Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА DISCORD-БОТА ДЛЯ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ І МОЖЛИВІСТЮ ПОСТОБРОБКИ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ МОВИ C#

DEVELOPMENT OF A DISCORD BOT FOR IMAGE GENERATION USING ARTIFICIAL INTELLIGENCE WITH IMAGE POST-PROCESSING USING C#

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗс-21
Приходько Владислав Іванович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«10» 12 2024р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Приходьку Владиславу Івановичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка Discord-бота для генерації зображень з використанням штучного інтелекту і можливістю постобробки зображень з використанням мови C#»*
Керівник роботи: *к.т.н., доцент Ящук А. А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*

3. Вихідні дані до роботи: *засоби генерації зображень, середовища розробки, системи контролю версій, методичні вказівки до виконання кваліфікаційної роботи бакалавра*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз проблеми й аналогів, визначення вимог, UML-діаграма прецедентів, вибір засобів і методів, реалізація бота, UML-діаграма компонентів додатку, тестування й налагодження, захист ІКС, документація, інсталяція для Windows і Linux.*

5. Перелік графічного матеріалу: *8 рисунків*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ящук А. А.*		
Специфікація вимог до розробленої системи	*Ящук А. А.*		
Розробка об'єкта проектування	*Ящук А. А.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,67 %		
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Владислав ПРИХОДЬКО

Керівник кваліфікаційної роботи _____ Андрій ЯЩУК

АНОТАЦІЯ

Приходько В. І. Розробка Discord-бота для генерації зображень з використанням штучного інтелекту і можливістю постобробки зображень з використанням мови C#. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі здійснено аналіз проблемної області, розглянуто аналогічні розробки та сформульовано завдання. У другому розділі визначено вимоги до програмного забезпечення, обґрунтовано вибір інструментів і технологій, а також описано архітектуру системи. У третьому розділі реалізовано Discord-бота, наведено фрагменти коду ключових компонентів, створено UML-діаграму обробки команд і розглянуто механізм постобробки зображень. У висновках підбито підсумки виконаної роботи та окреслено перспективи подальшого розвитку системи.

Ключові слова: Discord-бот, генерація зображень, штучний інтелект, Stable Diffusion, C#, .NET, Discord API, Windows, Linux, ін’єкція залежностей (DI), IDE.

ABSTRACT

Prykhodko V. Development of a Discord Bot for Image Generation Using Artificial Intelligence with Image Post-processing Using C#. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references.

The first chapter presents an analysis of the problem domain, reviews similar solutions, and formulates the objectives. The second chapter defines the software requirements, justifies the choice of tools and technologies, and describes the system architecture. The third chapter covers the implementation of the Discord bot, provides code fragments of key components, presents the UML diagram of command processing, and discusses the post-processing mechanism for generated images. The conclusions summarize the results of the work and outline prospects for further development of the system.

Keywords: Discord bot, image generation, artificial intelligence, Stable Diffusion, C#, .NET, Discord API, Windows, Linux, dependency injection (DI), IDE.

ЗМІСТ

ВСТУП

Актуальність теми. З огляду на стрімкий розвиток технологій штучного інтелекту та зростання популярності платформ для спілкування, таких як Discord, інтеграція інструментів генерації зображень у середовища онлайн-комунікації є надзвичайно затребуваною. Discord активно використовується не лише для розваг, а й у професійних спільнотах, де автоматизація рутинних завдань і розширення можливостей користувачів відіграє важливу роль. Створення Discord-бота, здатного генерувати зображення за текстовим описом із подальшою постобробкою, дозволяє розробити сучасний, інтерактивний та функціональний інструмент на перетині штучного інтелекту, хмарних сервісів і десктопних технологій.

Метою кваліфікаційної роботи є створення програмного засобу – Discord-бота для генерації та редагування зображень із використанням штучного інтелекту, реалізованого засобами мови програмування C# та сучасного технологічного стеку.

Об’єктом кваліфікаційної роботи є процес генерації та обробки зображень із використанням засобів штучного інтелекту в рамках клієнт-серверної архітектури.

Предметом кваліфікаційної роботи є програмні засоби розробки Discord-бота, методи обробки текстових запитів і генерації зображень, а також алгоритми забезпечення інтеграції та захисту інформації.

У межах цієї кваліфікаційної роботи буде реалізовано повний цикл створення програмного забезпечення – від аналізу предметної області до розгортання готового продукту з підтримкою інсталяції на різних операційних системах.

У процесі виконання роботи буде здійснено такі дії:

– проведено аналіз сучасного стану проблеми та огляд аналогічних розробок;

– визначено функціональні та нефункціональні вимоги до програмного забезпечення та побудовано UML-діаграму прецедентів взаємодії з Discord-ботом;

– обґрунтовано вибір засобів, методів і алгоритмів для реалізації системи, а також підібрано оптимальні інструменти розробки;

– реалізовано функціональність Discord-бота з використанням обраних технологій, наведено фрагменти коду ключових компонентів і побудовано UML-діаграму компонентів обробки команд;

– проведено тестування та налагодження розробленої системи;

– забезпечено заходи з інформаційної безпеки та захисту даних;

– підготовлено супровідну документацію до програмного продукту;

– створено інсталяційні пакети для платформ Windows та Linux.

РОЗДІЛ 1
АНАЛІЗ ЗАСОБІВ ДЛЯ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Штучний інтелект (ШІ) у сфері генерації зображень є однією з найбільш динамічно розвинених галузей сучасних технологій. Серед ключових рішень, які отримали широке визнання, можна виділити такі проєкти, як DALL-E (розроблений OpenAI), MidJourney, Stable Diffusion (розроблений Stability AI), а також Dream by Wombo. Ці системи дозволяють створювати зображення на основі текстових запитів користувача, використовуючи складні нейронні мережі та diffusion-моделі.

Огляд аналогів:

– DALL-E 2 демонструє високу якість зображень, гнучкість у відображенні складних сцен і можливість варіювати стиль [1]. Проте використання обмежене через політику доступу та обмежену можливість тонкої постобробки прямо у сервісі;

– MidJourney орієнтований на створення художніх і креативних зображень з вираженим стилем [2]. Система інтегрована через Discord-бота, однак можливості редагування створеного зображення обмежені попередньо запрограмованими командами;

– Stable Diffusion надає більшу свободу, оскільки модель є відкритою, що дозволяє розгортати її на локальних серверах і налаштовувати відповідно до потреб [3]. Однак інтеграція її в прості інтерфейси (наприклад, чат-боти) вимагає додаткової розробки і технічної експертизи;

– Dream by Wombo робить акцент на простоті для кінцевого користувача, але має обмежену кастомізацію результатів і в основному спрямований на розважальний сегмент [4].

Переваги існуючих рішень:

– висока якість генерованих зображень;

– можливість створення оригінальних візуальних матеріалів без потреби у професійних художніх навичках;

– поява інтерфейсів для масового використання.

Недоліки існуючих рішень:

– обмежені можливості постобробки зображень безпосередньо після генерації;

– не завжди інтуїтивно зрозумілий інтерфейс взаємодії для недосвідчених користувачів;

– відсутність повної гнучкості у налаштуванні моделей або результатів без глибоких технічних знань;

– обмеження у використанні на власних платформах або спільнотах (платність, залежність від сервісів).

Патентний пошук. Огляд патентних баз (зокрема, Google Patents) свідчить про велику кількість патентів, пов’язаних із генерацією зображень за допомогою штучного інтелекту та їх обробкою. Однак конкретних запатентованих рішень для інтеграції функціоналу генерації та постобробки в екосистему Discord виявлено не було, що залишає простір для інноваційної реалізації такого проєкту.

Обґрунтування доцільності створення нової системи. З огляду на вказані обмеження існуючих рішень, виникає потреба у створенні Discord-бота, який би:

– надавав користувачам можливість не лише генерувати зображення, а й здійснювати базову постобробку прямо в чаті;

– забезпечував простий та зрозумілий інтерфейс взаємодії;

– використовував сучасні AI-моделі генерації зображень;

– був придатним для гнучкого налаштування під конкретні потреби спільнот.

Запропонований підхід дозволить підвищити доступність технологій штучного інтелекту для масового користувача та розширити можливості творчої роботи в онлайн-спільнотах.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Були поставлені наступні завдання на кваліфікаційну роботу бакалавра:

– здійснити аналіз сучасного стану проблеми та проаналізувати аналогічні розробки;

– визначити функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення та побудувати UML-діаграму прецедентів взаємодії з Discord-ботом;

– обґрунтувати вибір засобів, методів і алгоритмів розв’язання поставленого завдання, а також підібрати оптимальні інструменти розробки;

– реалізувати Discord-бота з використанням обраних засобів, навести фрагменти коду ключових компонентів системи, побудувати UML-діаграму компонентів обробки команд Discord-бота;

– провести тестування та налагодження Discord-бота;

– забезпечити захист інформаційно-комп’ютерної системи;

– підготувати документацію до програмного продукту;

– створити інсталяційні пакети для операційних систем Windows та Linux.

Висновки до розділу 1

У першому розділі було проведено аналіз сучасного стану проблеми генерації зображень за допомогою штучного інтелекту, а також досліджено існуючі рішення, що дозволяють реалізовувати подібні функції у вигляді чат-ботів, зокрема для платформи Discord. Зокрема, було проаналізовано найпопулярніші сервіси генерації зображень – DALL-E 2, MidJourney, Stable Diffusion та Dream by Wombo. В результаті аналізу з’ясовано, що, попри значні

досягнення у сфері генеративного ШІ, більшість існуючих рішень мають низку обмежень – як у плані доступності інструментів постобробки, так і зручності користувацької взаємодії.

Проведене порівняння аналогів і виявлені недоліки засвідчили доцільність створення нового інструменту – Discord-бота, який би об’єднував функціональність генерації зображень за текстовим запитом з можливістю виконання базової обробки отриманого зображення безпосередньо у межах платформи.

Також у цьому розділі було чітко визначено перелік завдань, що охоплюють повний цикл розробки програмного забезпечення – від аналізу проблемної області до створення інсталяційних пакетів. Отримані результати аналітичного етапу слугуватимуть основою для подальшого проектування, реалізації та тестування Discord-бота, що розглядається у наступних розділах.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

При розробці Discord-бота для генерації зображень із застосуванням штучного інтелекту та можливістю постобробки ключовим є вибір відповідної архітектури системи, яка дозволяє інтегрувати сучасні генеративні моделі (наприклад, Stable Diffusion) із зручним інтерфейсом у вигляді чату. Для моделювання програмної системи обрано клієнт-серверну архітектуру з централізованим бекендом, який відповідає за обробку запитів, комунікацію з моделлю ШІ, зберігання результатів та реалізацію інструментів постобробки.

