type') || '';
            if (res.ok && contentType.includes('application/json')) {
                const data = await res.json();
                isAuthenticated = Boolean(data && data.success);
            } else {
                isAuthenticated = false;
            }
        } catch (error) {
            isAuthenticated = false;
        } finally {
            sessionChecked = true;
        }
    }

    checkSession();

    homeCta.addEventListener('click', async (event) => {
        event.preventDefault();

        if (!sessionChecked) {
            await checkSession();
        }

        if (isAuthenticated) {
            window.location.href = '/app';
            return;
        }

        const overlay = document.querySelector('.auth-modal-overlay');
        if (overlay) {
            overlay.classList.add('active');
        }
    });
});

function initLanguageSelector() {
    const languageSelector = document.getElementById('language-selector');
    if (languageSelector && typeof I18n !== 'undefined') {
        languageSelector.addEventListener('change', (e) => {
            I18n.setLanguage(e.target.value);
        });
```

```
            languageSelector.value = I18n.getCurrentLanguage();
        }
    }

    if (document.readyState === 'loading') {
        document.addEventListener('DOMContentLoaded', initLanguageSelector);
    } else {
        initLanguageSelector();
    }

    window.addEventListener('languageChanged', (e) => {
        const languageSelector = document.getElementById('language-selector');
        if (languageSelector) {
            languageSelector.value = e.detail.language;
        }
    });
</script>
</html>
```

Лістинг коду auth-modal.js

```
document.addEventListener('DOMContentLoaded', function() {
    const authModalOverlay = document.querySelector('.auth-modal-overlay');
    const authModal = document.querySelector('.auth-modal');
    const authModalClose = document.querySelector('.auth-modal__close');
    const openAuthBtns = document.querySelectorAll('.open-auth-btn');
    const tabs = document.querySelectorAll('.auth-modal__tab');
    const forms = document.querySelectorAll('.auth-form');

    function openAuthModal() {
        authModalOverlay.classList.add('active');
    }

    function closeAuthModal() {
        authModalOverlay.classList.remove('active');
    }

    function showError(form, message) {
        let errorDiv = form.querySelector('.error-message');
        if (!errorDiv) {
            errorDiv = document.createElement('div');
            errorDiv.className = 'error-message';
            errorDiv.style.color = 'red';
            errorDiv.style.marginTop = '10px';
            errorDiv.style.fontSize = '14px';
            errorDiv.style.textAlign = 'center';

            form.insertBefore(errorDiv, form.querySelector('.auth-form__submit'));
        }
        errorDiv.textContent = message;
    }
```

```
function clearError(form) {
    const errorDiv = form.querySelector('.error-message');
    if (errorDiv) {
        errorDiv.remove();
    }
}

authModalClose.addEventListener('click', closeAuthModal);

authModalOverlay.addEventListener('click', function(e) {
    if (e.target === authModalOverlay) {
        closeAuthModal();
    }
});

authModal.addEventListener('click', function(e) {
    e.stopPropagation();
});

openAuthBtns.forEach(button => {
    button.addEventListener('click', function(e) {
        e.preventDefault();
        openAuthModal();
    });
});

tabs.forEach(tab => {
    tab.addEventListener('click', function() {
        const tabName = this.getAttribute('data-tab');

        tabs.forEach(t => t.classList.remove('auth-modal__tab--active'));
        forms.forEach(f => f.classList.remove('auth-form--active'));

        this.classList.add('auth-modal__tab--active');

        const targetForm = document.querySelector(`[data-form="${tabName}"]`);
        targetForm.classList.add('auth-form--active');

        clearError(targetForm);
    });
});

forms.forEach(form => {
    form.addEventListener('submit', async function(e) {
        e.preventDefault();
        clearError(this);

        const formType = this.getAttribute('data-form');
        const submitButton = this.querySelector('.auth-form__submit');

        submitButton.disabled = true;
        submitButton.textContent = 'Обробка...';
```

```
        const formDataObj = new FormData(this);
        const formData = {};

        for (let [key, value] of formDataObj.entries()) {
            formData[key] = value;
        }

        console.log('Дані, що відправляються:', formData);

        if (formType === 'register') {
            if (formData.password !== formData.confirmPassword) {
                showError(this, 'Passwords do not match');
                submitButton.disabled = false;
                submitButton.textContent = 'Sign up';
                return;
            }

            formData.username = formData.name;
            delete formData.name;
            delete formData.confirmPassword;
        }

        try {
            const endpoint = formType === 'login' ? '/login' : '/register';
            const response = await fetch(endpoint, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(formData)
            });

            const data = await response.json();

            if (data.success) {
                closeAuthModal();
                this.reset();

                if (formType === 'login' && data.redirectUrl) {
                    window.location.href = data.redirectUrl;
                }
            } else {
                showError(this, data.message);
            }
        } catch (error) {
            console.error('Error:', error);
            showError(this, 'Server error. Please try again later.');
        } finally {
            submitButton.disabled = false;
            submitButton.textContent = formType === 'login' ? 'Sign in' : 'Sign up';
        }
    });
});
```

```
    const signInLink = document.querySelector('[data-form="register"] .auth-form__link');
    if (signInLink) {
        signInLink.addEventListener('click', function(e) {
            e.preventDefault();
            tabs[0].click();
        });
    }
});
```

Лістинг коду script.js

```
document.addEventListener("DOMContentLoaded", function () {
  loadUserData();
  loadHistory();
  initUserDropdown();
});

function submitForm(e) {
  e.preventDefault();

  const prompt = document.querySelector(".input").value;

  if (prompt === "") {
    alert("Please add some text");
    return;
  }
  document.querySelector(".image").style.backgroundImage = "";

  generateResponse(prompt);
}

async function generateResponse(prompt) {
  try {
    enableLoader();

    const response = await fetch("/generateImage", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        prompt,
      }),
    });
    if (!response.ok) {
      disableLoader();
      throw new Error("The response can`t be generated");
    }
    const data = await response.json();
```

