УДК 004.42

Авдосьєв І.А.
Бабюк Н.П

АРХІТЕКТУРА РОЗПОДІЛЕНОЇ БАГАТОБОТОВОЇ СИСТЕМИ ДЛЯ ПЛАТФОРМИ DISCORD З ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ НА БАЗІ АСИНХРОННОЇ ЕКОСИСТЕМИ PYTHON

Вінницький національний технічний університет

Анотація

Стаття присвячена проектуванню та розробці архітектури розподіленої системи Discord-ботів із підтримкою одночасного відтворення потокового аудіо, автоматизованої модерації та інтеграції великих мовних моделей. Здійснено детальний аналіз організації асинхронного виконання чотирьох незалежних ботів у межах єдиного процесу Python. Описано механізми ізоляції стану між агентами, управління Lavalink-вузлами та побудови системи штучного інтелекту з персонажами на основі Groq API. Запропоновано архітектурне рішення на основі патерну Cog та делегування команд через менеджер воркерів.

Ключові слова: Python, discord.py, asyncio, Wavelink, Lavalink, Groq API, великі мовні моделі, Discord-бот, потокове аудіо, багатоагентна система, Tortoise ORM.

Abstract

The article is devoted to the design and development of the architecture of a distributed Discord bot system supporting simultaneous audio streaming, automated moderation, and integration of large language models. A detailed analysis of the asynchronous execution of four independent bots within a single Python process is carried out. Mechanisms for state isolation between agents, Lavalink node management, and the construction of an artificial intelligence system with character-based interactions using the Groq API are described. An architectural solution based on the Cog pattern and command delegation through a worker manager is proposed.

Keywords: Python, discord.py, asyncio, Wavelink, Lavalink, Groq API, large language models, Discord bot, audio streaming, multi-agent system, Tortoise ORM.

Постановка проблеми

Платформа Discord набула широкого поширення як середовище для спільнот різного спрямування від ігрових до освітніх. Автоматизація функцій серверу за допомогою програмних агентів (ботів) є актуальним напрямом розробки. Однак сучасні вимоги до функціоналу Discord-ботів виходять за межі простих команд: необхідність одночасного відтворення музики в кількох голосових каналах, інтелектуального спілкування з користувачами та автоматизованої модерації формують комплексне архітектурне завдання. Проблема полягає у тому, що стандартна модель одного бота не дозволяє займати одночасно більше одного голосового каналу, що суттєво обмежує масштабованість рішення. Метою дослідження є розробка архітектурного рішення, яке забезпечить одночасну роботу декількох ботів в єдиному процесі Python із повною ізоляцією стану між агентами та централізованим управлінням.

Архітектура багатоботової системи

Розроблена система складається з чотирьох Discord-ботів, що функціонують у межах єдиного асинхронного процесу: головного бота Rainbow та трьох ботів-воркерів Iris, Aurora і Selene. Оркестрацію побудовано на базі стандартної бібліотеки asyncio мови Python: функція asyncio.gather() запускає всі чотири клієнти паралельно, дозволяючи їм незалежно обробляти Discord-події без блокування одне одного.

Центральним компонентом системи є WorkerManager менеджер, відповідальний за делегування команд відтворення вільному боту. Алгоритм пріоритизації надає перевагу головному боту Rainbow, далі послідовно перебирає воркерів Iris, Aurora, Selene. Усі slash-команди (/play, /skip, /pause тощо) зосереджені виключно в інтерфейсі Rainbow; воркери діють без власних команд, отримуючи завдання через внутрішні виклики Python-об'єктів.

Критичним аспектом архітектури є ізоляція кешу discord.py між ботами. Оскільки кожен Discord-клієнт підтримує власний внутрішній стан (_state), об'єкти Guild та Channel, отримані від одного бота, не можуть бути використані іншим без ризику підключення не того агента до голосового каналу. Для запобігання крос-контамінації у методі play_track кожного воркера явно виконується виклик worker.get_guild(guild_id) для отримання власного об'єкта серверу.

Музичний модуль та потокове аудіо

Відтворення аудіо побудовано на базі бібліотеки Wavelink асинхронної обгортки навколо Lavalink, автономного Java-сервера для обробки аудіопотоків. Для забезпечення відмовостійкості система підтримує кілька Lavalink-вузлів одночасно через механізм Wavelink Pool. При цьому об'єкт Pool є глобальним синглтоном, спільним для всіх ботів, що вимагає особливого підходу до ізоляції.