Аналіз вимог до системи було проведено за допомогою методів опитування потенційних користувачів, вивчення функціоналу існуючих аналогів, а також сценарного аналізу використання. На основі виявлених потреб побудовано перелік функціональних вимог до програмного забезпечення.

Сформовано наступні функціональні вимоги до програмного продукту:

– генерація зображення на основі текстового запиту користувача (prompt);

– збереження створених зображень із доступом до історії;

– можливість постобробки створеного зображення, включаючи зміну розміру, застосування фільтрів (пікселізація, зображення з відтінків сірого, олійний стиль), а також регулювання контрастності та яскравості;

– завантаження зображення користувачем для подальшої обробки;

– відправка результату користувачу у вигляді вкладеного файлу в Discord;

– обмеження кількості запитів на генерацію або обробку (rate-limiting);

– логування активності користувачів та помилок для подальшої діагностики;

– можливість налаштування префіксів команд для окремих серверів Discord.

Умови використання системи передбачають інтерактивну взаємодію через чат-команди в середовищі Discord. Тому особлива увага приділена UI/UX-дизайну командної системи, оскільки команди мають бути інтуїтивно зрозумілими, короткими й такими, що швидко запам’ятовуються. Користувачеві має бути доступна контекстна допомога та зворотний зв’язок у разі помилок або некоректного введення.

Проєктування взаємодії системи з користувачем та серверною частиною здійснюється з урахуванням обробки запитів через REST API або WebSocket (для генерації зображень у реальному часі), а також через Discord API для реалізації команд і повідомлень [5, 6, 7]. Інформаційні потоки передбачають двосторонню комунікацію між Discord-ботом та сервером, з урахуванням черг запитів, механізмів обробки в паралельному режимі та відповіді користувачу. Також під час проєктування враховано вимогу до кросплатформенності. Бот має бути легко розгорнутий як на платформі Windows, так і на Linux, з мінімальними вимогами до середовища виконання.

Вимоги до безпеки включають обмеження доступу до адміністративних функцій, перевірку дозволів через Discord OAuth2, захист від спаму, обробку некоректних запитів, налаштування списку каналів, у яких дозволено використання бота, а також захист від перевантаження системи (через обмеження кількості одночасних запитів) [8].

У процесі проектування програмного забезпечення створено UML-діаграму прецедентів, яка відображає взаємодію користувачів і адміністраторів із Discord-ботом через основні сценарії використання системи. На діаграмі також представлено перелік доступних можливостей постобробки зображень, які повинен підтримувати Discord-бот, серед яких є обрізання, регулювання яскравості та контрастності, а також перетворення зображення у відтінки сірого. Діаграма зображена на рисунку 2.1, вона дозволяє формалізувати логіку функціонування системи та слугує основою для подальшої реалізації.

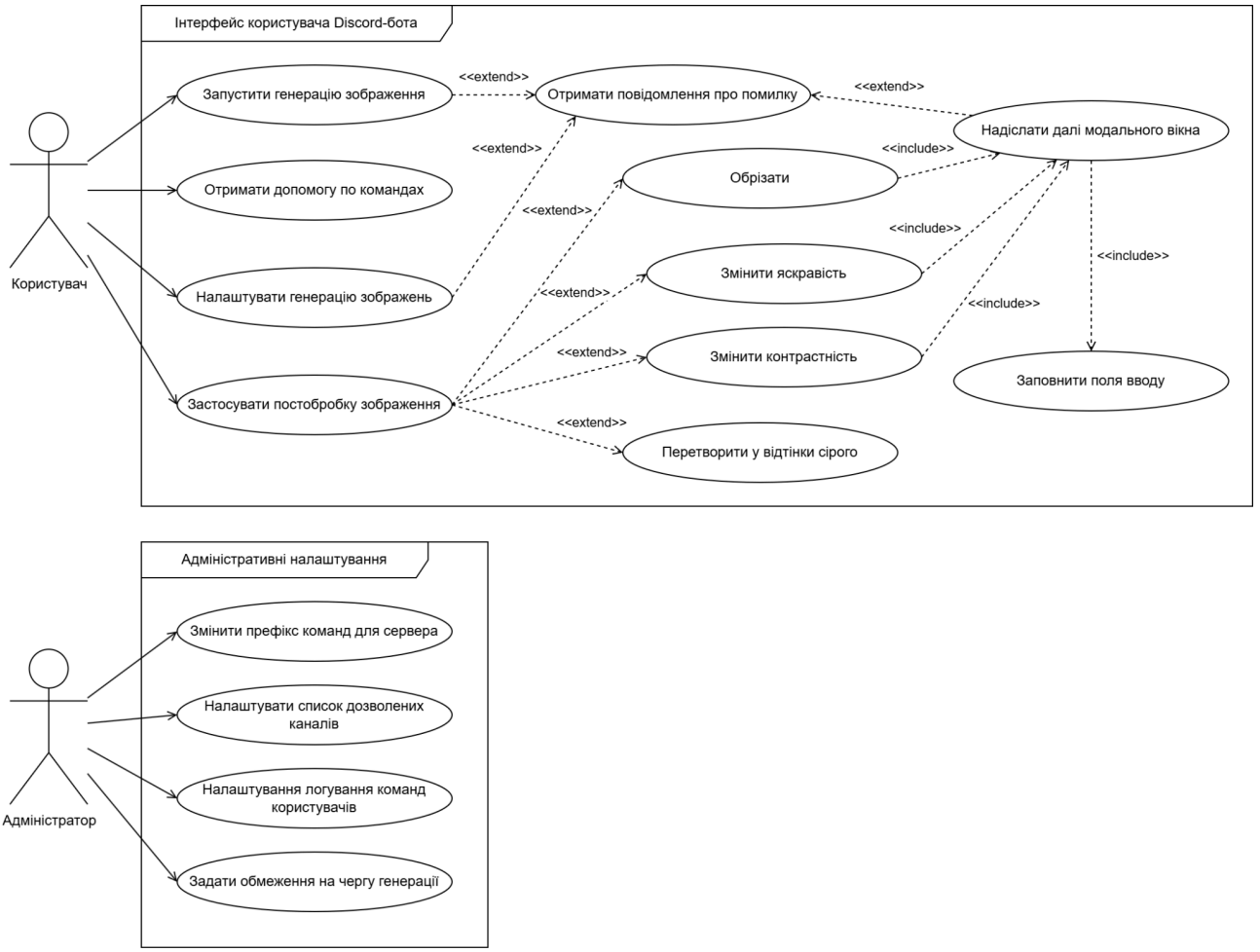

Рисунок 2.1 – UML-діаграма прецедентів взаємодії з Discord-ботом

Отримані результати моделювання свідчать про доцільність вибраної архітектури та дозволяють забезпечити гнучкість системи, її масштабованість та зручність для кінцевих користувачів. Використання сучасних технологій генерації зображень у поєднанні з інтерактивним чат-інтерфейсом відкриває нові можливості застосування ШІ у творчій діяльності та автоматизованому створенні контенту.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

На початковому етапі розробки програмного забезпечення важливим завданням стало визначення мови програмування, яка найкраще підходить для реалізації Discord-бота, здатного взаємодіяти з API генерації зображень,

обробляти запити користувачів у реальному часі та здійснювати постобробку отриманих зображень. Було розглянуто кілька найбільш популярних мов, зокрема Python, Java та C#.

Python є одним з найпопулярніших виборів для розробки ботів, зокрема завдяки простоті синтаксису, великій кількості бібліотек для роботи з Discord (наприклад, discord.py) та гнучкості в інтеграції з моделями штучного інтелекту, що часто пишуться саме цією мовою. Проте Python має деякі суттєві недоліки, серед яких – обмежена продуктивність у багатопотокових сценаріях, динамічна типізація, що може спричинити помилки на етапі виконання, а також порівняно низька швидкість обробки запитів у високонавантажених сервісах.

Java є потужною та стабільною мовою з широким набором інструментів для розробки серверних застосунків. Вона забезпечує хорошу продуктивність, підтримує багатопотоковість, має суворий контроль типів. Водночас, Java менш популярна у сфері Discord-розробки, її екосистема для створення ботів є менш гнучкою та зручною порівняно з іншими мовами. Крім того, Java-проєкти, як правило, вимагають більшої кількості конфігурацій, що може уповільнити розгортання MVP-версії продукту.

C# виявився оптимальним вибором для реалізації поставленої задачі [9, 10]. Він поєднує високу продуктивність, строгий контроль типів, сучасний синтаксис і підтримку асинхронного програмування, що є критично важливим для обробки одночасних запитів користувачів. Платформа .NET, яка використовується разом із C#, надає розробнику потужні інструменти для створення API, роботи з мережею, файловою системою, обробки графіки та взаємодії з базами даних [11].

Серед ключових переваг C# і платформи .NET:

– підтримка асинхронного програмування (async/await) забезпечує ефективну обробку паралельних запитів;

– сувора статична типізація дозволяє зменшити кількість помилок на етапі виконання;

– модульна та масштабована архітектура забезпечує зручність для подальшого розширення системи;

– кросплатформні можливості забезпечують розгортання на Windows, Linux або в хмарних середовищах, зокрема на Azure, AWS та Docker;

– сучасна екосистема забезпечує доступ до стабільних бібліотек, таких як ImageSharp для обробки зображень чи HttpClient для інтеграцій;

– висока продуктивність робить платформу придатною для систем з великою кількістю одночасних користувачів.

Таким чином, серед усіх проаналізованих варіантів C# було обрано як основну мову програмування для реалізації проєкту Discord-бота, що виконує генерацію та обробку зображень з використанням штучного інтелекту. Цей вибір забезпечує баланс між продуктивністю, безпекою, гнучкістю та зручністю розробки.

Далі буде описано та обґрунтовано вибір середовища розробки (IDE).

Для ефективної розробки програмного забезпечення важливо обрати інтегроване середовище розробки (IDE), яке забезпечить зручний інтерфейс, інструменти для налагодження, підтримку тестування, підсвітку синтаксису, автоматичну генерацію коду, інтеграцію з системами контролю версій та ефективну роботу з платформою .NET.

Серед популярних IDE для мови програмування C# було розглянуто три основні варіанти:

– JetBrains Rider;

– Visual Studio Code;

– Microsoft Visual Studio.

JetBrains Rider – це потужне кросплатформне інтегроване середовище розробки, створене компанією JetBrains, відомою своїми інструментами для програмістів, зокрема IntelliJ IDEA та WebStorm. Rider заснований на IntelliJ-платформі та використовує ReSharper як рушій для аналізу коду. Основні переваги Rider:

– кросплатформеність (Windows, macOS, Linux);

– висока швидкість пошуку, навігації та рефакторингу;

– потужний інструментарій аналізу коду завдяки інтегрованому ReSharper;

– вбудована підтримка Git, Docker, Unity, Azure, підтримка тестових фреймворків;

– зручна робота з проектами ASP.NET Core, .NET 8.