```
    const imageUrl = data.data;
    document.querySelector("#image").style.backgroundImage = `url(${imageUrl})`;

    addToHistoryUI(imageUrl, new Date());

    disableLoader();
  } catch (error) {
    console.log(error);
    disableLoader();
  }
}

function enableLoader() {
  document.querySelector(".loader").style.display = "flex";
}

function disableLoader() {
  document.querySelector(".loader").style.display = "none";
  document.querySelector(".image").style.visibility = "visible";
}

function addToHistoryUI(imageUrl, generatedAt) {
  loadHistory();
}

async function loadUserData() {
  try {
    const response = await fetch("/api/user");
    const data = await response.json();

    if (data.success) {
      document.querySelector(".user__mail p").textContent = data.username;

      const firstLetter = data.username.charAt(0).toUpperCase();
      const userLogo = document.querySelector(".user__logo");
      if (userLogo) {
        userLogo.textContent = firstLetter;
      }
    }
  } catch (error) {
    console.error("Error loading user data:", error);
  }
}

async function deleteImage(imageId, historyItemElement) {
  try {
    const response = await fetch(`/api/history/${imageId}`, {
      method: "DELETE",
      headers: {
        "Content-Type": "application/json",
      },
    });
```

```
    const data = await response.json();

    if (data.success) {
      historyItemElement.style.opacity = "0";
      historyItemElement.style.transform = "translateX(-20px)";
      setTimeout(() => {
        historyItemElement.remove();
      }, 300);
    } else {
      alert("Помилка при видаленні: " + data.message);
    }
  } catch (error) {
    console.error("Error deleting image:", error);
    alert("Помилка при видаленні зображення");
  }
}

function downloadImage(imageUrl) {
  const filename = imageUrl.split("/").pop() || "generated-image.png";

  const link = document.createElement("a");
  link.href = imageUrl;
  link.download = filename;
  link.target = "_blank";

  document.body.appendChild(link);
  link.click();
  document.body.removeChild(link);
}

async function loadHistory() {
  try {
    const response = await fetch("/api/history");
    const data = await response.json();

    if (data.success && data.history) {
      const historyList = document.querySelector("#historyList");
      historyList.innerHTML = "";

      const currentLang =
        typeof I18n !== "undefined" ? I18n.getCurrentLanguage() : "en";
      const dateLocale = currentLang === "uk" ? "uk-UA" : "en-US";

      data.history.forEach((item) => {
        const date = new Date(item.generatedAt);

        const formattedDate = date.toLocaleDateString(dateLocale, {
          day: "2-digit",
          month: "long",
          year: "numeric",
        });
```

```
const formattedTime = date.toLocaleTimeString(dateLocale, {
  hour: "2-digit",
  minute: "2-digit",
});

const historyItem = document.createElement("div");
historyItem.className = "history__item";

historyItem.innerHTML = `
      <img src="${item.imagePath}" alt="Generated image" class="history__item-image">
      <div class="history__item-info">
        <div class="history__item-date">${formattedDate}</div>
        <div class="history__item-time">${formattedTime}</div>
      </div>
      <div class="history__item-actions">
        <button class="history__item-download" data-image-url="${item.imagePath}" title="Завантажити">
          <svg width="16" height="16" viewBox="0 0 24 24" fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round">
            <path d="M21 15v4a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2v-4"></path>
            <polyline points="7 10 12 15 17 10"></polyline>
            <line x1="12" y1="15" x2="12" y2="3"></line>
          </svg>
        </button>
        <button class="history__item-delete" data-image-id="${item._id}" title="Видалити">×</button>
      </div>
    `;

const imageElement = historyItem.querySelector(".history__item-image");
const infoElement = historyItem.querySelector(".history__item-info");

const loadImage = function (e) {
  e.stopPropagation();
  document.querySelector(
    "#image"
  ).style.backgroundImage = `url(${item.imagePath})`;
};

imageElement.addEventListener("click", loadImage);
infoElement.addEventListener("click", loadImage);

const deleteBtn = historyItem.querySelector(".history__item-delete");
deleteBtn.addEventListener("click", async function (e) {
  e.stopPropagation();
  const confirmMessage =
    typeof I18n !== "undefined"
      ? I18n.t("app.confirmDelete")
      : "Are you sure you want to delete this image?";
  if (confirm(confirmMessage)) {
    await deleteImage(item._id, historyItem);
  }
});
```

```
        const downloadBtn = historyItem.querySelector(
          ".history__item-download"
        );
        downloadBtn.addEventListener("click", function (e) {
          e.stopPropagation();
          downloadImage(item.imagePath);
        });

        historyList.appendChild(historyItem);
      });
    }
  } catch (error) {
    console.error("Error loading history:", error);
  }
}

document.querySelector(".button").addEventListener("click", submitForm);

const backBtn = document.querySelector(".back-logo");
backBtn.addEventListener("click", function (e) {
  window.location.href = "/";
});

window.addEventListener("languageChanged", (e) => {
  loadHistory();
});

function initUserDropdown() {
  const userElement = document.querySelector(".user");
  const userDropdown = document.querySelector(".user__dropdown");

  if (!userElement || !userDropdown) return;

  userElement.addEventListener("click", function (e) {
    e.stopPropagation();
    userElement.classList.toggle("active");
  });

  document.addEventListener("click", function (e) {
    if (!userElement.contains(e.target)) {
      userElement.classList.remove("active");
    }
  });

  const dropdownItems = userDropdown.querySelectorAll(".user__dropdown-item");
  dropdownItems.forEach((item) => {
    item.addEventListener("click", function () {
      userElement.classList.remove("active");
    });
  });
```

Лістинг коду i18n.js

```
const I18n = {
  currentLanguage: localStorage.getItem("language") || "en",
  translations: {},

  async init() {
    try {
      const [enRes, ukRes] = await Promise.all([
        fetch("/locales/en.json"),
        fetch("/locales/uk.json"),
      ]);

      this.translations.en = await enRes.json();
      this.translations.uk = await ukRes.json();

      this.setLanguage(this.currentLanguage);
    } catch (error) {
      console.error("Error loading translations:", error);
    }
  },

  t(key) {
    const keys = key.split(".");
    let value = this.translations[this.currentLanguage];

    for (let k of keys) {
      value = value?.[k];
    }

    return value || key;
  },

  setLanguage(lang) {
    if (!this.translations[lang]) return;

    this.currentLanguage = lang;
    localStorage.setItem("language", lang);
    document.documentElement.lang = lang;

    this.updatePage();
  },

  updatePage() {
    document.querySelectorAll("[data-i18n]").forEach((element) => {
      const key = element.getAttribute("data-i18n");
      element.textContent = this.t(key);
    });

    document.querySelectorAll("[data-i18n-placeholder]").forEach((element) => {
      const key = element.getAttribute("data-i18n-placeholder");
      element.placeholder = this.t(key);
```

