

Технічний звіт: Архітектура та аналіз Telegram-бота IUI (MLBB-BOSS)

1. Архітектура проєкту

Загальний огляд: Бот IUI (Independent Ukrainian eSports Initiative) побудований на основі **Python 3.11** з використанням асинхронного фреймворку **Aiogram 3** для взаємодії з Telegram API. Проєкт дотримується багатшарової архітектури, що розділяє логіку на окремі компоненти:

- **Бот-ядро та роутери:** Головний запуск реалізовано у файлі `mls_core.py`, де ініціалізується об'єкт бота (`Bot`) з токеном і задається режим парсингу HTML для повідомлень. Використовується **Dispatcher** та декілька **Router** для групування обробників за тематикою. Зокрема, є роутери: `admin_router`, `vision_router` (аналіз зображень), `registration_router` (реєстрація профілю), `profile_router`, `heroes_router`, `tournaments_router` тощо. Головний роутер включає всі підроутери через `dp.include_router(...)`, що забезпечує модульність та ізоляцію команд різних розділів. Такий підхід спрощує підтримку, дозволяючи вмикати/вимикати цілі блоки функціоналу.
- **Обробники команд та FSM:** Логіка реакції на повідомлення користувача реалізована у пакеті `handlers`. Кожен розділ функціоналу має свій модуль обробників (наприклад, `profile.py` для профілів, `tournaments.py` для турнірів, `gpt_handler.py` для AI-чату, `vision/router.py` для роботи зі скріншотами тощо). Обробники використовують можливості **Aiogram FSM** (Finite State Machine) для реалізації покрокових сценаріїв. В файлі `states/all_states.py` визначено кілька класів станів, що успадковують `StatesGroup` – наприклад, `ProfileStates`, `ScreenshotAnalysisStates` тощо – з переліком станів (State) для кожного сценарію. Це дозволяє боту “пам'ятати” контекст між повідомленнями (наприклад, очікування завантаження зображення або введення даних профілю) і реагувати відповідно.
- **База даних (ORM) та кеш:** Проєкт використовує **SQLAlchemy 2.0** для роботи з базою даних (у продакшні – PostgreSQL). В каталозі `models` описані ORM-моделі (наприклад, `User`, `Stats`, `ScreenshotAnalysis`, `Favorites` тощо), що відповідають таблицям. Підключення до БД здійснюється асинхронно через `AsyncEngine` (`postgresql+asyncpg`) із пулом підключень. Існує **middleware** `DbSessionMiddleware`, який на кожний апдейт створює нову сесію (`AsyncSession`) та додає її в контекст обробника – це реалізує своєрідний **Dependency Injection** для БД. Таким чином, обробники можуть отримати сесію із `handler.data["session"]` і виконувати транзакції. Окрім БД, для тимчасового зберігання даних використовується **Redis** (через бібліотеку `redis.asyncio`) як кеш. Зокрема, реалізований сервіс `RedisCache` для кешування відповідей AI: він формує ключ на основі ID користувача та запиту і зберігає результат з TTL, щоб повторні однакові запити не зверталися до API повторно. Redis також може використовуватися для зберігання FSM-станів або rate-limit обмежень (налаштовано через `aioredis` у стеку).

- **Сервіси та DI:** У проєкті виділено пакет `services` з класами, що інкапсулюють доступ до зовнішніх API та інфраструктури. Наприклад, `OpenAIService` / `GPTService` – для викликів OpenAI API, `HeroService` – для даних про героїв, `ImageAnalysisService` – для обробки зображень, `Scheduler` – для планування задач, `TokenTracker` – для моніторингу використання токенів OpenAI тощо. Такі сервіси створюються при старті бота (наприклад, через фабрику `AIModelFactory`) і можуть передаватися обробникам або викликатися через singletons. **Dependency Injection** реалізовано частково через контекст – окремі middleware додають потрібні об'єкти. Наприклад, middleware для БД згаданий вище, middleware `FileManager` може опрацьовувати документи/зображення та додавати шляхи, `TrackingMiddleware` – відстежує дії користувачів і може логувати їх через Sentry. Інші залежності (наприклад, інстанс Redis або OpenAI API ключ) зберігаються у налаштуваннях (`config.settings`) та у глобальних сервісах.