Проте Rider має також низку недоліків:

– платна ліцензія (лише обмежена студентська або пробна версія безкоштовна);

– вища вимогливість до ресурсів комп’ютера;

– у деяких випадках можуть виникати проблеми з повною підтримкою останніх оновлень .NET одразу після їхнього релізу.

На рисунку 2.2 зображено інтерфейс користувача JetBrains Rider.

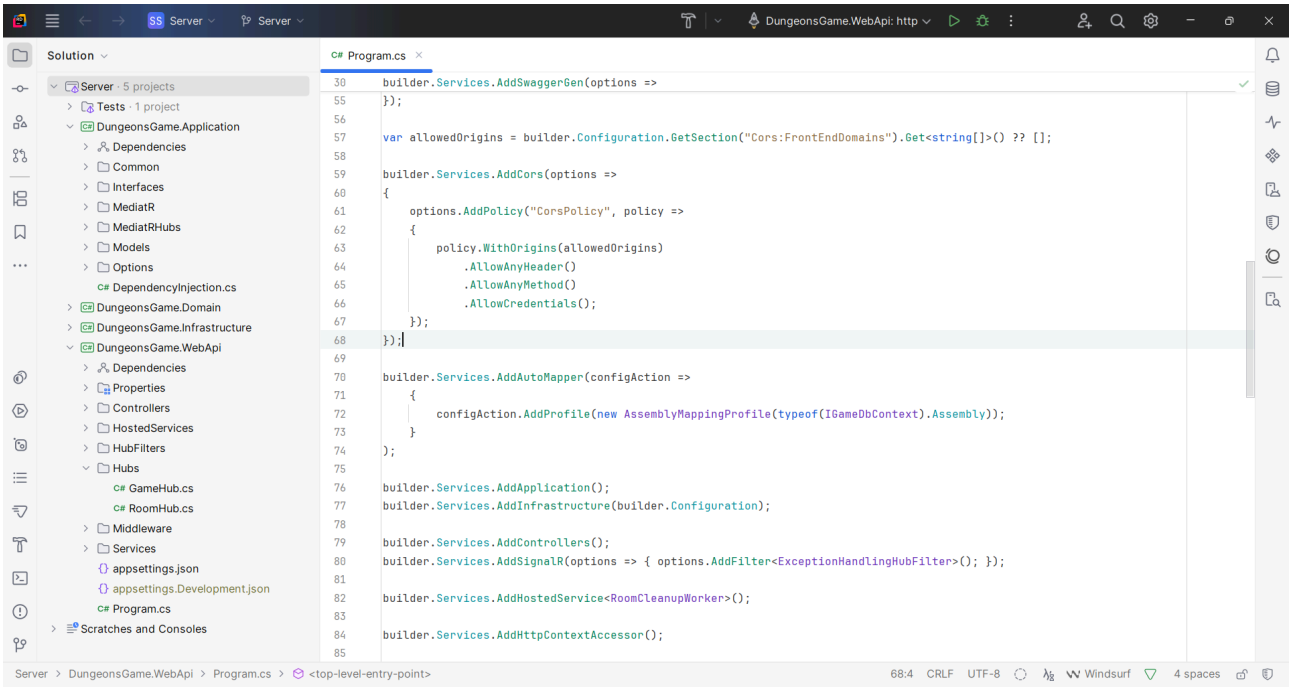

Рисунок 2.2 – Інтерфейс користувача JetBrains Rider

Visual Studio Code (VS Code) – це легке, безкоштовне, кросплатформне редакторське середовище, яке підтримує C# через розширення (наприклад, C# від Microsoft, OmniSharp тощо). Основні переваги VS Code:

– висока швидкість роботи, мінімальне споживання ресурсів;

– велика кількість плагінів для різних мов та платформ;

– потужна інтеграція з Git;

– кросплатформеність (Windows, Linux, macOS);

– автоматичне оновлення;

– безкоштовність.

Однак VS Code є текстовим редактором, а не повноцінним IDE. Через це існують обмеження:

– обмежені можливості для налагодження складних проєктів;

– менша глибина інтеграції з .NET, ніж у повноцінних IDE;

– відсутність вбудованого інструментарію для тестування, візуального дизайну, UI-компонентів тощо (усе це потребує встановлення сторонніх розширень);

– необхідність додаткової конфігурації для запуску повного циклу розробки.

На рисунку 2.3 зображено інтерфейс користувача Visual Studio Code.

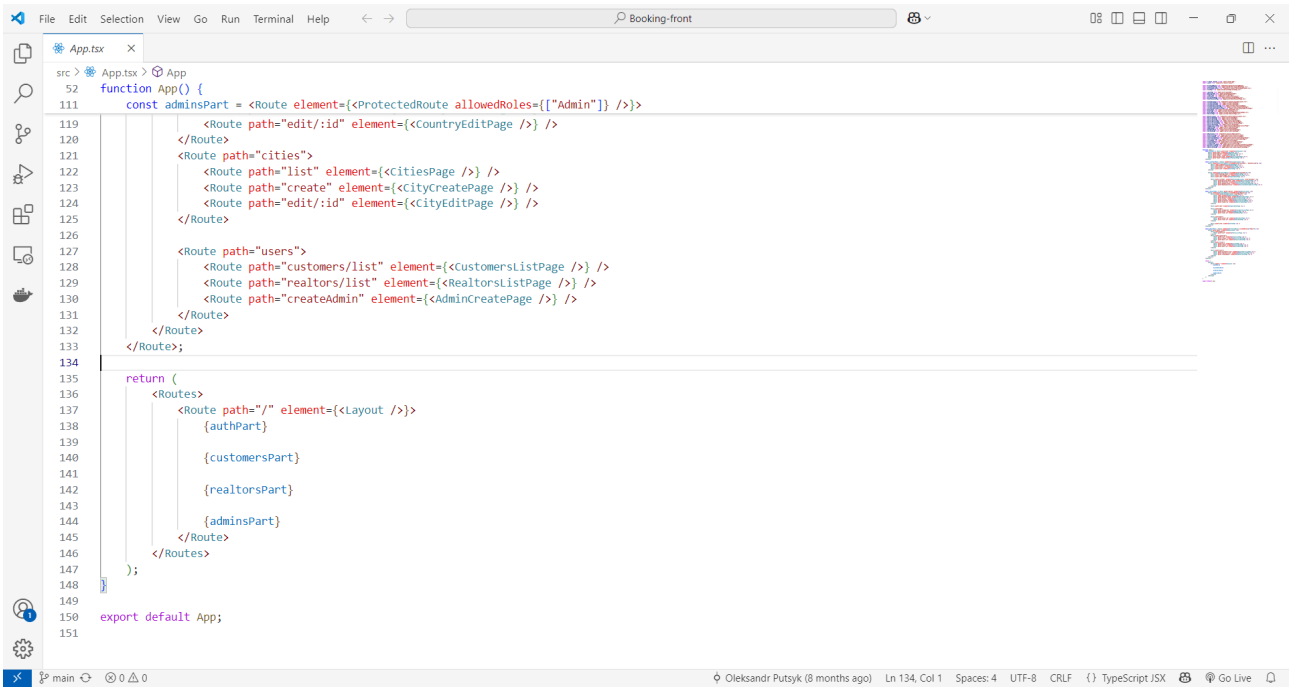

Рисунок 2.3 – Інтерфейс користувача Visual Studio Code

Visual Studio – офіційне IDE від Microsoft для розробки на платформі .NET [12]. Саме ця IDE має найглибшу інтеграцію з C# і .NET Core/.NET 6/7/8, а також підтримує всі сучасні технології Microsoft. Серед її ключових переваг:

– повна підтримка C#, Razor, XAML, .NET, Blazor, ASP.NET Core;

– потужний візуальний інтерфейс для побудови інтерфейсів, налаштування, запуску і відлагодження проєктів;

– вбудовані засоби для тестування, налагодження, профілювання;

– інтеграція з Git та GitHub;

– підтримка систем CI/CD, Azure DevOps, Docker;

– інтеграція з Azure та іншими хмарними сервісами Microsoft;

– гнучке середовище для написання та запуску юніт-тестів (з підтримкою MSTest, xUnit, NUnit);

– безкоштовна версія Community для студентів, викладачів та невеликих команд;

– постійні оновлення та підтримка нових версій .NET з першого дня релізу.

Недоліки Visual Studio:

– працює лише на Windows (Visual Studio for Mac підтримує лише частину функціоналу);

– вища вимогливість до ресурсів, ніж VS Code;

– інколи спостерігаються затримки при запуску великих проєктів.

Щоб обрати найбільш доцільне інтегроване середовище розробки було виведено наступні вимоги до Discord-бота:

– повноцінна підтримка C# та .NET 8;

– зручна робота з залежностями, пакетами NuGet, ресурсами й API;

– швидке налагодження та рефакторинг;

– можливість налаштування слеш-команд Discord-бота, обробки HTTP-запитів, роботи з графікою та інтеграції сторонніх SDK.

На рисунку 2.4 зображено інтерфейс користувача Visual Studio.