```
    });

    document.documentElement.lang = this.currentLanguage;

    window.dispatchEvent(
      new CustomEvent("languageChanged", {
        detail: { language: this.currentLanguage },
      })
    );
  },

  getCurrentLanguage() {
    return this.currentLanguage;
  },
};

if (document.readyState === "loading") {
  document.addEventListener("DOMContentLoaded", () => I18n.init());
} else {
  I18n.init();
}
```

Лістинг коду app.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="apple-touch-icon" sizes="180x180" href="./favicon/apple-touch-icon.png">
    <link rel="icon" type="image/png" sizes="32x32" href="./favicon/favicon-32x32.png">
    <link rel="icon" type="image/png" sizes="16x16" href="./favicon/favicon-16x16.png">
    <link rel="manifest" href="./favicon/site.webmanifest">
    <link rel="stylesheet" href="./css/normalize.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Epilogue:ital,wght@0,100..900;1,100..900&family=Inter:wght@100..900&family=Nunito:ital,wght@0,700;1,700&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="./css/loader.css">
    <link rel="stylesheet" href="./css/app.css">
    <title>Pic.AI - Generate Images</title>
    <script defer src="./js/i18n.js"></script>
    <script defer src="./js/script.js"></script>
</head>
<body>
    <header class="header">
        <div class="header__container">
            <div class="back-logo">
                <img src="./sources/img/svg/lens_purple.svg" alt="logo">
```

```
            <h2>Pic.AI</h2>
        </div>
        <nav class="nav">
            <ul class="nav__menu">
                <li class="lang-selector">
                    <select id="language-selector" class="lang-select">
                        <option value="en">English</option>
                        <option value="uk">Українська</option>
                    </select>
                </li>
            </ul>
        </nav>
        <div class="user">
            <div class="user__mail"><p></p></div>
            <div class="user__logo"><img src="./sources/img/svg/avatar.svg" alt="Avatar"></div>
            <div class="user__dropdown">
                <a href="/logout" class="user__dropdown-item" data-i18n="nav.logout">Logout</a>
            </div>
        </div>
    </div>

</header>

<main class="main">
    <div class="main__container">
        <form id="section__form">
            <textarea type="text" class="input" id="prompt" data-i18n-placeholder="app.prompt" placeholder="Enter some prompt here"></textarea>
            <button class="button" type="submit" data-i18n="app.start">Start!</button>
        </form>
        <div class="content__wrapper">
            <div class="image__container">
                <div class="image" id="image"></div>
                <div class="loader">
                    <span></span>
                    <span></span>
                    <span></span>
                </div>
            </div>
            <div class="history" id="history">
                <h3 class="history__title" data-i18n="app.history">Generation History</h3>
                <div class="history__list" id="historyList">
                </div>
            </div>
        </div>


    </div>
</main>
<footer class="footer">
    <div class="footer__container">
        <p data-i18n="app.footer">© Pic.AI, 2025. All rights reserved.</p>
```

```
        </div>
    </footer>

    <script>
        function initLanguageSelector() {
            const languageSelector = document.getElementById('language-selector');
            if (languageSelector && typeof I18n !== 'undefined') {
                languageSelector.addEventListener('change', (e) => {
                    I18n.setLanguage(e.target.value);
                });

                languageSelector.value = I18n.getCurrentLanguage();
            }
        }

        if (document.readyState === 'loading') {
            document.addEventListener('DOMContentLoaded', initLanguageSelector);
        } else {
            initLanguageSelector();
        }

        window.addEventListener('languageChanged', (e) => {
            const languageSelector = document.getElementById('language-selector');
            if (languageSelector) {
                languageSelector.value = e.detail.language;
            }
        });
    </script>

</body>
</html>
```

Лістинг коду en.json та uk.json

```
{
  "nav": {
    "home": "Home",
    "about": "About Us",
    "howItWorks": "How it works",
    "examples": "Examples",
    "contacts": "Get In Touch",
    "logout": "Logout"
  },
  "header": {
    "signIn": "Sign in",
    "signUp": "Sign up",
    "logo": "Pic.AI"
  },
  "home": {
    "heading": "Craft the Unseen:",
    "headingHighlight": "Where AI Meets Artistry",
    "subtitle": "Transform your imagination into stunning visual masterpieces using cutting-edge AI technology",
```

```
    "cta": "Start Creating"
  },
  "aboutUs": {
    "comment1": "Welcome to Pic.AI, where your imagination is transformed into visual masterpieces using cutting-edge artificial intelligence technology.",
    "comment2": "Our aim is to make the creative process more accessible and engaging for everyone eager to bring their creative ideas to life.",
    "comment3": "We leverage the power of AI to help you create images that would be difficult or impossible to craft by hand."
  },
  "howItWorks": {
    "title": "How it works?",
    "step1": "Sign up",
    "step2": "Write a prompt",
    "step3": "Generate image",
    "step4": "Download"
  },
  "examples": {
    "title": "Examples"
  },
  "contacts": {
    "title": "Get In Touch",
    "email": "Email",
    "phone": "Phone",
    "address": "Address",
    "addressCity": "61002, Kharkiv, Ukraine",
    "addressStreet": "Yaroslava Mudroho Str. 25"
  },
  "footer": {
    "copyright": "© Pic.AI, 2025. All rights reserved."
  },
  "auth": {
    "login": "Sign in",
    "register": "Sign up",
    "email": "Email",
    "password": "Password",
    "confirmPassword": "Confirm Password",
    "name": "Full Name",
    "submit": "Submit",
    "noAccount": "Don't have an account?",
    "haveAccount": "Already have an account?",
    "signInLink": "Sign in",
    "signUpLink": "Sign up"
  },
  "app": {
    "prompt": "Enter some prompt here",
    "start": "Start!",
    "history": "Generation History",
    "logout": "Logout",
    "delete": "Delete",
    "download": "Download",
    "confirmDelete": "Are you sure you want to delete this image?",
    "footer": "© Pic.AI, 2025. All rights reserved."
```

```
  }
}

////////////////////////////////////////////////////////////////////
{
  "nav": {
    "home": "Головна",
    "about": "Про нас",
    "howItWorks": "Як це працює",
    "examples": "Приклади",
    "contacts": "Зв'язатися з нами",
    "logout": "Вихід"
  },
  "header": {
    "signIn": "Вхід",
    "signUp": "Реєстрація",
    "logo": "Pic.AI"
  },
  "home": {
    "heading": "Створіть неможливе:",
    "headingHighlight": "Де ШІ зустрічається з мистецтвом",
    "subtitle": "Трансформуйте вашу уяву в чудові візуальні шедеври, використовуючи передові технології ШІ",
    "cta": "Почати творити"
  },
  "aboutUs": {
    "comment1": "Ласкаво просимо до Pic.AI, де ваша уява перетворюється на візуальні шедеври за допомогою передових технологій штучного інтелекту.",
    "comment2": "Наша мета - зробити творчий процес більш доступним та привабливим для всіх, хто прагне втілити свої творчі ідеї.",
    "comment3": "Ми використовуємо потужність ШІ, щоб допомогти вам створювати зображення, які важко або неможливо створити вручну."
  },
  "howItWorks": {
    "title": "Як це працює?",
    "step1": "Зареєструйтесь",
    "step2": "Напишіть опис",
    "step3": "Зачекайте",
    "step4": "Завантажуйте"
  },
  "examples": {
    "title": "Приклади"
  },
  "contacts": {
    "title": "Зв'язатися з нами",
    "email": "Email",
    "phone": "Телефон",
    "address": "Адреса",
    "addressCity": "61002, Харків, Україна",
    "addressStreet": "вул. Ярослава Мудрого, 25"
  },
  "footer": {
    "copyright": "© Pic.AI, 2025. Всі права захищені."
```

```
  },
  "auth": {
    "login": "Вхід",
    "register": "Реєстрація",
    "email": "Email",
    "password": "Пароль",
    "confirmPassword": "Підтвердіть пароль",
    "name": "Повне ім'я",
    "submit": "Відправити",
    "noAccount": "Немає акаунту?",
    "haveAccount": "Вже є акаунт?",
    "signInLink": "Увійти",
    "signUpLink": "Зареєструватися"
  },
  "app": {
    "prompt": "Введіть опис тут",
    "start": "Почати!",
    "history": "Історія генерацій",
    "logout": "Вихід",
    "delete": "Видалити",
    "download": "Завантажити",
    "confirmDelete": "Ви впевнені, що хочете видалити це зображення?",
    "footer": "© Pic.AI, 2025. Всі права захищені."
  }
}
```