- **Зовнішні інтеграції:** Бот активно використовує сторонні сервіси для розширення функцій:

- **OpenAI API:** інтегровано модель **GPT-4** для AI-чатбота і навіть її мультимодальну можливість (Vision) для аналізу зображень. Проєкт має налаштування двох режимів – швидшого (можливо GPT-3.5) і потужнішого GPT-4, а також експериментальний доступ до GPT-4 Vision. Для текстових запитів користувачів бот відправляє повідомлення до OpenAI (з відповідним системним промптом про те, що він – асистент по MLBB) і повертає відповідь. Для **Vision**-функціоналу бот може надсилати зображення (скріншот) у модель GPT-4-Vision: у коді це реалізовано шляхом конвертації картинки в **base64** та включення її як data-URL в промпт моделі. Це дозволяє GPT-4 проаналізувати зображення (наприклад, результати матчу чи профіль гравця зі скріншоту) і надіслати текстовий аналіз користувачу. Такий рівень AI-інтеграції є передовим – бот фактично поєднує чат-бота і комп'ютерний зір на базі GPT-4.

- **Cloudinary:** для роботи з графікою використовується хмарне сховище Cloudinary. Зображення (наприклад, аватари, скріншоти профілів або ілюстрації гідів) можуть завантажуватися на Cloudinary через API, а потім бот оперує лише посиланнями. У коді є утиліта `cloudinary_upload.py` з функцією `upload_image_to_cloudinary`, яку викликають обробники скріншотів перед аналізом – це дозволяє зберігати історію та уникнути перевантаження Telegram файлами.

- **OCR та Computer Vision:** Окрім GPT-4 Vision, бот має класичний OCR-режим на базі бібліотеки **EasyOCR** і OpenCV. Є команда `/ocr`, яка переводить бота в стан очікування зображення, після чого бот завантажує надіслане фото та застосовує EasyOCR для витягу тексту. Це резервний інструмент для розпізнавання тексту зі скріншотів (може бути корисним для читання результатів ігор чи ID), якщо GPT-4 недоступний або для економії токенів. Також OpenCV використовується для попередньої обробки зображень (наприклад, покращення якості, пошук областей ROI) в модулі `utils/vision/image_tools.py` – це допомагає підвищити точність аналізу (зокрема, перед OCR або перед відправкою в GPT).

- **Sentry та моніторинг:** Проєкт підключений до **Sentry** для відстеження помилок у реальному часі. Будь-які виключення, особливо в асинхронних обробниках, логуються через Sentry SDK (ініціалізується в `mls_core.py`), що дозволяє розробникам отримувати сповіщення про збої. Логування виконано у форматі JSON для аудитів. Також передбачені **health-check** та **error-handling** механізми: є middleware, що ловить необроблені помилки (наприклад, при обробці скріншотів) і надсилає користувачу дружнє повідомлення замість падіння бота.

Взаємодія між модулями: Зважаючи на таку структуру, модулі взаємодіють через чітко визначені інтерфейси: - Обробники отримують дані від користувача (команди, повідомлення, callback-кнопки) через Aiogram і, за потреби, змінюють **стан FSM** користувача. Деякі обробники

викликають інші (наприклад, переходи між підменю) шляхом відправки нових повідомлень або редагування існуючих. - Якщо потрібні дані з БД або стороннього API, обробник звертається до відповідного **сервісу**. Наприклад, при команді перегляду профілю обробник викликає `User` модель через сесію БД (або через утиліту в `utils/db.py`, де є готові CRUD-функції). Для виклику AI обробник звертається до `AIDispatcher` або сервісу `GPTService`, передаючи йому запит і контекст; той вже вирішує, який моделі звернутися (GPT-4 чи швидший), чи є кеш для цього запиту, і виконує зовнішній HTTP-запит до OpenAI. - **Middleware** забезпечують “склеювання” шарів: при кожному апдейті послідовно виконуються middleware, які додають у контекст корисні залежності (сесію БД, підключення до Redis, дані про користувача тощо) або обробляють помилки. Таким чином досягається слабе зв'язування – основний код обробників не турбується про те, як саме отримати сесію чи як логувати дію, це відбувається прозоро.