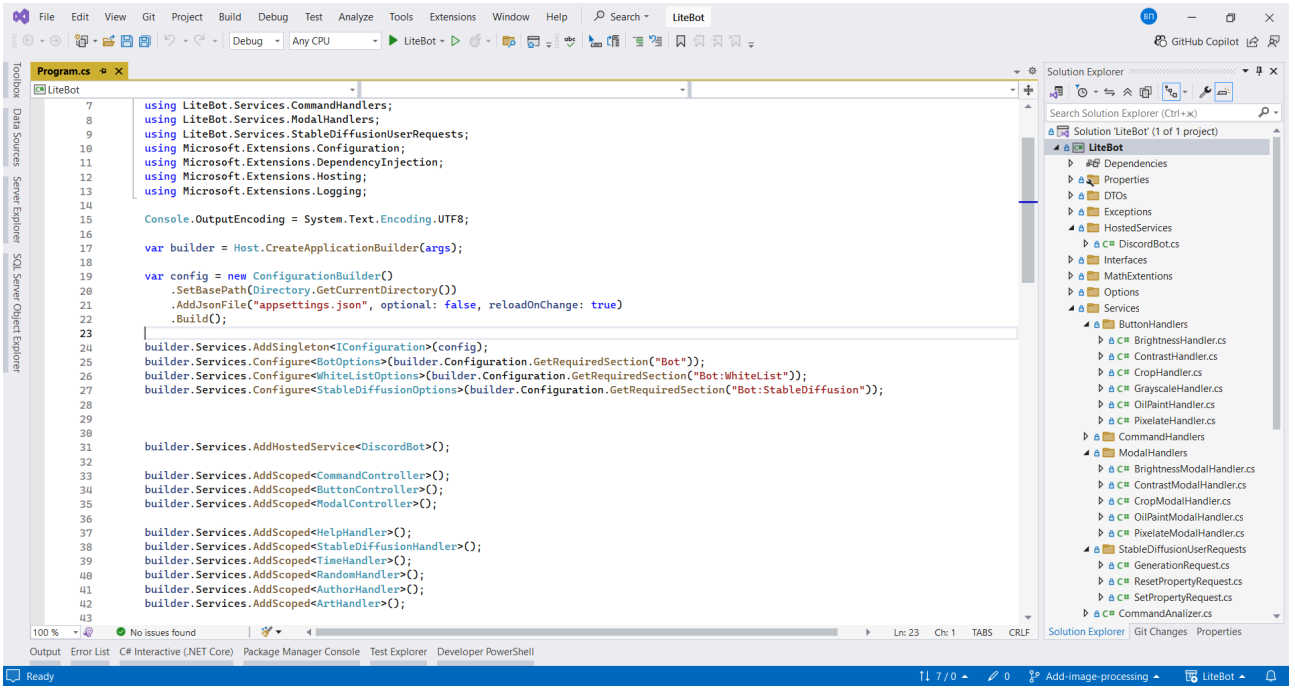

Рисунок 2.4 – Інтерфейс користувача Visual Studio

Найбільш доцільним вибором стало використання середовища Visual Studio 2022 Community Edition. Воно безкоштовне для студентів, має повну інтеграцію з .NET-платформою, сучасний інтерфейс, підтримує всі необхідні технології, а також забезпечує стабільну і ефективну розробку складних серверних додатків.

Таким чином, Visual Studio було обрано як основне середовище розробки для реалізації функціональності Discord-бота, генерації зображень за допомогою штучного інтелекту, та виконання постобробки отриманих результатів.

Далі в даному розділі буде описано огляд та обґрунтування систем контролю версій.

Система контролю версій (VCS, Version Control System) – це важливий інструмент для розробки програмного забезпечення, який дозволяє зберігати історію змін у коді, працювати над проєктом у команді, відстежувати баги, здійснювати паралельну розробку, створювати гілки для нових функцій і повертатися до стабільних версій проєкту при необхідності.

У процесі аналізу та вибору засобів було розглянуто декілька популярних VCS:

– Apache Subversion;

– Mercurial;

– Git.

Першим на розгляді буде Apache Subversion (SVN).

Apache Subversion – централізована система контролю версій. Вона передбачає наявність одного центрального сховища, де зберігається весь код. Серед її переваг:

– просте управління дозволами;

– централізований контроль змін;

– зрозуміла структура репозиторіїв;

– часто використовується у корпоративному середовищі.

Однак дана система має і суттєві недоліки:

– залежність від центрального сервера, що унеможливлює виконання операцій коли сервер не доступний;

– відсутність гнучкої роботи з гілками та злиттям (merge), порівняно з розподіленими системами;

– повільна робота з великими проєктами;

– менш зручна для індивідуальної чи відкритої командної розробки.

Наступним на розгляді буде Mercurial (hg).

Mercurial – розподілена система контролю версій, схожа за концепцією на Git. Вона має такі переваги:

– простий інтерфейс;

– висока швидкість виконання базових операцій;

– рівномірна продуктивність під час роботи з великими проєктами;

– краща підтримка Windows порівняно з Git.

Проте:

– менш поширена серед проєктів з відкритим вихідним кодом;

– обмежена кількість навчальних матеріалів і документації;

– слабша інтеграція з хмарними сервісами на кшталт GitHub чи GitLab.

Останнім на розгляді буде Git.

Git – найпоширеніша на сьогоднішній день розподілена система контролю версій, створена Лінусом Торвальдсом [13]. Вона є де-факто стандартом у сфері програмної інженерії.

Переваги Git:

– повна розподіленість, оскільки кожен розробник має локальну копію репозиторію;

– гнучка та швидка робота з гілками (branches), злиттями (merge), збереженням стану коду (commit, stash);

– широка підтримка сервісів, серед яких GitHub, GitLab, Bitbucket та Azure Repos;

– потужні інструменти інтеграції в IDE (Visual Studio, Rider, VS Code);

– активна спільнота, велика кількість навчальних матеріалів;

– можливість налаштування CI/CD, pull-requests, перевірок коду.

Серед недоліків Git іноді відзначають:

– вищу складність при першому знайомстві;

– потужність командної строки може бути складною для новачків.

З огляду на вимоги до ефективного управління версіями коду, можливості розподіленої командної роботи, інтеграцію з хмарними сервісами, підтримку з боку IDE та хостинг-платформ, система Git була обрана для даного проєкту як найбільш гнучке, масштабоване та поширене рішення.

Крім того, Git надає можливість використовувати такі сервіси, як GitHub, для розміщення репозиторію, реалізації pull-запитів, рецензій коду, а також безперервної інтеграції (CI), що підвищує якість і стабільність проєкту.

Таким чином, для забезпечення ефективної роботи над Discord-ботом, що генерує зображення із застосуванням ШІ, була обрана система Git.

Далі в розділі буде описано вибір хмарного сервісу для збереження Git-репозиторію.

Після вибору Git як системи контролю версій наступним кроком стало визначення хмарної платформи для розміщення репозиторію. Такий сервіс повинен забезпечувати безпечне зберігання коду, зручний доступ до історії змін, підтримку командної розробки, інструменти для перевірки та рецензії коду, інтеграцію з CI/CD, а також мати хорошу інтеграцію з популярними середовищами розробки.

Було розглянуто декілька провідних сервісів:

– GitHub;

– GitLab;

– Bitbucket.

GitHub є найпопулярнішою хмарною платформою для розміщення Git-репозиторіїв у світі.

Платформа підтримує:

– публічні та приватні репозиторії;

– зручну систему управління гілками, pull-запитами (pull requests), перевірками та злиттям;

– інтеграцію з GitHub Actions для реалізації CI/CD;

– можливість створення документації у вигляді wiki;

– широку підтримку спільноти та плагінів;

– інтеграцію з IDE (Visual Studio, Rider, VS Code тощо);

– захищений доступ через SSH, OAuth, PAT.

Крім того, GitHub активно підтримує open-source-проєкти та має зручний веб-інтерфейс для роботи навіть без встановлення локальних інструментів.

GitLab – це альтернатива GitHub з більшою орієнтацією на приватні корпоративні проєкти.

Його переваги:

– потужна вбудована система CI/CD;

– повна автономність, завдяки можливості установки на власний сервер;

– вбудоване відстеження задач (issues) і Kanban-дошки;

– гнучке керування ролями та доступом.

Недоліки:

– менш інтуїтивний інтерфейс порівняно з GitHub;

– у безкоштовній версії обмежена кількість CI/CD-хвилин;

– менш активна спільнота open-source.

Bitbucket – сервіс від Atlassian, інтегрований з іншими продуктами цієї компанії (Jira, Confluence тощо).

Серед переваг:

– хороша інтеграція з Jira;

– підтримка приватних репозиторіїв у безкоштовній версії;

– вбудований Pipelines (аналог CI/CD).

Недоліки:

– менш активна спільнота;

– менше навчальних матеріалів;

– не така широка підтримка серед сторонніх сервісів і плагінів.

На основі аналізу функціональності, підтримки, зручності використання та популярності було обрано GitHub як хмарну платформу для зберігання репозиторію проєкту.

Серед вирішальних факторів:

– висока популярність і надійність;

– простий інтерфейс та інтеграція з Visual Studio;

– можливість використання GitHub Actions для автоматизації;

– підтримка markdown-файлів і документації;

– широка підтримка інструментів CI/CD, хостингу сторінок (GitHub Pages) та інтеграцій із Discord.

Таким чином, GitHub став логічним вибором для збереження коду, ведення історії змін та забезпечення безперебійної роботи над проєктом.

Однією з ключових частин розробки Discord-бота є взаємодія з Discord API, що забезпечує можливість обробки повідомлень, команд, взаємодії з каналами, серверами, користувачами тощо. Для зручності та ефективності

розробки зазвичай використовуються готові бібліотеки, які інкапсулюють низькорівневу роботу з HTTP-запитами та WebSocket-з’єднанням.

Для мови програмування C# існує декілька популярних бібліотек для створення Discord-ботів:

– Discord.Net;

– DSharpPlus;

– Remora.Discord.

Discord.Net – найвідоміша й найпопулярніша бібліотека на C# для роботи з Discord API.

Вона забезпечує:

– повну реалізацію Discord REST API і Gateway;

– зручну систему команд (CommandService);

– підтримку подій (message received, user joined тощо);

– підтримку інтерактивних повідомлень, slash-команд та реакцій;

– хорошу документацію та підтримку спільноти;

– можливість роботи в асинхронному середовищі (на базі Task/async-await).

Ця бібліотека є стабільною, активно підтримується, має високий рівень абстракції й дозволяє легко реалізовувати навіть складну логіку в ботах.

DSharpPlus – ще одна популярна бібліотека для C#, яка має схожий функціонал.

Дана бібліотека має наступні особливості:

– пропонує сучасний API;

– має модульну архітектуру (поділена на core, commands, interactivity, voice тощо);

– підтримує Slash-команди, інтерактивність, а також голосовий функціонал.

Однак документація DSharpPlus менш розгорнута, а активність спільноти дещо нижча, ніж у Discord.Net.

Remora.Discord – відносно нова, але перспективна бібліотека, що прагне бути строго типізованою й орієнтованою на сучасні підходи .NET. Проте, через свою складність та малу кількість прикладів вона не є оптимальним вибором для навчального чи кваліфікаційного проєкту.

На основі порівняльного аналізу було обрано Discord.Net як основну бібліотеку для взаємодії з Discord API. Вибір обумовлений такими перевагами:

– високий рівень стабільності та підтримка широкого спектру можливостей Discord API;

– зручна робота з командами, подіями та реакціями;

– наявність великої кількості прикладів, документації, підтримка спільноти;

– хороша інтеграція з .NET-екосистемою та підтримка сучасних версій платформи (.NET 6+);

– підтримка асинхронного програмування, що є важливою складовою у розробці масштабованих ботів.

Це робить Discord.Net найбільш доцільним інструментом для реалізації функціональності Discord-бота в межах кваліфікаційної роботи.

Далі на розгляді буде вибір API для генерації зображень.

Одним із ключових елементів розроблюваної системи є модуль генерації зображень на основі текстових запитів користувача. Для реалізації цієї функціональності доцільно використовувати програмно доступне API, яке надає можливості для роботи з сучасними генеративними моделями зображень, зокрема на базі архітектури Stable Diffusion.