ДОДАТОК Б

ТЕКСТ СТАТТІ: ЗБЕРЕЖЕННЯ ТА ОРГАНІЗАЦІЯ ЗГЕНЕРОВАНИХ AI-ЗОБРАЖЕНЬ: ПРОСТІ ПІДХОДИ

УДК 004.91

ЗБЕРЕЖЕННЯ ТА ОРГАНІЗАЦІЯ ЗГЕНЕРОВАНИХ AI-ЗОБРАЖЕНЬ: ПРОСТІ ПІДХОДИ

Кирилов Д.І., Лебединський А.В.

Харківський національний автомобільно-дорожній університет, Харків

Розробка та використання AI-генерованих зображень стає все поширенішим явищем у навчальному, творчому та професійному середовищі. Завдяки доступності інструментів на кшталт DALL·E, MidJourney, Stable Diffusion та інших, навіть користувачі без художніх навичок можуть швидко створювати візуальний контент для презентацій, навчальних матеріалів, дизайну інтерфейсів чи маркетингових кампаній [1]. Однак разом із зростанням обсягів створеного контенту виникає нагальна потреба в систематизації та ефективному зберіганні цих даних. Відсутність чіткої структури призводить до втрати файлів, ускладнення пошуку, дублювання роботи та порушення авторських прав, особливо коли важливо відстежити походження зображення або умови його використання. Тому важливо впроваджувати прості, але дієві підходи до організації цифрових активів, особливо в умовах інтенсивного використання генеративних моделей.

Першим кроком до ефективної організації є створення чіткої системи іменування файлів, яка слугує основою для подальшої структуризації цифрових активів [2]. Рекомендується використовувати описові, але стандартизовані назви, що дотримуються єдиного шаблону. Ідеальна назва файлу має включати такі компоненти: дату створення у форматі *РРРР-ММ-ДД* (що забезпечує хронологічне сортування), скорочену назву моделі, яка була використана для генерації (наприклад, dalle3, sd3, midjourney), основну тему або сюжет, ключові

об'єкти чи стилістику, а також версію файлу. Наприклад: *2025-10-15_dalle3_portrait-ukrainian-woman-nature_v1.png*. Така структура дозволяє одразу зрозуміти, коли було створено зображення, якою моделлю, і за яким «промптом» (непрямо), що особливо корисно при роботі з великою кількістю варіантів одного задуму.

Другим важливим елементом є використання чіткої папкової структури, побудованої за логічними принципами – тематичними, часовими або проектними. Така ієрархія дозволяє швидко орієнтуватися в обсязі згенерованих зображень і уникнути хаосу, який часто виникає при масовій генерації контенту. Наприклад, коренева тека *AI_Images* може бути розбита на підтеки за роками: *AI_Images/2025/, AI_Images/2024/*, що спрощує архівування та пошук за періодом. Усередині кожної річної теки доцільно створити підпапки за напрямами – такими як *Education_Materials, Research_Visuals, Creative_Projects*, *Presentations* чи *Student_Assignments*, залежно від призначення зображень. Усередині кожної проектної папки можна додатково виділити структуру за етапами роботи: наприклад, підкаталоги *01_Prompts, 02_Generated_Variants, 03_Selected, 04_Edited, 05_Final_Export* [3].

```
AI_Images/
└── 2025/
    └── Education_Materials/
        ├── 01_Prompts/
        ├── 02_Generated_Variants/
        ├── 03_Selected/
        ├── 04_Edited/
        └── 05_Final_Export/
```

Рисунок 1 – Діаграма структури каталогів

Це дозволяє не тільки зберігати фінальний результат, а й відстежувати еволюцію ідеї, що особливо корисно в навчальному процесі або при командній роботі. Наприклад, при підготовці навчальних слайдів можна зберігати кілька варіантів візуалізації одного й того ж поняття, порівнюючи, як різні «промпти»

впливають на результат. Така структура також полегшує автоматизацію: скрипти чи інструменти типу PowerShell, Python або Bulk Rename Utility можуть обробляти файли масово, сортувати їх або додавати теги на основі шляху до файлу [4]. Крім того, чітка ієрархія спрощує резервне копіювання та синхронізацію з хмарними сховищами, забезпечуючи безпеку даних. У разі передачі матеріалів колегам чи студентам така організація робить спільну роботу прозорою та інтуїтивно зрозумілою.

Третім підходом є збереження метаданих разом із зображеннями, що забезпечує прозорість, відтворюваність і етичну відповідальність у роботі з AI-генерованим контентом. Ключовим є фіксація не лише вихідного «промпту», а й усіх параметрів, які вплинули на результат: seed (початкове число для генератора випадковості), кількість ітерацій, модель та її версія, розмір зображення, швидкість генерації, негативні «промпти» та інші налаштування [5]. Ці дані дозволяють не лише відтворити ідентичне зображення при повторній генерації, а й зрозуміти, як саме певні зміни в «промпті» чи параметрах впливають на вихідний результат, що особливо важливо в навчальних та дослідницьких умовах.