Резюме архітектури: Проєкт вирізняється добре продуманою структурою: **ядро бота** відповідає за ініціалізацію та конфігурацію, **обробники** – за прикладну логіку взаємодії з користувачем, **становая машина** – за діалоги і сценарії, **моделі та сервіси** – за роботу з даними і API, **база даних та кеш** – за зберігання і швидкий доступ до інформації, **AI сервіси** – за розумні відповіді та аналіз, а **middleware** та утиліти – за інфраструктурні завдання (логування, валідація, завантаження файлів). В основі – сучасний технологічний стек: Aiogram, AsyncIO, SQLAlchemy, Redis, OpenAI GPT-4, Sentry, що відповідає вимогам продуктивності та масштабування для реального використання.

2. Якість коду та патерни

Відповідність принципам чистого коду: Код проєкту написаний з урахуванням стандартів PEP8 та загальних принципів чистого коду. Імена змінних і функцій інформативні, дотримується єдиний стиль оформлення. Про це свідчить наявність CI-інтеграції з авто-перевірками – у репозиторії налаштовано GitHub Actions, що запускає аналіз коду та тести для кожного Pull Request. Автор явно наголошує, що CI не пропустить зміни без тестів і що SOLID, типізація та коректне `async/await` – обов'язкові вимоги. Такий підхід гарантує підтримку високої якості: всі суттєві частини коду покриті **юніт-тестами** (понад 90% покриття) і статично типізовані (використовується `typing` і Pydantic для перевірки структур даних). В репозиторії є конфігурації автоформатування та літерів (наприклад, `flake8/black`), а також файли з автоматичного виправлення коду (скрипти “auto-fixer”), що вказує на регулярний рефакторинг. Підтримується чітка структура каталогів і модулів – жоден файл не “роздутий”, зазвичай відповідає тільки за одну сферу (Single Responsibility Principle). Наприклад, функції роботи з БД винесено в `utils/db.py` і `services/database.py`, логіку inline-клавіатур – у модулі `keyboards`, а різні сценарії – в окремі обробники. Це відповідає принципам **SRP** і полегшує розуміння та підтримку.

Використані патерни проєктування: В коді простежується кілька поширених патернів: - **FSM (Finite State Machine):** як згадано, активно використовується патерн кінцевого автомата для діалогів. Aiogram надає зручний інструмент FSM, і розробник правильно його застосував для складних сценаріїв (реєстрація, аналіз скріншотів, AI-чат). Стани та переходи задокументовано, назви станів зрозумілі (напр. `WAITING_FOR_IMAGE`, `CONFIRMING_ANALYSIS` тощо), що робить сценарії прозорими. - **Dependency Injection:** хоча явного фреймворку DI немає, патерн досягається через контекстне впровадження залежностей. Middleware-конструктори додають об'єкти (сесія БД, Redis-клієнт, тощо) у `event.data`, і обробники отримують їх через параметри функції. Також деякі сервіси створюються в глобальній області і доступні через імпорт (наприклад, `model_factory` або конфігурація `config.settings`). Це своєрідний Service Locator підхід. Така організація спрощує тести – можна підмінити, наприклад, Redis на фейковий, передавши інший RedisCache. - **Repository / DAO:** доступ до бази даних інкапсульований у окремі функції (`utils/`