На ринку доступно кілька підходів до інтеграції таких моделей:

– використання хмарних API (наприклад, від Stability.ai, Hugging Face, Replicate тощо) забезпечує високу доступність і не потребує локальних обчислювальних ресурсів, але має низку обмежень. Серед них – платна модель використання, обмеження на кількість запитів, залежність від стабільності інтернет-з’єднання, а також обмежені можливості кастомізації параметрів генерації;

– локальні рішення з доступом через API дають змогу мати повний контроль над процесом генерації, забезпечують повну конфіденційність, можливість налаштовувати модель, оптимізувати продуктивність відповідно до ресурсів системи. Основною умовою використання таких підходів є наявність потужного графічного процесора (GPU) та налаштування середовища для запуску моделі.

Після порівняльного аналізу було обрано локальну реалізацію Stable Diffusion – stable-diffusion-webui від AUTOMATIC1111, яка є однією з найпопулярніших, активно підтримується спільнотою та має зручний вбудований API для взаємодії з іншими програмами [14].

Основні причини вибору stable-diffusion-webui:

– відкрите програмне забезпечення з активним розвитком і великою кількістю підтримуваних розширень;

– можливість тонкого налаштування параметрів генерації, таких як steps, cfg_scale, sampler, вибір моделей і чекпоїнтів;

– наявність API-інтерфейсу з підтримкою генерації зображень за текстовим описом (text-to-image), модифікації зображень (image-to-image), інверсії, upscaling тощо;

– підтримка широкого спектра моделей та LoRA-розширень;

– можливість повної локалізації, що критично для збереження приватності й незалежності від зовнішніх серверів;

– зручне управління чергами запитів, контроль використання GPU, інтеграція з іншими модулями.

З технічної точки зору, взаємодія з цією реалізацією здійснюється через HTTP API. Discord-бот надсилає запити на локальний сервер stable-diffusion-webui, передаючи параметри генерації. У відповідь API повертає зображення у форматі base64 або збережене у вказаній директорії, звідки воно потім передається користувачу через Discord.

Таким чином, обраний підхід забезпечує максимальну гнучкість, розширюваність і відповідність функціональним вимогам проєкту без залежності від сторонніх сервісів.

Висновки до розділу 2

У цьому розділі було обґрунтовано архітектуру Discord-бота для генерації та постобробки зображень із використанням штучного інтелекту, визначено функціональні та нефункціональні вимоги до системи, а також спроєктовано основні сценарії її використання. Було обрано клієнт-серверну модель із централізованим бекендом, сформовано перелік ключових функцій бота, визначено вимоги до інтерфейсу, безпеки та комунікації з користувачем. Для візуалізації сценаріїв взаємодії створено UML-діаграму прецедентів.

Також у даному розділі було здійснено аналіз і обґрунтований вибір програмних засобів, методів та інструментів, необхідних для реалізації кваліфікаційної роботи. Розглянуто кілька популярних мов програмування, серед яких обрано C# як оптимальне рішення завдяки підтримці асинхронного програмування, суворій типізації, високій продуктивності, сучасній екосистемі та кросплатформенності.

Серед середовищ розробки було проаналізовано Visual Studio, Rider та Visual Studio Code. У підсумку, для реалізації проєкту обрано Visual Studio, яка забезпечує глибоку інтеграцію з .NET, потужні засоби відлагодження та високу стабільність роботи.

Як систему контролю версій обрано Git, що дозволяє ефективно відстежувати зміни у коді, працювати в команді та підтримує безліч хостингових сервісів. Для розміщення репозиторію використано GitHub – найбільш популярну хмарну платформу з широким набором інструментів і підтримкою спільної розробки.

Для взаємодії з Discord API обрана бібліотека Discord.Net, яка найкраще інтегрується з C# і .NET, підтримує асинхронну модель та забезпечує високу стабільність і документованість.

Важливим компонентом є механізм генерації зображень. Проведено огляд хмарних і локальних API, після чого було обрано локальну реалізацію stable-diffusion-webui від AUTOMATIC1111 з її API як найгнучкіше, розширюване та незалежне рішення для генерації зображень.

Таким чином, сформовано повний набір технологічних рішень, які забезпечать ефективну реалізацію поставленого завдання в межах кваліфікаційної роботи.

РОЗДІЛ 3
РОЗРОБКА DISCORD-БОТА ДЛЯ ГЕНЕРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ І МОЖЛИВІСТЮ ПОСТОБРОБКИ

3.1 Практична реалізація об’єкта проєктування

На даному етапі розробки було реалізовано Discord-бота з використанням мови програмування C# та бібліотеки Discord.Net, що дозволяє взаємодіяти з API Discord. Реалізація бота виконувалась у середовищі розробки Visual Studio з використанням платформи .NET 8, що забезпечує високу продуктивність та підтримку сучасних підходів до побудови серверного програмного забезпечення.

Основна логіка ініціалізації програми відбувається у методі Main, який у сучасних версіях .NET реалізується через створення об’єкта Host. У цьому методі здійснюється конфігурація залежностей за допомогою вбудованого механізму DI-контейнера [15]. У лістингу 3.1 наведено фрагмент коду, який відповідає за реєстрацію сервісів та конфігурацію.

Лістинг 3.1 – Фрагмент методу Main

```
var builder = Host.CreateApplicationBuilder(args);

var config = new ConfigurationBuilder()
      .SetBasePath(Directory.GetCurrentDirectory())
      .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
      .Build();

builder.Services.AddSingleton<IConfiguration>(config);
builder.Services.Configure<BotOptions>(builder.Configuration.GetRequired
Section("Bot"));
builder.Services.Configure<WhiteListOptions>(builder.Configuration.GetRe
quiredSection("Bot:WhiteList"));
builder.Services.Configure<StableDiffusionOptions>(builder.Configuration
.GetRequiredSection("Bot:StableDiffusion"));

builder.Services.AddHostedService<DiscordBot>();

builder.Services.AddScoped<CommandController>();
```

```
builder.Services.AddScoped<ButtonController>();
builder.Services.AddScoped<ModalController>();
```

кінець лістингу 3.1

За допомогою DI (Dependency Injection) до контейнера додаються як окремі сервіси (обробники команд, налаштування бота), так і життєвий цикл самого Discord-бота, реалізованого як фоновий сервіс IHostedService.

Клас DiscordBot реалізує інтерфейс IHostedService і відповідає за підключення до Discord-серверів, обробку повідомлень, кнопок та форм. У методі StartAsync, тіло котрого наведено в лістингу 3.2, відбувається ініціалізація клієнта Discord.

Лістинг 3.2 – Метод StartAsync

```
_client = new DiscordSocketClient(
      new DiscordSocketConfig {
            GatewayIntents = GatewayIntents.All
      }
);

_client.MessageReceived += HandleCommandAsync;
_client.ButtonExecuted += HandleButtonAsync;
_client.ModalSubmitted += ModalSubmittedAsync;
_client.Log += LogAsync;
_client.Ready += ReadyAsync;

await _client.LoginAsync(TokenType.Bot, GetTokenOrThrow());
await _client.StartAsync();
```

кінець лістингу 3.2

Взаємодія з користувачем відбувається через текстові команди. Для обробки команд реалізований контролер CommandController, який визначає, який саме обробник (ICommandHandler) повинен бути викликаний. Наприклад, якщо користувач надсилає команду «sd», відповідальність за обробку покладається на StableDiffusionHandler. Фрагмент алгоритму отримання обробника команди наведено в лістингу 3.3.

Лістинг 3.3 – Фрагмент алгоритму отримання обробника текстової команди

```
if (commandName == "sd") {
      return
serviceProvider.GetRequiredService<StableDiffusionHandler>();
}
```

кінець лістингу 3.3

Команди обробника StableDiffusionHandler дозволяють користувачу взаємодіяти з сервером генерації зображень Stable Diffusion. Наприклад, можна змінити параметри генерації (ширину, висоту, prompt, кількість кроків тощо). Якщо команда не містить параметрів, то буде виконана генерація зображення з уже встановленими налаштуваннями. Алгоритм обробки команди без параметрів наведено в лістингу 3.4.

Лістинг 3.4 – Алгоритм обробки команди без параметрів

```
var arguments = commandInfo.Argument;

if (arguments == string.Empty) {
      stableDiffusionQueue
            .Enqueue(
                  serviceProvider.GetRequiredService<GenerationRequest>()
            );
}
```

кінець лістингу 3.4

Також обробник підтримує допоміжну команду «?», яка виводить інструкцію по використанню всіх доступних підкоманд. Реалізація обробки даної команди наведено в лістингу 3.5.

Лістинг 3.5 – Алгоритм обробки команди допомоги

```
else if (arguments == "?") {
      await HelpMessageAsync();
}
```

кінець лістингу 3.5

Для черги запитів на генерацію використовується клас StableDiffusionQueue, який дозволяє асинхронно обробляти запити з тривалим виконанням в окремому потоці, запобігаючи перевантаженню сервера. Лістинг 3.6 містить код реалізації класу StableDiffusionQueue.

Лістинг 3.6 – Реалізація класу StableDiffusionQueue

```
public class StableDiffusionQueue(
      ILogger<StableDiffusionQueue> logger,
      IOptions<StableDiffusionOptions> options
) : IStableDiffusionQueue {
      private readonly StableDiffusionOptions _stableDiffusionOptions =
options.Value;
      private readonly ConcurrentQueue<IStableDiffusionUserRequest> queue
= new();
      private readonly SemaphoreSlim semaphore = new(1, 1);

      public void Enqueue(IStableDiffusionUserRequest request) {
            if (queue.Count >= _stableDiffusionOptions.QueueMaxSize)
                  throw new StableDiffusionQueueOverflowException();

            queue.Enqueue(request);
            _ = ProcessQueueAsync();
      }

      private async Task ProcessQueueAsync() {
            if (!semaphore.Wait(0))
                  return;

            try {
                  while (queue.TryDequeue(out var request)) {
                        try {
                              await request.ExecuteAsync();
                        }
                        catch (Exception ex) {
                              logger.LogError(ex, "Error while processing
request");
                        }
                  }
            }
            finally {
                  semaphore.Release();
            }
      }
}
```

кінець лістингу 3.6

Клас StableDiffusionApi є ключовим компонентом, що забезпечує інтеграцію Discord-бота з зовнішнім сервером генерації зображень, який працює на базі моделі Stable Diffusion. Даний клас реалізує інтерфейс IStableDiffusionApi і виконує дві основні функції:

– надсилання запитів на генерацію зображень (GenerateImagesAsync);

– отримання прогресу поточного запиту (GetProgressAsync).