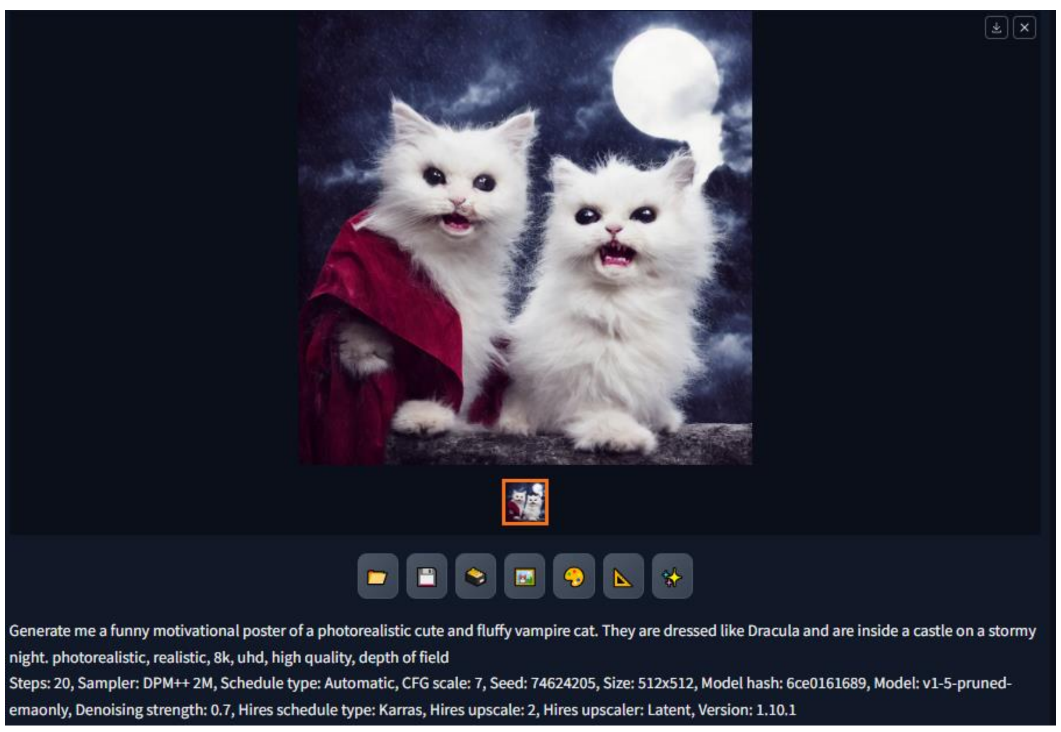


Рисунок 2 – Метадані згенерованого зображення в інтерфейсі AUTOMATIC1111

Для зберігання цієї інформації можна використовувати кілька стратегій. Найпростіший спосіб – створення супровідного текстового файлу з розширенням .txt, названим аналогічно до зображення (наприклад, image_v3_prompt.txt для файлу image_v3.png). Такий файл може містити структуровану інформацію у форматі, зручному для читання та аналізу. Альтернативно, слід використовувати формати файлів, що підтримують вбудовані метадані, зокрема PNG, який дозволяє зберігати коментарі без втрати якості. Деякі генератори типу Stable Diffusion через вебінтерфейси (наприклад, AUTOMATIC1111), автоматично зберігають «промпти» та параметри в EXIF-подібних тегах PNG-файлів, що робить їх самодостатніми.

У поєднанні ці підходи формують надійну основу для довгострокового зберігання, швидкого доступу та етичного використання AI-зображень. Вони особливо корисні в освітніх та дослідницьких середовищах, де важливі відтворюваність, прозорість і авторське право. Систематичне іменування файлів, ієрархічна папкова структура, збереження метаданих та використання інструментів управління цифровими активами разом створюють інтегровану екосистему, яка забезпечує не лише організацію, а й інтелектуальну цінність кожного зображення. Це дозволяє викладачам та студентам ефективно використовувати AI-арт як навчальний матеріал, аналізувати еволюцію ідей, порівнювати вплив різних «промптів» і параметрів, а також демонструвати прогрес у творчих чи технічних проектах [6].

Крім того, така організація сприяє дотриманню академічної доброчесності: коли «промпти», моделі та умови генерації задокументовані, стає можливим чітко визначити межу між власною творчістю та внеском штучного інтелекту. Це важливо для захисту авторських прав, уникнення плагіату та формування критичного ставлення до генерованого контенту. У дослідницькій діяльності такі стандарти дозволяють іншим науковцям перевіряти результати, відтворювати експерименти та розвивати ідеї, що сприяє прогресу в галузі штучного інтелекту та цифрової культури. Впровадження цих

практик на рівні навчальних закладів закладає основу для відповідальної та професійної роботи з AI у майбутньому.

Список використаних джерел

[1] Hrynkevych O., Soroka Y. Comparative analysis of neural networks Midjourney, Stable Diffusion, and DALL-E and methods of their implementation in the educational process of students of design specialities. Scientific Bulletin of Mukachevo State University. Series "Pedagogy and Psychology". 2023. Vol. 9, № 3. P. 36-44. https://doi.org/10.52534/msu-pp3.2023.36

[2] Regli W. What Are the Core Elements Necessary for Effective Digital Asset Management? Modern Economy. 2021. Vol. 12, № 10. P. 1485-1510. https://doi.org/10.4236/me.2021.1210077

[3] Holmes N. File Naming Best Practices for Digital Asset Management. Acquia. 2022. URL: https://www.acquia.com/blog/file-naming-conventions (дата звернення: 15.10.2025).

[4] The Guide to Folder Structures: Best Practices for Professional Service Firms. SuiteFiles. 2025. URL: https://www.suitefiles.com/guide/the-guide-to-folder-structures-best-practices-for-professional-service-firms-and-more/ (дата звернення: 15.10.2025).

[5] Essential guide to image metadata. FotoWare. 2025. URL: https://www.fotoware.com/blog/essential-guide-to-image-metadata (дата звернення: 15.10.2025).

[6] Guidance for generative AI in education and research. UNESCO. 2023. URL: https://unesdoc.unesco.org/ark:/48223/pf0000386693 (дата звернення: 15.10.2025).