db.py) та в **ORM-моделях**. Хоча класичних класів-репозиторіїв немає, CRUD-операції винесені у окремий шар, відокремлений від бізнес-логіки. Наприклад, замість виконання SQL-запиту прямо в обробнику, код викликає функцію `get_user_by_id(session, user_id)` із `utils/db.py`, що повертає об'єкт `User`. Це підвищує **DRY** – один і той самий код доступу до даних не дублюється всюди. - **DTO/VO**: для передачі даних між шарами використовуються або ORM-моделі, або прості словники. Бібліотека **Pydantic** згадується у залежностях, тож є ознаки використання моделей Pydantic для валідації зовнішніх даних (наприклад, при парсингу відповіді стороннього API або при обробці JSON-конфігурацій). Це можна розглядати як патерн Data Transfer Object – коли зовнішні дані (можливо, відповіді MLBB API) транслюються у внутрішні структури. Наприклад, у `parsers/mlbb_parser.py` ймовірно реалізовано розбір якихось даних про гру у зручний формат, який далі використовується в боті. - **Командний патерн і фабрики**: Aiogram сам реалізує командний патерн для обробників (кожна функція-обробник – це по суті обробка певної “команди”/події). У коді також є **фабрики** – зокрема, `KeyboardFactory` і кілька утиліт для генерації меню (`get_main_menu()`, `get_profile_menu()`). Вони працюють як патерн **Factory** для створення об'єктів `InlineKeyboardMarkup` залежно від контексту (наприклад, зібрати список кнопок героїв за певною роллю). - **Використання асинхронності та патерн “async/await all the way”**: Увесь код написаний максимально асинхронно. Будь-які операції вводу-виводу виконуються не блокуючи івент-луп: використано `aiohttp` для HTTP-запитів (до OpenAI, до Telegram файлів тощо), `asyncpg` через SQLAlchemy для БД, `redis.asyncio` для кешу. Розробник окремо зазначив, що **Event Loop ніколи не блокується – лише async код**. Важкі обчислення (наприклад, обробка зображень OpenCV або EasyOCR) по можливості винесені: наприклад, зображення зберігається у тимчасовий файл, після чого OCR-визнання проводиться. Хоча EasyOCR працює синхронно, обгорнення його виклику в `asyncio.to_thread` дозволило б повністю не блокувати потік – сподіваємось, це було враховано. В цілому асинхронна архітектура дозволяє боту швидко реагувати і обробляти кілька запитів паралельно (що критично для живих турнірів і активного чату).

Читабельність та підтримка: Код достатньо легко читати – він містить коментарі (багато з яких українською мовою, пояснюючи логіку для локальної команди розробки), структурований на розділи з розділовими коментарями (`/// =====`) для візуального розбиття. Функції не перевантажені зайвими аргументами; багато з них невеликі за розміром і виконують одну задачу. Також реалізовано **обробку винятків**: критичні місця загорнуті в блоки try/except з логуванням помилок (`logger.error(...)`), щоб відловити несподівані ситуації і не впасти. Для користувача передбачені дружні повідомлення при помилках (наприклад, якщо AI недоступний або зображення не розпізнано, бот відправить повідомлення з вибаченням, а не трасування стектрейсу). Це є ознакою зрілості проекту.

Повторне використання та розширюваність: Завдяки виділенню загального функціоналу в утилітні модулі, код не містить дублювання. Наприклад, форматування повідомлень з шаблонів зроблено через `MessageTemplateManager` – замість вручну складати текст для кожного випадку, використовуються готові шаблони і заміни, що спрощує локалізацію та оновлення текстів. Клавіатури будується універсально – є централізовані **карти кнопок** (`buttons.py`, `inline_keyboards_map.py`), що дозволяють переназначити дії кнопок без зміни логіки обробників. Додавання нового розділу меню або функції не потребує переписування ядра – досить створити новий Router з обробниками і підключити його. Така модульність свідчить про хорошу розширюваність.

Насамкінець, слід відзначити, що проект підтримує високу **якість коду** протягом розвитку: інтегровані інструменти аналізу (Sentry, логування), регулярне авто-оновлення залежностей та форматування (workflow “weekly code maintenance” на GitHub), наявний **набір тестів** (в папці

`tests/` і навіть мок-сервери для зовнішніх API). Усе це нетипово для любительського бота і піднімає проект до рівня продакшн-якості.

3. Функціональні можливості бота

Бот IUI надає користувачам кіберспортивної спільноти України широкий набір функцій, об'єднаних в одному Telegram-боті. Розглянемо ключовий функціонал та сценарії:



- **Профілі гравців:** Бот дозволяє кожному користувачу створити та керувати **своїм геймерським профілем**. При першому старті бот може запропонувати зареєструватися: ввести свій MLBB ID, вибрати основну роль (героя типу танк, маг, стрілець тощо) і додаткові ролі, вказати поточний ранг і найвищий досягнутий ранг. Цей сценарій реалізовано через діалог FSM – бот послідовно задає питання і зберігає відповіді. Після реєстрації профіль містить інформацію про гравця: **роль** у команді, улюблені герої, рейтинг (ранг), статистику ігор (якщо підключена) та інші дані. Особливістю є **AI-статистика** – бот може аналізувати ваші ігрові показники за допомогою AI. Наприклад, якщо користувач надасть скріншот зі своєю сторінкою статистики, бот (через модуль зору) прочитає дані (Win rate, KDA тощо) і збереже їх у профілі. Команда **«Мій профіль»** або відповідна кнопка в меню дозволяє переглянути свій профіль: бот надішле оформлений повідомленням список ваших даних. Передбачені розділи профілю: *Рейтинг* (поточний ранг, MMR), *Команда* (можливість зазначити команду або склад, якщо граєте в команді), *Досягнення* (нагороди, турнірні результати), *Налаштування* (наприклад, зміна ніку, мови інтерфейсу). Профіль – це своєрідна візитка гравця, яку бот використовує також і в інших функціях (наприклад, для турнірних заявок або підбору суперників).
- **Турніри:** Один з найцікавіших розділів – **кіберспортивні турніри**. Бот виступає платформою для локальних змагань. За його допомогою адміністрація може анонсувати новий турнір, а гравці – подати заявки на участь. Розділ **“Турніри”** у меню надає такі можливості:
 - **Анонси та реєстрація:** Користувачі бачать список доступних турнірів або найближчих подій. Біля кожного може бути кнопка “Долучитися” (якщо реєстрація відкрита) або “Деталі” для перегляду сітки/розкладу. При натисканні “Долучитися” бот може запитати підтвердження і зареєструвати користувача (додає його в список учасників). В цей момент бот бере дані з профілю (нікнейм, ID) і передає організаторам.
 - **Сітки та результати:** Після початку турніру бот може відправити зареєстрованим учасникам сітку турніру (наприклад, у вигляді зображення чи посилання). Якщо у бота є інтеграція зі стороннім турнірним ПО або API, він може оновлювати **live-статистику** матчів. В іншому разі, модератор вручну оновлює результати, а бот може розсилати повідомлення про проходження команд далі, про початок фіналу тощо.
 - **Нагадування та розсилка:** Бот виконує роль координатора – він може надсилати учасникам нагадування про матч (завдяки `scheduler` службі) або повідомляти всіх підписаних на турнір про те, хто переміг. Також доступна функція **розсилки** – адміністратор турніру через *панель BOSS* (спеціальне адмін-меню, ймовірно `boss_panel.py`) може відправити всім учасникам повідомлення (наприклад, правила або будь-яку інформацію).

Основний сценарій користувача: Звичайний гравець відкриває меню **“Навігація”**, обирає **“Турніри”**. Бот відправляє список турнірів з номерами або назвами. Користувач тисне на потрібний – отримує детальну інформацію (формат, призи, дати). Якщо турнір відкритий – кнопка **“Подати заявку”**. Натискає – бот може попросити підтвердити використання профільних даних,

після чого надсилає відповідь “ Вас зареєстровано на турнір!” і, можливо, додає в канал/групу турніру. Далі, коли настане час, бот пришле особисте повідомлення з розкладом або зіставленим опонентом.

- **AI-асистент (чат та поради):** Одна з **унікальних фіч** бота – вбудований **GPT-4 асистент**. Користувач може спілкуватися з ботом майже як з ChatGPT, але в контексті гри **Mobile Legends: Bang Bang**. Реалізовано це через окремий модуль обробників `gpt_handler.py`: існує команда (наприклад, `/ask` або `/ai`), яка переводить бота в режим AI-чатування. У цьому режимі всі подальші повідомлення користувача сприймаються як запит до AI. Приклад сценарію:
 - Користувач вводить команду `/ai` або натискає кнопку “AI-асистент” у меню. Бот відповідає: “Ви перейшли в режим AI-помічника. Поставте будь-яке запитання про MLBB.”
 - Користувач питає: “Який найкращий білд на героя Лейла зараз?”. Бот надсилає проміжне повідомлення наприклад “ Думаю...”, далі на бекенді формує запит до OpenAI. Завдяки спеціальному системному промпту, AI знає, що він **ML-GPT** – експерт з Mobile Legends, і відповідає українською. Відповідь може включати рекомендацію предметів, емблем та тактик для названого героя. Бот надсилає користувачу цю відповідь, форматуючи список предметів у вигляді маркованого списку або емодзі.
 - Користувач може задати уточнююче питання, і AI пам’ятає контекст (бот зберігає історію діалогу короткочасно, можливо, у Redis або в змінній FSM). Таким чином, реалізовано **багатокроковий діалог** з AI.