StableDiffusionApi виконує роль зовнішнього шлюзу між ботом і сервером генерації зображень. Саме через нього бот надсилає запити на створення зображень за текстовим описом і дізнається, коли процес буде завершено. Його виняткова відповідальність за ці дві операції допомагає зберігати чисту архітектуру та спрощує тестування або заміну бекенду в майбутньому. Лістинг 3.7 містить код реалізації класу StableDiffusionApi.

Лістинг 3.7 – Реалізація класу StableDiffusionApi

```
public class StableDiffusionApi(
     IOptions<StableDiffusionOptions> stableDiffusionOptions
) : IStableDiffusionApi {

     private readonly StableDiffusionOptions _stableDiffusionOptions =
stableDiffusionOptions.Value;

     public async Task<Txt2ImgResponseDto>
GenerateImagesAsync(Txt2ImgRequestDto postDto, CancellationToken
cancellationToken = default) {
          using HttpClient client = new();
          client.Timeout =
TimeSpan.FromSeconds(_stableDiffusionOptions.ImageGenerationTimeoutInSec
onds);
          var httpResponseMessage = await
client.PostAsJsonAsync($"{_stableDiffusionOptions.ApiUrl}txt2img",
postDto, cancellationToken);

          var stringResponse = await
httpResponseMessage.Content.ReadAsStringAsync(cancellationToken);

          return
JsonConvert.DeserializeObject<Txt2ImgResponseDto>(stringResponse)
               ?? throw new
NullReferenceException($"{nameof(StableDiffusionApi)}.{nameof(GenerateIm
agesAsync)}");
     }
```

```
      public async Task<ProgressResponseDto>
GetProgressAsync(CancellationToken cancellationToken = default) {
            using HttpClient client = new();
            string responseContent = await
client.GetStringAsync($"{_stableDiffusionOptions.ApiUrl}progress",
cancellationToken);

            return
JsonConvert.DeserializeObject<ProgressResponseDto>(responseContent)
                  ?? throw new
NullReferenceException($"{nameof(StableDiffusionApi)}.{nameof(GetProgres
sAsync)}");
      }
}
```

кінець лістингу 3.7

Для побудови, налаштування та розміщення кнопок у повідомленні Discord було створено клас PostprocessingOptionsBuilder. Лістинг 3.8 містить програмний код реалізації цього класу.

Лістинг 3.8 – Реалізація класу PostprocessingOptionsBuilder

```
public class PostprocessingOptionsBuilder :
IPostprocessingOptionsBuilder {
      public ComponentBuilder GetBuilder() {
            var firstRowBuilder = new ActionRowBuilder()
                  .WithButton("Обрізати", "sd:crop")
                  .WithButton("Яскравість", "sd:brightness")
                  .WithButton("Контраст", "sd:contrast");

            var secondRowBuilder = new ActionRowBuilder()
                  .WithButton("Відтінки сірого", "sd:grayscale")
                  .WithButton("Олійний стиль", "sd:oilPaint")
                  .WithButton("Пікселізація", "sd:pixelate");

            ComponentBuilder builder = new ComponentBuilder()
                  .AddRow(firstRowBuilder)
                  .AddRow(secondRowBuilder);

            return builder;
      }
}
```

кінець лістингу 3.8

Структуру взаємодії основних модулів системи зображено на рисунку 3.1, дана діаграма демонструє, яким чином між собою пов’язані компоненти бота, зокрема класи StableDiffusionQueue та StableDiffusionApi, і як відбувається обробка запитів на генерацію зображень у межах загальної архітектури.

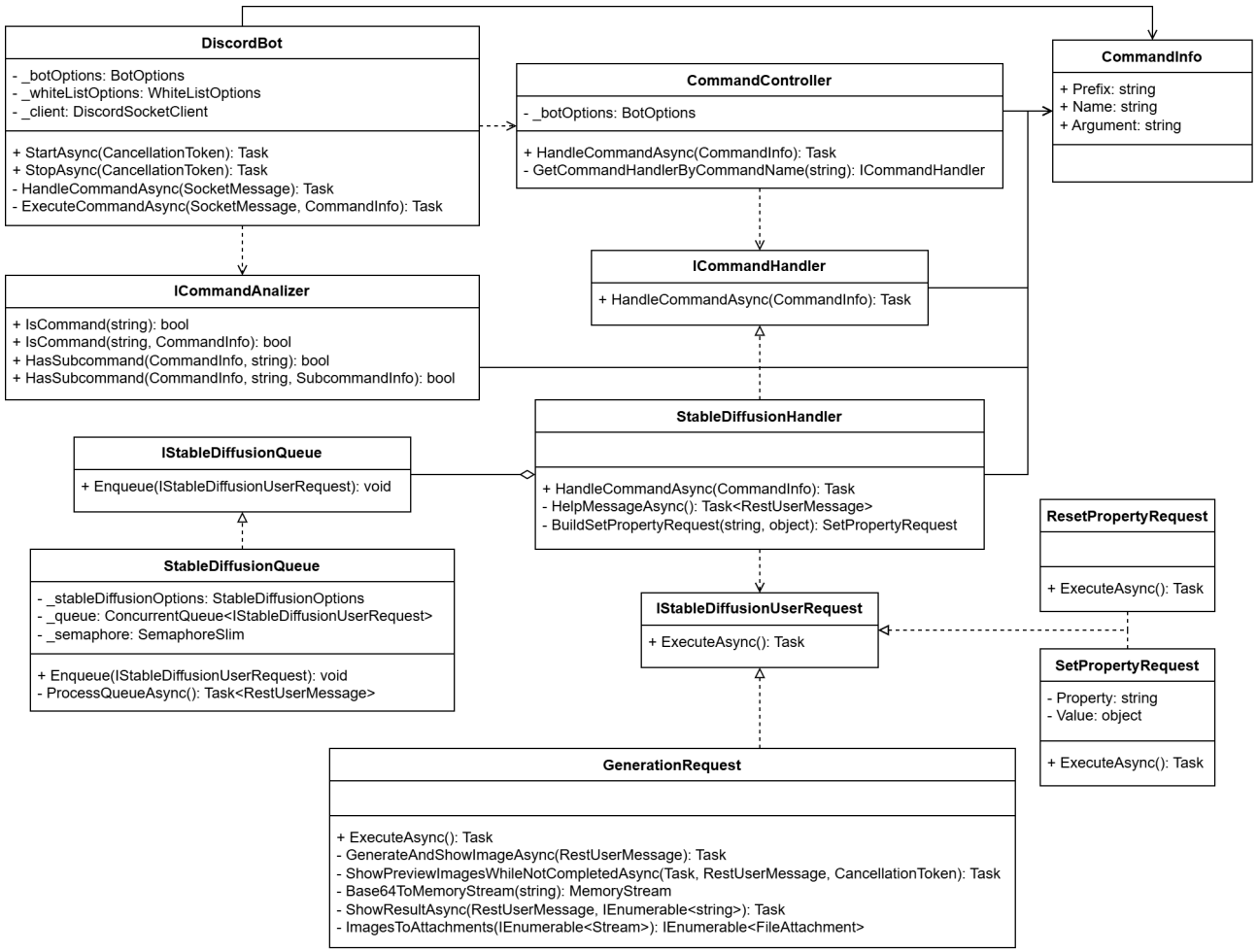

Рисунок 3.1 – UML-діаграма компонентів обробки команд Discord-бота

Важливою особливістю реалізованого програмного продукту є підтримка інтерактивної постобробки згенерованих зображень. Після завершення генерації результат прикріплюється до повідомлення користувача, а також додаються інтерактивні кнопки для застосування фільтрів. Для цього використовується спеціальний компонент PostprocessingOptionsBuilder, який формує набір кнопок, що надають можливості редагування зображення – такі як обрізка, зміна яскравості й контрастності, перетворення в чорно-біле, стилізація

під живопис або пікселізація. Всі дії постобробки реалізовані як окремі обробники кнопок або модальних вікон і визначаються через інтерфейси IButtonHandler та IModalHandler, що забезпечує розширюваність і гнучкість. Взаємодія з користувачем відбувається за допомогою класів ButtonController та ModalController, які, у свою чергу, делегують обробку відповідному сервісу, що зареєстрований у контейнері залежностей. Такий підхід дозволяє легко додавати нові типи постобробки без порушення існуючої логіки або модифікації центральних контролерів.

Реалізований Discord-бот демонструє ефективне застосування принципів об’єктно-орієнтованого програмування, шаблону впровадження залежностей (Dependency Injection) і сучасних підходів до побудови модульної архітектури в середовищі .NET. Його структура побудована таким чином, щоб забезпечити легке додавання нових функціональностей без потреби змінювати існуючий код – зокрема це стосується як команд генерації зображень, так і розширення можливостей постобробки.

Сервіс StableDiffusionApi інкапсулює взаємодію із зовнішнім сервером генерації, забезпечуючи єдину точку доступу до функціоналу штучного інтелекту. Разом із чергою StableDiffusionQueue та обробниками інтерфейсів користувача (ButtonController, ModalController), цей сервіс формує основу для асинхронної обробки запитів і динамічного налаштування взаємодії з користувачем. Така архітектура не лише спрощує розробку та тестування, а й забезпечує гнучкість, масштабованість і зручність підтримки програмного продукту в довгостроковій перспективі.

Розробку Discord-бота було успішно завершено. Усі заплановані функціональні можливості реалізовано у повному обсязі, включаючи генерацію зображень через API Stable Diffusion, механізми постобробки, інтеграцію з Discord та забезпечення безпеки взаємодії. Результатом стала функціональна, гнучка та надійна інформаційно-комп’ютерна система, готова до використання в реальних умовах.

На рисунку 3.2 представлено процес виконання текстової команди користувача та подальшу генерацію зображення засобами реалізованого Discord-бота.