ДОДАТОК В
ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ДИПЛОМНОЇ РОБОТИ

Міністерство освіти і науки України
Харківський національний автомобільно-дорожній університет

Механічний факультет

Кафедра комп’ютерних наук і інформаційних систем

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ДИПЛОМНОЇ РОБОТИ

магістра

РОЗРОБКА ВДОСКОНАЛЕНОЇ ПЛАТФОРМИ ДЛЯ ГЕНЕРАЦІЇ ТА УПРАВЛІННЯ ЗОБРАЖЕННЯМИ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Завідувачка кафедри канд. техн. наук, доц. Ганна ПЛЄХОВА

Нормоконтролер, к.т.н. доц. Сергій НЕРОНОВ

Керівник, док. філ. доц. Андрій ЛЕБЕДИНСЬКИЙ

Студент гр. МК-61-24 Дмитро КИРИЛОВ

Харків – 2025

МЕТА РОБОТИ, ОБ'ЄКТ ДОСЛІДЖЕННЯ ТА ЗАДАЧІ

Мета роботи – розробка вдосконаленої вебплатформи Pic.AI для автоматизованої генерації зображень на основі текстових описів з використанням технологій штучного інтелекту OpenAI DALL-E 3, що забезпечує високу якість візуалізації, зручність користувацького інтерфейсу та ефективне управління історією генерацій.

Об'єкт дослідження - процес генерації та управління цифровими зображеннями з використанням технологій штучного інтелекту в вебсередовищі.

Задачі:

- проаналізувати сучасні методи генерації зображень та існуючі програмні рішення, виявити їхні переваги та недоліки;
- обґрунтувати вибір архітектурного підходу та технологічного стеку для реалізації платформи;
- розробити архітектуру системи та структуру бази даних для зберігання профілів користувачів та історії генерацій;
- реалізувати програмні модулі автентифікації, взаємодії з API DALL-E 3 та обробки зображень;
- створити адаптивний клієнтський інтерфейс із підтримкою багатомовності.
- провести тестування функціональності, безпеки та продуктивності розробленої вебплатформи.

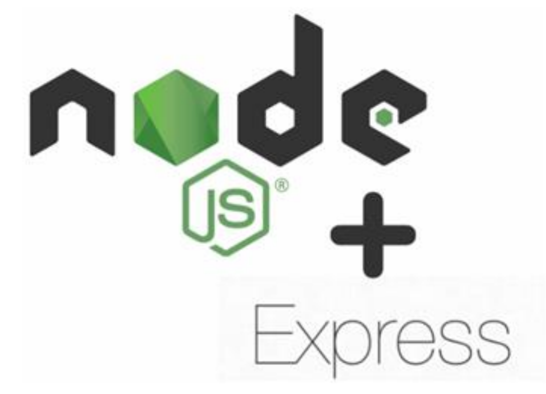
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

У роботі було проаналізовано наступні технології генерації зображень: Stable Diffusion, DALL·E 3, MidJourney.

Для реалізації функціональності генерації зображень у межах розроблюваного додатку було обрано технологію DALL·E 3. Рішення ґрунтується на поєднанні двох ключових критеріїв відбору: якості візуального результату та практичної придатності інструмента до програмної інтеграції в серверну частину системи.

Рисунок В.1 – Існуючі технології генерації зображень

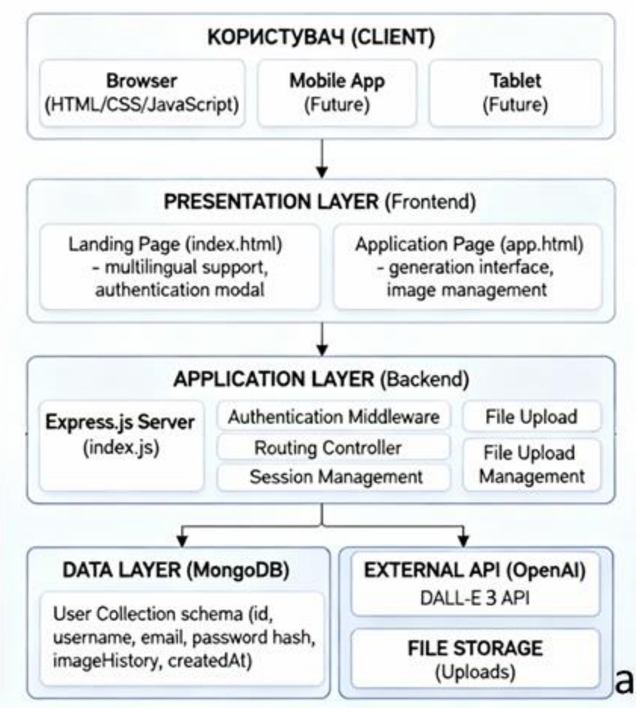
ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЧНОГО СТЕКУ