Можливості AI-асистента включають: поради по стратегії гри, оптимізації складу команди, генерацію веселих відповідей чи мемів на тему кіберспорту (бот спеціально тренований чи налаштований видавати іноді гумористичні репліки). Також AI може пояснювати патчноути, рекомендувати контр-піки проти певного героя, навчати новачків основам. Усі відповіді генеруються **GPT-4**, тому вони доволі розгорнуті і “людяні”. Завдяки кешуванню, повторні популярні запити (наприклад “meta герої цього місяця”) можуть приходити миттєво, якщо відповідь уже була збережена.

- **Білди та гайди:** Окремо від вільного AI-чату, бот має структуровані розділи з **гайдами** по грі та **рекомендованими білдами**. У меню “ Гайди” користувач може знайти, наприклад, “Meta аналітика”, “Найкращі практики” чи “Новачкам”. Цей розділ міг бути реалізований як статичний набір статей або інтерактивний перегляд контенту:
- **Meta-аналітика:** бот може показувати списки топ-героїв мети, найпопулярніші піки/бани у змагальних іграх. Можливо, інтегровані скрипти парсингу реальних даних (в репозиторії є `parsers/mlbb_parser.py`, що натякає на збір статистики). Інформація подається у вигляді повідомлення з текстом, таблицями або зображеннями (наприклад, графік Win Rate героїв).
- **Білди героїв:** існує розділ “ Білди”, де гравець може обрати конкретного героя і отримати рекомендований набір предметів та емблем для нього. Дані білди могли бути підготовлені вручну (наприклад, від локальних кіберспортсменів) або генеруватися AI на основі актуальної інформації. Не виключено, що бот робить API-запит до стороннього сервісу з білдами або використовує ті ж можливості GPT (але з іншим промптом) для формування білду. У будь-якому разі, для користувача це виглядає як структурований міні-гайд: бот надсилає назву героя, іконки або список предметів (з емодзі меча, щита і т.п.), рекомендує порядок збору. Також можуть бути поради, проти яких героїв цей білд ефективний.
- **Гайди/аналітика:** можуть охоплювати такі теми, як “Як піднятися з Epic до Mythic”, “Топ-5 помилок при грі танком”, “Огляд останнього оновлення”. Ці тексти або написані командою,

або сформовані AI і відредаговані. Бот може видавати їх за командою або за розкладом (наприклад, щотижневий гайдз – “Герой тижня”).

- **Навігація та інтерфейс користувача:** Бот надає **інтуїтивний UX** через поєднання команд і інтерактивних кнопок. Після старту (команда `/start`), новому користувачу може бути показано *вітальне повідомлення з каруселлю* – серія повідомлень із картинками і текстом, між якими можна гортати «Далі/Назад». Це свого роду **інтерактивний туторіал**, реалізований в `intro_carousel_handler.py`: бот завантажує зображення (ймовірно, з Cloudinary) із описом можливостей бота (наприклад, “ IUI – твій провідник у кіберспорті!”), користувач тисне «Далі» і переглядає наступні слайди. Таким чином новачок одразу бачить, що бот вміє (профілі, турніри, AI тощо) – це покращує залученість.