Кожен воркер отримує власний виділений Lavalink-вузол з унікальним ідентифікатором (WORKER_IRIS, WORKER_AURORA, WORKER_SELENE). Завдяки цьому механізм диспетчеризації подій node.client.dispatch() надсилає аудіо-події виключно боту-власнику вузла, унеможливлюючи їх перехоплення іншими агентами. Для пошуку треків Rainbow використовує функцію get_rainbow_node(), яка повертає лише non-WORKER_ вузли, виключаючи вузли воркерів з пулу пошуку.

Система підтримує два джерела медіаконтенту YouTube та Spotify через інтеграцію lavasrc-плагіну Lavalink. Для оптимізації швидкодії реалізовано LRU-кеш пошукових запитів ємністю 100 записів, спільний між Rainbow та воркерами. Автоматичне перемикання вузлів відбувається при виявленні деградації: 2 і більше помилок протягом 2 хвилин ініціюють failover на резервний вузол.

Система логування та моніторингу

Зважаючи на багатоботову природу системи, де чотири агенти генерують події одночасно, централізована система логування є критичним інструментом діагностики та налагодження. Реалізовано ієрархічну структуру журналювання на базі стандартного модуля logging мови Python із власним механізмом ротації сесій.

При кожному запуску системи автоматично створюється окрема директорія у форматі Logs/DD_MM_YYYY-HH-MM/, що відповідає часу старту. Система зберігає не більше трьох останніх сесій, автоматично видаляючи застарілі при перевищенні ліміту. Така організація дозволяє швидко зіставляти поведінку системи між різними запусками без ручного сортування файлів.

Кожен модуль системи веде власний лог-файл: музичний модуль генерує окремі файли для bot, cache, commands, components, worker_bot та worker_manager; модуль адміністрування — для commands, tasks, models; AI-модуль — для bot, context, memory, db. Глобальний файл errors.log агрегує повідомлення рівня ERROR і вище з усіх модулів одночасно, що спрощує виявлення критичних збоїв у продакшн-середовищі. Виведення у консоль доповнено ANSI-кольоровим форматуванням для швидкого візуального розрізнення рівнів DEBUG, INFO, WARNING та ERROR, тоді як файловий формат залишається plain-text для сумісності з будь-якими інструментами аналізу логів.

Модуль штучного інтелекту

Система містить модуль чат-бота, побудований на базі Groq API з використанням моделі llama-3.3-70b-versatile. Кожен із чотирьох ботів має власну унікальну особистість, визначену через системні промпти у форматі TOML: Rainbow тепла та дружна, Iris стримана та саркастична, Aurora енергійна у стилі Gen-Z, Selene мудра та спокійна. Вибір мови відповіді визначається автоматично на основі аналізу частки кириличних символів у повідомленні користувача.

Модуль контексту реалізує RAG-light підхід: перед формуванням запиту до LLM система збирає метадані каналу (назва, тема, категорія), короткострокову пам'ять (15 останніх повідомлень), соціальний контекст (карма та активні претензії користувача), а також інжектує документацію відповідних команд при виявленні ключових слів, пов'язаних із налаштуваннями або довідкою.

Система карми забезпечує динамічне ставлення бота до користувача в діапазоні від -100 до +100 балів. LLM додає спеціальний тег [K:±N] у кінці відповіді, який парсується та застосовується до бази даних. Негативні зміни карми, що перевищують поріг у -3 одиниці, автоматично фіксуються як

pretenzii (grudges) з описом інциденту, що впливає на подальший тон спілкування бота з цим користувачем.

Система модерації та допоміжні модулі

Модуль модерації побудовано на базі Tortoise ORM із SQLite як сховищем даних. Система реалізує триступеневу модель попереджень: кожне накладене попередження (/warn) автоматично призначає користувачеві відповідну роль згідно з конфігурацією сервера. Для тимчасових покарань реалізовано фоновий цикл на базі asyncio, що кожні 60 секунд перевіряє терміни дії та автоматично скасовує покарання. Додатково реалізовано систему амністії: якщо з моменту накладення попередження минуло більше 7 днів, воно автоматично знімається в рамках щоденного перегляду.

Розважальні модулі системи включають три підсистеми кастомізації нікнеймів: ColorNick (анімована зміна кольору ролі в режимах Rainbow, Gradient, Wave та Random), FontNick (Unicode-трансформація нікнейму у стилі Bold, Italic, Fraktur тощо) та NickPrefix (анімований префікс у стилі DVD-заставки, Spinner або LoadingBar). Управління конфігурацією реалізовано через централізований ConfigManager з LRU-кешуванням та підтримкою гарячого перезавантаження JSON-файлів без зупинки бота.