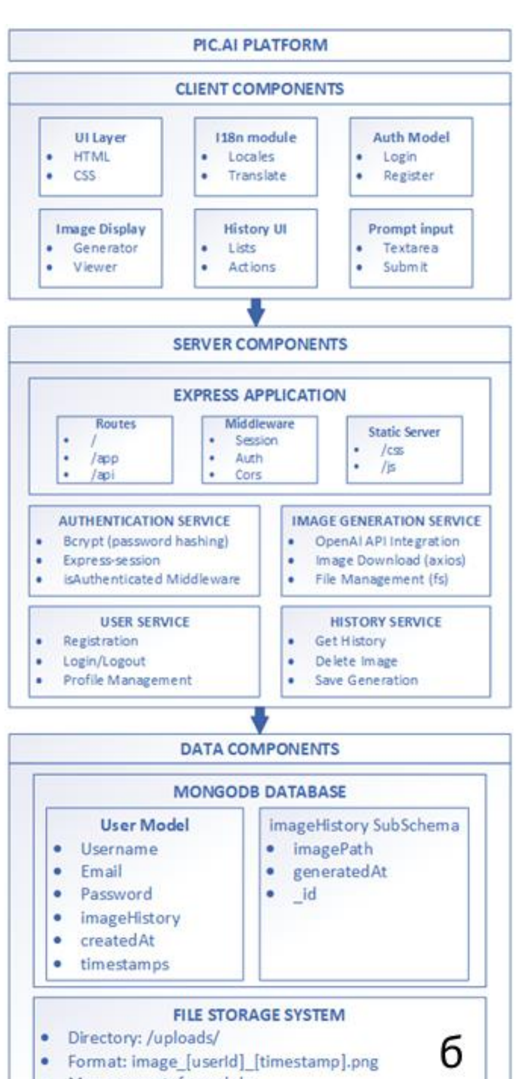


4

Рисунок В.2 – Технологічний стек вебплатформи

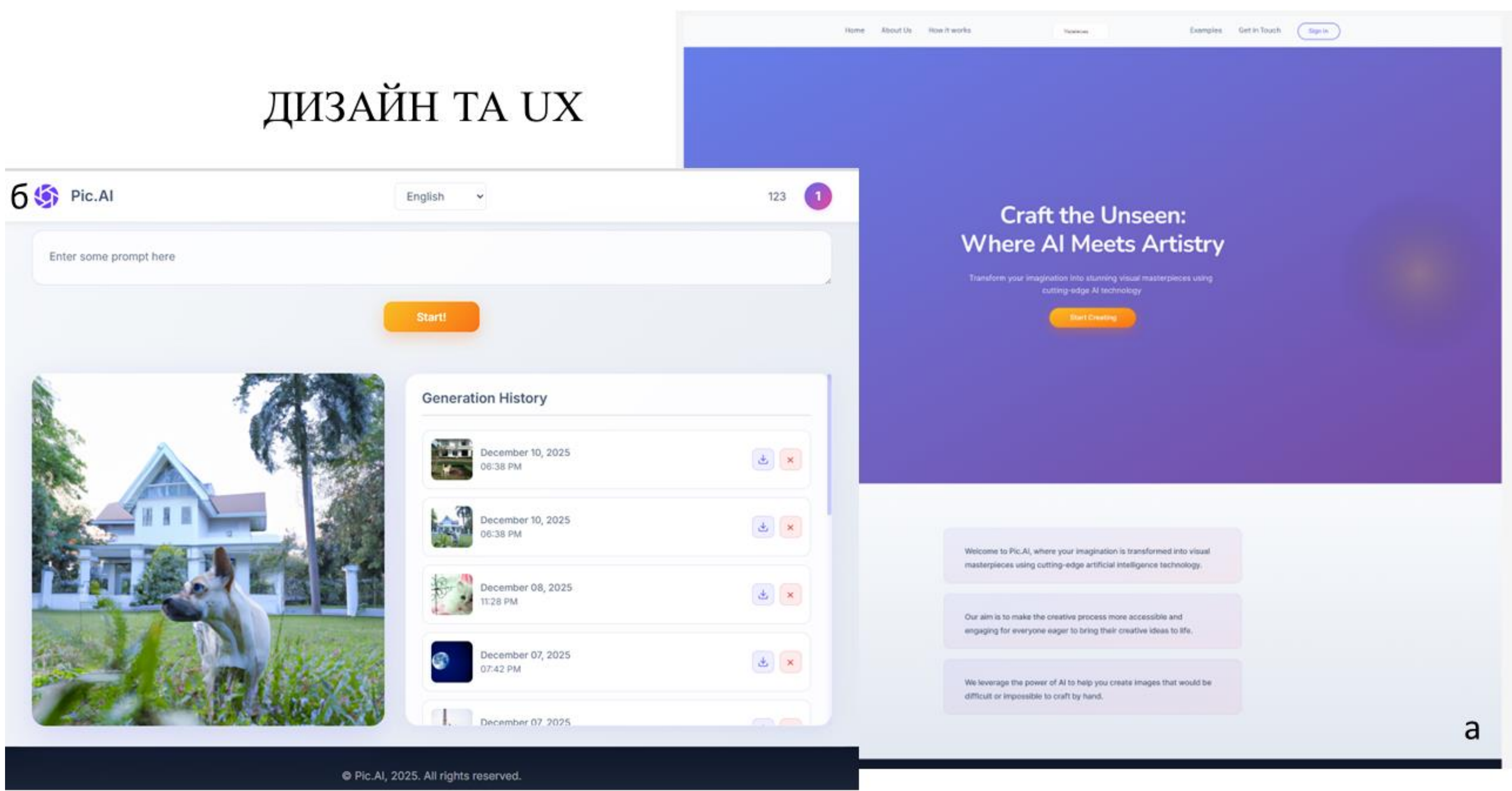
АРХІТЕКТУРА ПЛАТФОРМИ





5

Рисунок В.3 – Архітектурна (а) та компонентна діаграми (в)



б

Рисунок В.4 – Дизайн привітальної сторінки (а) та головної сторінки вебплатформи (б)

РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

Метод	Маршрут	Опис
POST	/register	Реєстрація користувача
POST	/login	Вхід у систему
GET	/api/user	Отримати дані користувача
GET	/logout	Вихід з облікового запису
GET	/api/history/	Отримати історію
DELETE	/api/history/:imageId	Видалити зображення
POST	/generateImage	Генерація зображення

Таблиця В.1 – Опис API-маршрутів

- **Система автентифікації**: bcrypt хешування, express-session для управління сесіями;
- **Безпека**: валідація входів, захист паролів, логування помилок без розкриття технічних деталей;
- **Інтеграція з OpenAI**: SDK для Node.js, обробка асинхронних запитів з Fetch API, завантаження та збереження зображень локально на сервері;
- **Обробка помилок**: Try/catch блоки на всіх критичних операціях, коректні HTTP статус-коди (200, 201, 400, 401, 404, 500);
- **Управління файлами**: Унікальні імена файлів (image_{userId}_{timestamp}.png), ізоляція за користувачами, видалення при вилученні запису.

РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

Параметри генерації зображення

- Модель: dall-e-3 (найновіша, найкраща якість)
- Розмір: 1024×1024 px (оптимальна якість/швидкість)
- Кількість: 1 зображення за раз (обмеження API)
- Формат: PNG

Обробка помилок

- Невірний API ключ → повідомлення користувачу
- Перевищена квота → інформація про ліміти
- Тимчасова недоступність → пропозиція повтору
- Некоректний запит → деталізована помилка від OpenAI