Основна навігація побудована на **Inline-кнопках**. При виклику команди або натисканні головного меню, бот надсилає повідомлення з клавіатурою. Наприклад: - Головне меню може містити кнопки: “ *Навігація*”, “ *Мій профіль*”, “ *Зворотній зв'язок*”, “*? Допомога*”. Це зроблено, щоб відділити персональні дії від загальних. Якщо натиснути “**Навігація**”, відкриється підменю зі списком розділів: *Герої, Команди, Турніри, Білди, Гайди, Розваги, Бусти, Аукціон, Вивчай, Тренуй, Прогнози, Гравиці*. (Деякі з них, як видно, ще знаходяться в розробці або зарезервовані на майбутнє). Кожен пункт меню – це фактично перехід в інший розділ. - При виборі розділу бот надсилає *опис розділу* і новий набір кнопок. Опис береться з словника `SECTION_DESCRIPTIONS` – наприклад, для “Герої” буде текст “Розділ героїв: статистика, здібності та ролі.”, щоб користувач розумів, що там можна знайти. Кнопки в цьому розділі можуть бути, скажімо, список класів героїв (Асасин, Маг, Танк, Стрілець, Підтримка). Це реалізовано через JSON-файли (наприклад, `hero/assassin.json` містить перелік героїв-асасинів). Коли користувач обирає клас, бот надсилає пагіновану **карусель героїв** – наприклад, показує 5 героїв цього класу з кнопками “◀ / ▶” для гортання та “і Детальніше” біля кожного. Цей механізм описано у `hero_navigation.py` та `navigation_carousel.py`. Так користувач може проглядати довгі списки прямо в чаті, не виводячи все полотном. - **Пагінація та inline-карусель:** Бот використовує callback-кнопки з даними виду `page:2` або `hero:Layla` для перемикання сторінок і вибору елементів. Обробник `global_inline.py` відслідковує натискання універсальних кнопок “Назад” або “В меню”, щоб забезпечити повернення на попередній рівень меню без збоїв. Це забезпечує **зручну навігацію**, схожу на міні-додаток всередині Telegram. - **Стан користувача:** Коли користувач перебуває в певному розділі або режимі (наприклад, режим AI-чату або режим очікування скріншоту для аналізу), бот може міняти підказку в полі вводу (через `Bot.set_chat_menu` Aiogram функції) чи надсилати службові повідомлення. FSM дозволяє розмежувати, куди потрапить наступне повідомлення користувача: якщо стан `ScreenshotAnalysisStates.WAITING_FOR_SCREENSHOT`, то фото піде в обробник аналізу, а не якесь інше.

- **Аналіз скріншотів та Vision-функції:** Це варто виокремити як особливу функцію. Користувач може надіслати боту скріншот зі статистикою після матчу або зі своїм профілем у грі. Бот здатен **розпізнати тип скріншоту** (матч, профіль, екран з героєм тощо) та **проаналізувати** його. Ця можливість реалізована у модулі `handlers/vision/router.py` та допоміжних класах:
- Користувач командою `/analyze` (або кнопкою “Аналіз скріншоту”) запускає процес. Бот відповідає: “Надішліть мені скріншот з гри, і я спробую його проаналізувати!”, після чого встановлює FSM-стан `WAITING_FOR_SCREENSHOT`.
- Користувач надсилає зображення (можна просто переслати з галереї). Бот ловить це через обробник `@router.message(ScreenshotAnalysisStates.WAITING_FOR_SCREENSHOT, F.photo)` і починає обробку. Спершу зображення оптимізується (зменшується розмір, обрізається зайве) – функція `optimize_image`. Далі викликається

- `get_screenshot_type` з модуля `vision/classifier.py`, що намагається визначити тип екрану. Для покращення точності може використовуватися **GPT-4 Vision** у швидкому режимі (попередньо натреновані підказки з описом візуальних елементів різних типів скрінів). Якщо AI не доступний, є fallback: наприклад, перевірка наявності певних шаблонів тексту через OCR (наприклад, слово "VICTORY" для екрану кінця матчу).
- Коли тип визначено, відповідний аналізер із `vision/analyzers.py` обробляє зображення. Наприклад, для матчу (`MATCH`) – витягує таблицю рахунків команд, визначає MVP, оцінює показники гравця; для профілю (`PROFILE`) – читає загальний рівень, кількість героїв, рейтинг; для *статистики героя* (`STATS`) – можливо, будує висновки про майстерність цього героя у гравця. Ці аналізатори можуть комбінувати OCR (через EasyOCR) та виклики OpenAI. Зокрема, є універсальний аналізатор `universal_vision_analyzer.py`, що формує для GPT-4-Vision запит: "Опиши, що на ньому зображено і дай пораду гравцю". Тобто AI сам вирішує, що важливого на скріншоті, і дає зворотний зв'язок (наприклад: "Бачу, що в тебе 5 перемог підряд – відмінний результат! Можливо, варто спробувати підвищити складність і грати рейтингові матчі, щоб швидше досягти міфіка.").
 - Після аналізу бот надсилає користувачу результат. Результат може бути досить деталізований: бот, наприклад, надсилає декілька повідомлень або одне відформатоване, де зазначає ключові цифри (KDA, золото, нанесений урон) і вставляє поради ("Спробуй грати обережніше на початку матчу, щоб менше вмирати" або "Твій відсоток перемог на цьому герої 60% – чудово, продовжуй в тому ж дусі!"). Такі підказки генеруються AI і роблять взаємодію дуже корисною, бо гравець отримує аналіз ніби від тренера.
 - **Історія аналізів:** Бот може зберігати історію проаналізованих скріншотів для кожного користувача (модель `ScreenshotAnalysis` в БД). Відповідно, в меню "Аналіз скріншотів" може бути опція "Переглянути минулі аналізи" – бот відправить список або дозволить гортати попередні результати (кнопки "◀ Попередній / Наступний ▶" з короткою інформацією і можливістю відкрити детально). Це реалізовано станом `BROWSING_HISTORY` і відповідними обробниками у `vision/router.py`. Така функція корисна, щоб гравець міг оцінити свій прогрес з часом.