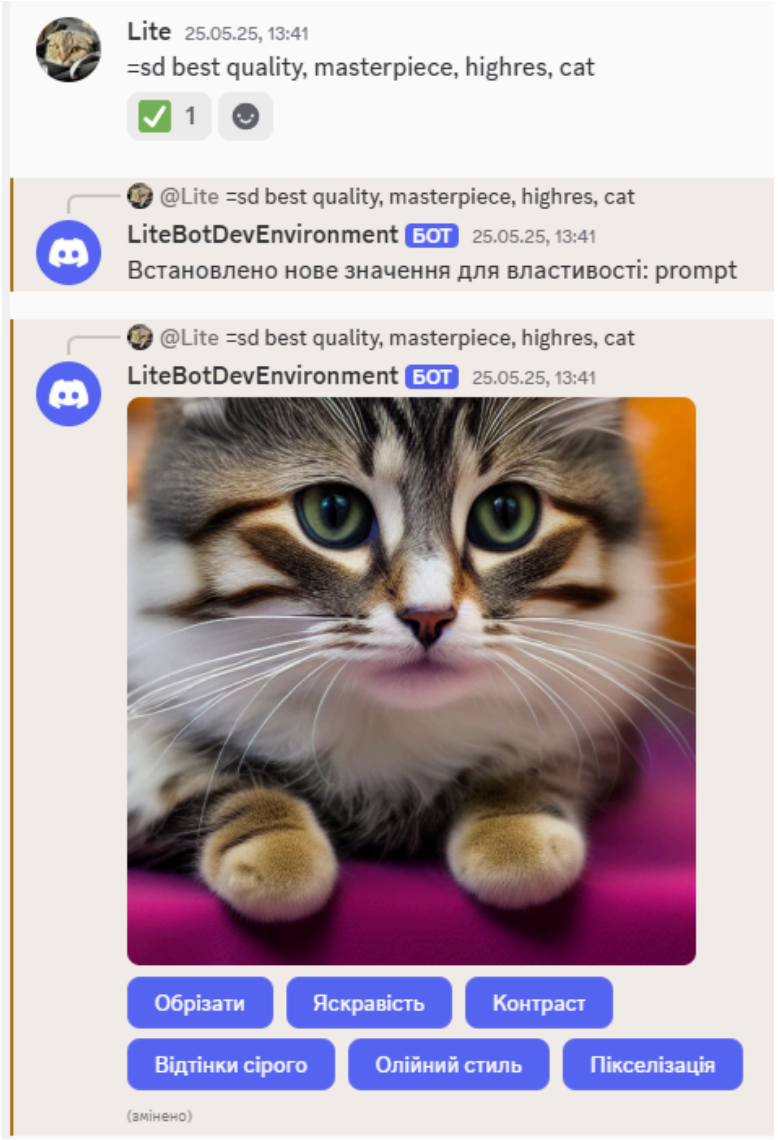

Рисунок 3.2 – Генерація зображення за текстовим запитом у Discord-боті

Після завершення генерації до повідомлення автоматично додаються кнопки, які надають користувачу можливості для подальшої постобробки зображення.

На рисунку 3.3 зображено вигляд модального вікна, яке очікує введення параметрів від користувача для подальшої обробки.

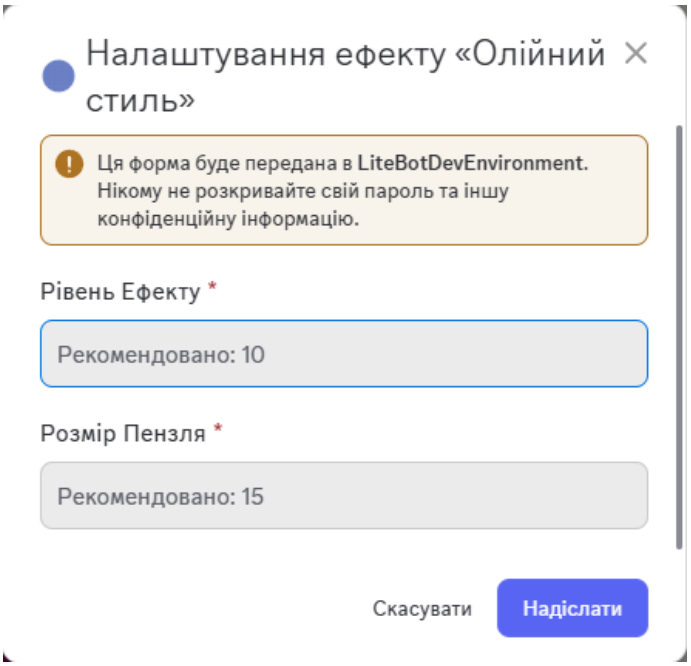

Рисунок 3.3 – Модальне вікно для введення параметрів у Discord-боті

Користувач має можливість послідовно застосовувати кілька ефектів постобробки. На рисунку 3.4 продемонстровано результати обробки з використанням ефектів «Олійний стиль» та «Відтінки сірого».

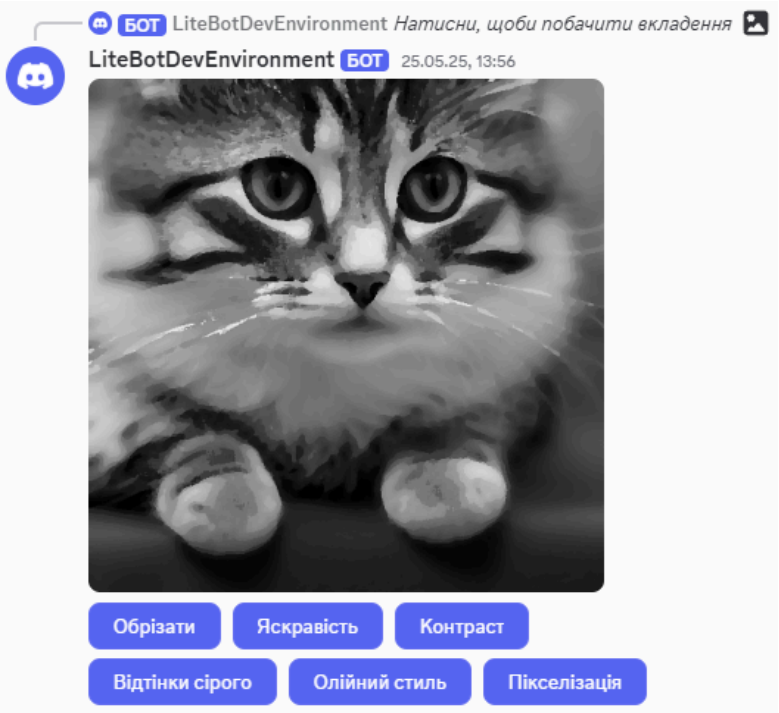

Рисунок 3.4 – Результат застосування ефектів пособробки

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Після реалізації основних функціональних компонентів інформаційно-комп’ютерної системи було проведено комплексне тестування та налагодження з метою перевірки її працездатності, відповідності технічним вимогам та очікуваній поведінці.

Основну увагу було приділено перевірці таких аспектів:

– стабільності взаємодії з API генерації зображень, зокрема правильності формування HTTP-запитів, коректної обробки JSON-відповідей і забезпечення таймаутів при надто тривалому очікуванні;

– коректності логіки обробки команд у Discord, зокрема валідації вхідних параметрів, обробці помилок та асинхронній взаємодії з користувачем;

– відображенню прогресу генерації зображень, що дозволяє користувачу бачити статус виконання запиту у реальному часі;

– наявності обробки виняткових ситуацій (наприклад, недоступності зовнішнього API, помилок десеріалізації або втрати підключення до Discord).

Для перевірки коректності функціонування застосовувались як ручні, так і автоматизовані методи тестування. Ручне тестування дозволило виявити логічні помилки в роботі бота при взаємодії з Discord у реальному часі. Було проаналізовано поведінку бота у різних сценаріях, зокрема:

– надсилання запиту на генерацію без параметрів або з некоректними параметрами;

– обробка декількох одночасних запитів від різних користувачів;

– робота в умовах обмеженої пропускної здатності мережі;

– симуляція відмови API для перевірки стійкості до збоїв.

У процесі налагодження використовувалися журнали подій (логи), які містили детальну інформацію про виконання запитів, час відповіді від сервера генерації, а також про помилки, які виникали під час роботи. Це дозволило оперативно виявляти та усувати недоліки на ранніх етапах тестування.

Загалом, результати тестування підтвердили функціональну придатність реалізованої системи. Сервіс стабільно працює при високому навантаженні, коректно обробляє запити користувачів і забезпечує зворотний зв’язок у вигляді повідомлень у Discord.

3.3 Захист інформаційно-комп’ютерної системи

У процесі проєктування та реалізації інформаційно-комп’ютерної системи було враховано базові вимоги до забезпечення її захищеності, зокрема на етапах автентифікації, авторизації, передачі даних і контролю доступу до функціональності бота.

У системі не реалізовано окремої автентифікації користувачів, оскільки автентифікація та авторизація відбувається через інфраструктуру Discord. При запуску Discord-бота він проходить автентифікацію шляхом передачі токена, який відповідає за ідентифікацію застосунку у системі Discord.

Цей токен генерується на платформі Discord Developer Portal і зберігається в конфігураційному файлі. Даний токен не був вбудований у вихідний код з міркувань безпеки.

Ідентифікація користувача в межах обробки запитів здійснюється за унікальним Discord-ідентифікатором автора повідомлення, що надається Discord API. Це дозволяє:

– точно визначати, хто ініціював команду;

– обмежувати доступ до певних функцій за Discord-ролями або ID;

– уникати необхідності створення окремої системи автентифікації.

Уся взаємодія між Discord-ботом і серверами Discord відбувається через зашифрований протокол HTTPS на базі TLS (Transport Layer Security). Даний протокол забезпечує:

– конфіденційність переданих даних;

– захист від атак типу «людина посередині» (MITM);

– перевірку справжності сервера.

Цей рівень шифрування забезпечується автоматично бібліотекою Discord.NET, яка використовує стандартні засоби платформи .NET для роботи з мережею та SSL-сертифікатами.

При взаємодії з API Stable Diffusion використовується протокол HTTP, що є доцільним у випадках, коли сервер генерації зображень розгорнутий локально – на тій самій машині, що й бот. У такому середовищі HTTP забезпечує вищу швидкість обміну даними завдяки відсутності накладних витрат на шифрування.

Для підключення до віддалених або сторонніх серверів API передбачено можливість використання захищеного протоколу HTTPS. Для цього достатньо вказати відповідну схему (https://) у конфігураційному файлі бота, що дозволяє забезпечити конфіденційність і цілісність переданих даних при роботі через мережу.

Якщо зовнішній API не підтримує шифрування (тобто працює через HTTP), існує ризик компрометації даних. У такому разі рекомендується:

– обмежити доступ до API ззовні через фаєрволи;

– використовувати тунелювання трафіку або VPN;

– перевести API на HTTPS, використовуючи власний TLS-сертифікат.

З міркувань безпеки:

– бот не зберігає паролі користувачів;

– не виконує команди Shell або завантаження стороннього коду;

– має обмеження прав у межах Discord-серверу, на якому працює (наприклад, не може видаляти повідомлення або додавати учасників, якщо це не передбачено);

– обробка помилок реалізована з урахуванням захисту від викриття деталей внутрішньої реалізації (наприклад, помилки не містять стек трейсів у відповіді користувачу).

Також важливо періодично оновлювати залежності та бібліотеки, щоб уникнути відомих вразливостей у програмному забезпеченні.

3.4 Розробка документації для програмного продукту

Для ефективного використання та подальшої підтримки Discord-бота, призначеного для генерації зображень за допомогою API Stable Diffusion, було створено супровідну документацію, яка охоплює як мінімальні вимоги до середовища, так і практичні інструкції з налаштування, встановлення та запуску програмного продукту. Документація також містить довідкові матеріали, які можуть бути корисними для користувачів.