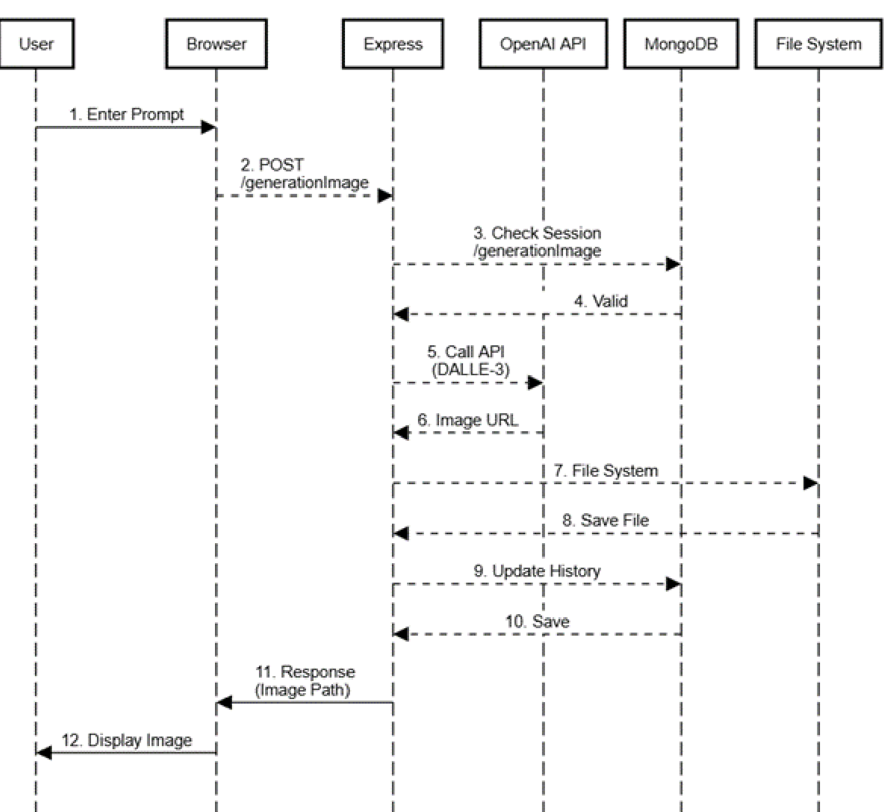


Рисунок В.5 – Діаграма послідовності

ТЕСТУВАННЯ ТА ЯКІСТЬ

Модульне тестування

- Тести для моделі User (валідація, хешування паролів)
- Тести для проміжного ПЗ isAuthenticated
- Тести для обробки помилок API

API тестування

- POST /register – успішна реєстрація, дублікат email, валідація
- POST /login – успішний вхід, невірні реквізити
- POST /generateImage – лише для авторизованих, обробка помилок
- DELETE /api/history/:imageId – перевірка приналежності файлу користувачу

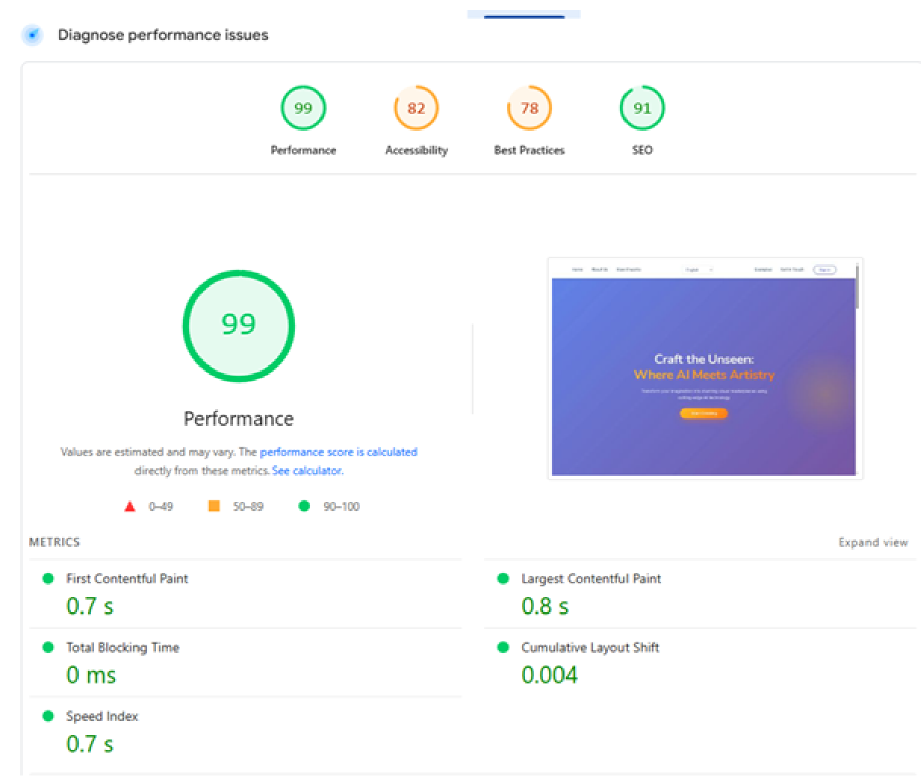


Рисунок В.6 – Метрика Google PageSpeed

РОЗГОРТАННЯ ТА ВИКОРИСТАННЯ

Архітектура розгортання на VPS

- VPS сервер з Linux (Ubuntu 22.04 LTS).
- Docker Engine встановлений на сервері.
- Pic.AI контейнер запущений в контейнері.
- Порт 5000 відкритий для доступу з інтернету.
- MongoDB Atlas за межами контейнера (хмарна БД).

Конфігурація середовища:

- PORT=5000
- MONGODB_URI=<atlas-connection-string>
- OPENAI_API_KEY=<secret-key>
- SESSION_SECRET=<random-string>

```
FROM node:20.15.0                  #Базовий образ Node.js
WORKDIR /pic_ai                    #Встановлення робочої директорії
COPY package*.json ./              #Копіювання файлів package.json та package-lock.json
RUN npm install                    #Встановлення залежностей
COPY . .                           #Копіювання всіх файлів проекту
EXPOSE 5000                        #Відкриття порту 5000
CMD ["npm", "run", "start"]        #Команда для запуску додатку
```


Рисунок В.7 – Вміст Dockerfile

ВИСНОВКИ

- В даній дипломній роботі магістра було досліджено та реалізовано повнофункціональну вебплатформу Pic.AI для генерації зображень на основі текстових описів з використанням технологій штучного інтелекту. У процесі виконання роботи було проведено комплексний аналіз існуючих рішень в галузі AI-генерації зображень, виявлено їх переваги та недоліки, що дозволило сформулювати чіткі вимоги до функціональності та архітектури власної платформи.

- Серверна частина платформи реалізована на Node.js із Express.js, MongoDB та Mongoose для гнучкого зберігання даних, система автентифікації базується на express-session та bcrypt, а ключовою компонентою є інтеграція з OpenAI API та DALL-E 3 для генерації зображень з локальним збереженням результатів.

- Клієнтська частина розроблена на нативних HTML5, CSS3 та JavaScript ES6+ з адаптивним дизайном та системою інтернаціоналізації для англійської та української мов через кастомний i18n.js модуль. Архітектура базується на модульності з RESTful API-маршрутами, а модульне тестування з Jest підтвердило коректність бізнес-логіки та готовність платформи до використання в реальних умовах.