Висновки

Розроблена архітектура розподіленої багатоботової системи для платформи Discord демонструє практичну ефективність у вирішенні задач одночасного обслуговування кількох голосових каналів. Застосування asyncio.gather() та патерну WorkerManager дозволило об'єднати чотири незалежних агенти в єдиному процесі зі збереженням повної ізоляції стану, уникаючи накладних витрат на міжпроцесну комунікацію. Інтеграція Wavelink та Lavalink із механізмом виділених вузлів для кожного воркера вирішила фундаментальну проблему синглтон-природи Wavelink Pool, забезпечивши коректну маршрутизацію аудіо-подій виключно до бота-власника вузла.

Інтеграція Groq API з системою карми, унікальними особистостями ботів та RAG-light контекстом реалізує рівень соціальної взаємодії, що якісно відрізняє розроблену систему від стандартних чат-ботів. Персоналізоване ставлення бота до кожного користувача на основі накопиченої карми формує динамічне середовище спілкування. Централізована система логування із сесійною організацією та per-module журналами, у поєднанні з Tortoise ORM та aiosqlite, забезпечує надійний моніторинг і зберігання даних у виробничому середовищі.

Отримані результати підтверджують доцільність застосування асинхронної багатоагентної архітектури для розробки складних Discord-систем. Запропоновані рішення щодо ізоляції стану між агентами та делегування команд через централізований менеджер можуть бути адаптовані у ширшому контексті розробки багатоагентних систем реального часу на базі екосистеми Python.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Rapptz, "discord.py: A Python wrapper for the Discord API," GitHub. [Online]. Available: https://github.com/Rapptz/discord.py. [Accessed: May 24, 2026].
2. Rapptz, "discord.py Documentation: Components, Slash Commands and Event Reference," discord.py Docs. [Online]. Available: https://discordpy.readthedocs.io/en/stable/. [Accessed: May 24, 2026].
3. PythonDiscord, "Wavelink: A robust and powerful Lavalink wrapper for discord.py," GitHub. [Online]. Available: https://github.com/PythonDiscord/Wavelink. [Accessed: May 24, 2026].
4. Lavalink Devs, "Lavalink: A standalone audio sending node based on Lavaplayer," GitHub. [Online]. Available: https://github.com/lavalink-devs/Lavalink. [Accessed: May 24, 2026].
5. Groq Inc., "Groq API Documentation: Fast AI inference," Groq Console. [Online]. Available: https://console.groq.com/docs/. [Accessed: May 24, 2026].
6. Meta AI, "Introducing LLaMA 3.3: A New State-of-the-Art Language Model," Meta AI Blog, 2024. [Online]. Available: https://ai.meta.com/blog/. [Accessed: May 24, 2026].
7. Python Software Foundation, "asyncio: Asynchronous I/O," Python Documentation, 2025. [Online]. Available: https://docs.python.org/3/library/asyncio.html. [Accessed: May 24, 2026].
8. Python Software Foundation, "logging: Logging facility for Python," Python Documentation, 2025. [Online]. Available: https://docs.python.org/3/library/logging.html. [Accessed: May 24, 2026].

9. tortoise-orm, "Tortoise ORM: Easy async ORM for python, built with relations in mind," GitHub. [Online]. Available: https://github.com/tortoise/tortoise-orm. [Accessed: May 24, 2026].
10. omnilib, "aiosqlite: asyncio bridge to the standard sqlite3 module," GitHub. [Online]. Available: https://github.com/omnilib/aiosqlite. [Accessed: May 24, 2026].

Авдосьєв Ілля Андрійович – Студент групи 4ПІ-22Б, факультет інформаційних технологій та компютерної інженерії, Вінницький національний технічний університет, м. Вінниця, email: avdosevg@gmail.com

Бабюк Наталя Петрівна – к.т.н., доцент кафедри програмного забезпечення, Вінницький національний технічний університет. e-mail: babiuk@vntu.edu.ua

Avdosiev Illia Andriiovych – Student of group 4PI-22B, Faculty of Information Technology and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, email: avdosevg@gmail.com

Babiuk Natalia P. – PhD, Associate Professor of the Department of Software Engineering, Vinnytsia National Technical University. e-mail: babiuk@vntu.edu.ua