Для коректної роботи бота користувачеві необхідно мати комп’ютер або сервер з операційною системою Windows 10 або однією з сучасних версій Linux, зокрема Ubuntu 20.04 або новішою. Системні ресурси повинні включати процесор архітектури x64 з двома або більше ядрами частотою від 2.0 ГГц, щонайменше 4 ГБ оперативної пам’яті, хоча рекомендується 8 ГБ і більше для стабільної роботи. Також обов’язковою умовою є наявність стабільного підключення до Інтернету, що забезпечує доступ до Discord API, а за потреби – і до віддалених серверів генерації зображень. Для запуску програмного коду потрібно встановити середовище розробки .NET SDK версії 8.0 або новішої. Крім того, необхідно мати Discord-акаунт з відповідними правами доступу для створення і налаштування бота.

Перед запуском бота потрібно створити конфігураційний файл appsettings.json, який має бути розміщений у кореневій директорії проєкту. Для зручності в репозиторії передбачено файл-зразок appsettings.example.json, що містить необхідні секції з прикладами заповнення. Після створення основного файлу конфігурації користувач має заповнити параметри, зокрема вказати токен Discord-бота, отриманий у Discord Developer Portal. За бажанням можна увімкнути механізм обмеження доступу через білий список каналів, де дозволено використання бота. Для цього активується функція WhiteList і зазначаються ідентифікатори дозволених каналів.

У параметрі ApiUrl вказується адреса API-сервера Stable Diffusion. Якщо сервер розгорнуто локально, доцільно використовувати HTTP-протокол,

наприклад, http://127.0.0.1:7860/sdapi/v1/. Якщо ж бот підключається до віддаленого ресурсу, підтримується використання протоколу HTTPS. У цьому випадку в конфігурації зазначається захищена адреса сервера, наприклад, https://remote-server-domain.com/sdapi/v1/. Також при потребі можна змінити тривалість тайм-ауту очікування завершення генерації зображення та встановити максимальний розмір черги запитів.

Після завершення налаштування бот може бути запущений безпосередньо з середовища розробки за допомогою стандартних засобів .NET або ж як самостійний виконуваний файл, попередньо зібраний у процесі компіляції проєкту. Бот може функціонувати як у режимі запуску через встановлений .NET SDK, так і шляхом розпакування відповідного інсталяційного архіву в бажану директорію користувача. В обох випадках обов’язковою умовою є наявність коректно створеного та налаштованого конфігураційного файлу appsettings.json, розміщеного у кореневій папці з додатком. Цей файл забезпечує доступ до ключових параметрів роботи бота, зокрема до токена Discord, налаштувань API генерації зображень та параметрів безпеки.

3.5 Розробка інсталяційного пакета

Розробка інсталяційного пакета є завершальним етапом створення програмного забезпечення, що забезпечує зручність розповсюдження, встановлення та запуску розробленого продукту кінцевими користувачами. Для Discord-бота, який реалізує генерацію зображень за допомогою API Stable Diffusion та можливістю подальшої постобробки, було підготовлено окремі інсталяційні архіви для операційних систем Windows та Linux.

Формування цих інсталяційних пакетів здійснювалося за допомогою вбудованої функції публікації (publish) у середовищі розробки Visual Studio 2022. Проєкт було збилджено двічі – окремо під Windows та Linux, із використанням відповідних налаштувань цільової платформи. У результаті

кожна версія містить усі необхідні для запуску файли, включно з виконуваним файлом, залежностями та конфігураційними елементами.

Готові збірки було упаковано у zip-архіви, один з яких призначений для Windows, а інший – для Linux. Користувачеві достатньо розпакувати обраний архів у будь-яку зручну директорію на своїй системі. У випадку з Windows запуск програми здійснюється шляхом відкриття виконуваного файлу LiteBot.exe. Для Linux-версії запуск виконується за допомогою терміналу командою ./LiteBot, попередньо надавши файлу права на виконання, якщо це необхідно.

Такий підхід до створення інсталяційного пакета дозволяє забезпечити кросплатформність, простоту розгортання та зручність у використанні без потреби додаткового встановлення чи налаштування залежностей.

Висновки до розділу 3

У третьому розділі було розглянуто практичні аспекти реалізації, тестування та захисту створеної інформаційно-комп’ютерної системи – Discord-бота для генерації зображень за допомогою API Stable Diffusion та з можливістю постобробки. Було описано архітектуру системи, ключові компоненти програмного коду та алгоритми, які забезпечують її функціональність, включаючи взаємодію з API генерації зображень та Discord. Окрім цього, було створено UML-діаграму компонентів обробки команд Discord-бота, а також додано рисунки, що зображають результати роботи розробленого додатку.

Проведено тестування функціональних можливостей бота та перевірку його стійкості до типових ситуацій, зокрема до великої кількості паралельних запитів. Наведено способи захисту, що використовуються в системі, зокрема автентифікацію користувачів через Discord, передачу даних через захищені протоколи та захист конфігурації. Окрему увагу приділено налаштуванню програмного продукту, створенню конфігураційного файлу та запуску бота як

через .NET SDK, так і через інсталяційні архіви. Крім того, було розроблено супровідну документацію, що охоплює вимоги до середовища, інструкції з налаштування та запуску, а також довідкові матеріали. Підготовлено інсталяційні пакети для Windows та Linux, що спрощують розгортання програми кінцевими користувачами.

Таким чином, розроблений Discord-бот є завершеним програмним продуктом, що відповідає сучасним вимогам до стабільності, гнучкості та безпеки. Усі етапи його створення, включаючи реалізацію, тестування, документування та створення інсталяційних архівів, були успішно виконані, що дозволяє інтегрувати систему у реальні умови використання.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи бакалавра було реалізовано програмний засіб – Discord-бот для генерації зображень на основі моделі штучного інтелекту Stable Diffusion та можливістю постобробки зображень.

На початковому етапі проаналізували сучасні технології генерації зображень та популярні сервіси (DALL-E 2, MidJourney, Stable Diffusion, Dream by Wombo). Виявлено проблеми з постобробкою, налаштуванням і інтеграцією в онлайн-комунікації. Огляд патентів підтвердив відсутність рішень, що поєднують генерацію й редагування зображень у Discord, що свідчить про інноваційність підходу. Обґрунтовано створення нового Discord-бота з функціями генерації та постобробки зручним інтерфейсом.

Далі при виконанні завдання було спроєктовано архітектуру Discord-бота для генерації та постобробки зображень на основі ШІ, обрано клієнт-серверну модель з централізованим бекендом, визначено функціональні та нефункціональні вимоги до системи, зокрема генерацію зображень за prompt-запитами, їх збереження, редагування та взаємодію через Discord. Проведено аналіз вимог, розроблено UML-діаграму прецедентів і обґрунтовано вибір технологій для реалізації зручного й масштабованого рішення.

Після проектування архітектури та аналізу вимог було здійснено вибір оптимальних інструментів для розробки програмної системи. Як основну мову програмування обрано C#, що найкраще підходить для реалізації поставленого завдання. Для розробки на C# використано Visual Studio – сучасне, функціональне та безкоштовне середовище розробки. Після аналізу систем контролю версій було обрано Git як надійне та зручне рішення, а для хмарного зберігання репозиторію – платформу GitHub. Для інтеграції з Discord API обрано бібліотеку Discord.Net. Завершальним етапом стало визначення сервісу для генерації зображень, де найбільш придатним виявився stable-diffusion-webui від AUTOMATIC1111.

На наступному етапі було реалізовано Discord-бота з використанням попередньо обраних технологій. Наведено лістинги ключових фрагментів коду, що ілюструють основну логіку роботи застосунку. Також створено UML-діаграму класів, яка відображає взаємозв’язки між основними компонентами системи. Завершується етап демонстрацією функціонування готового застосунку.

Для поліпшення роботи бота та усунення недоліків було проведено комплексне тестування та налагодження реалізованої системи з метою перевірки її стабільності, коректності та відповідності технічним вимогам. Особлива увага приділялась перевірці взаємодії з API генерації зображень, логіки обробки команд у Discord, відображенню прогресу та обробці виняткових ситуацій.

Також забезпечено захист інформаційно-комп’ютерної системи шляхом використання HTTPS для безпечної передачі даних між ботом та Discord API, зберіганням токена авторизації у конфігураційному файлі та застосуванням механізмів авторизації користувачів Discord.

Виконання кваліфікаційної роботи також охоплювало підготовку документації з розгортання та визначення й опис системних вимог бота.

Завершальним етапом роботи стала розробка інсталяційних пакетів, що забезпечують просте розгортання Discord-бота на пристроях з операційними системами Windows та Linux.

Таким чином, у межах кваліфікаційної роботи було успішно виконано повний цикл розробки програмного забезпечення від початкового аналізу до готового до використання продукту, що відповідає поставленим цілям і сучасним вимогам до якості, гнучкості та зручності користування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DALL-E 2. OpenAI. URL: https://openai.com/index/dall-e-2/ (date of access: 03.03.2025).

2. Midjourney. URL: https://www.midjourney.com/home (date of access: 09.04.2025).

3. Announcing SDXL 1.0. Stability AI. URL: https://stability.ai/news/stable-diffusion-sdxl-1-announcement (date of access: 05.03.2025).

4. Wombo. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Wombo (date of access: 07.03.2025).

5. Що таке rest api. Основні принципи та практики застосування. FoxmindEd. URL: https://foxminded.ua/shcho-take-rest-api/ (дата звернення: 12.03.2025).

6. WebSocket. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/websocket (date of access: 17.06.2025).

7. Discord developer portal. API docs for bots and developers. URL: https://discord.com/developers/docs/reference (date of access: 19.04.2025).

8. Discord developer portal. API docs for bots and developers. URL: https://discord.com/developers/docs/topics/oauth2 (date of access: 25.03.2025).

9. C# – a modern, open-source programming language. URL: https://dotnet.microsoft.com/en-us/languages/csharp (date of access: 04.04.2025).

10. Troelsen A., Japikse P. Pro C# 10 with .NET 6: foundational principles and practices in programming. Apress L. P., 2022. 1640 p.

11. What is .NET? An open-source developer platform. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (date of access: 05.04.2025).

12. Visual Studio 2022 IDE – AI for coding debugging and testing. Visual Studio. URL: https://visualstudio.microsoft.com/vs/ (date of access: 09.04.2025).

13. Git. URL: https://git-scm.com/ (date of access: 15.04.2025).

14. GitHub – AUTOMATIC1111/stable-diffusion-webui: Stable Diffusion web UI. GitHub. URL: https://github.com/AUTOMATIC1111/stable-diffusion-webui (date of access: 18.04.2025).

15. Dependency injection – .NET. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection (date of access: 29.04.2025).