У підсумку, функціонал бота покриває майже всі потреби кіберспортивної спільноти: **особисті профілі, пошук ігор/гравців, організація турнірів, отримання навчальних матеріалів і порад, живе спілкування та розваги через AI**. І все це – в межах одного Telegram-бота з дружнім інтерфейсом, українською мовою та цілодобовою доступністю. Це підтверджує статус проекту як *"першого Telegram-бота для кіберспортивної спільноти України"*, що поєднує безліч сервісів в одному.

(Примітка: Деякі з перелічених підрозділів, такі як "Команди", "Бусти", "Аукціон", "Міні-ігри" та "Мем-режим", судячи з позначок у меню, ще знаходяться в розробці або заплановані на майбутнє. У версії 1.0, яка зараз live, реалізовані профілі, AI-чат, турніри, гайди та базова навігація. В майбутніх версіях (1.1, 2.0) очікується поява розважальних ігор, генерації мемів та вебхуків згідно з Roadmap.)

4. Порівняння та унікальність проекту

Проект IUI (MLBB-BOSS) вигідно відрізняється від типових Telegram-ботів і навіть від існуючих публічних платформ по Mobile Legends своєю концепцією та реалізацією:

- **Багатофункціональність vs. спеціалізованість:** Більшість Telegram-ботів створюються під одну задачу (наприклад, **стат-бот** показує рейтинг гравця або **новинний бот** шле оголошення). IUI же є **єдиним центром** для геймера: він об'єднує функції, які раніше вимагали б окремих сервісів. Наприклад, замість того щоб перевіряти статистику на

одному сайті, читати гайди на форумі, а спілкуватися в Discord, користувач може зробити все через бота. Це більше схоже на мобільний додаток, але реалізовано в зручному форматі месенджера. Подібних ботів для MLBB немає – є офіційні або фан-сторінки, але не інтерактивні боти з таким охопленням.

- **Інтеграція штучного інтелекту:** Абсолютна родзинка – використання **GPT-4**. Зараз багато де говорять про AI, але практичних ботів, що вміло інтегрують **розмовний штучний інтелект у тематику гри**, практично нема. IUI став одним із перших, хто впровадив GPT-4 (включно з баченням) прямо в Telegram для масового використання. Це означає, що користувачі отримують рівень підтримки, який раніше був недоступний: живі поради під час гри, аналіз їх помилок, навіть створення жартів чи мемів на льоту. Інші MLBB-платформи (наприклад, офіційний сайт статистики) можуть дати цифри, але не пояснять їх і не поспілюються з вами про гру. Тут же бот виступає в ролі *другого тренера* або *напарника для обговорення стратегій*. Така інтеграція AI робить проект унікальним. Розробники навіть заявляють слоган, що це **перший український eSports-бот з AI-підтримкою** – і поки що це правда.
- **Глибина кастомної логіки:** На відміну від простих ботів, де вся логіка – кілька if/else, тут реалізовано складні сценарії і механіки. Наприклад, **пошагова реєстрація** з перевіркою валідності введених даних; **обробка зображень** (і не тривіальна, а з класифікацією типу скріншоту, OCR і комп'ютерним зором); **система меню з пагінацією**, яка нагадує міні-інтерфейс користувача всередині чату. Особливо варто згадати **тестування**: проект містить більше 50 тестів, що моделюють різні ситуації (HTTP-виклики, відповіді AI, обробка callback-ів). Це гарантує, що усі частини логіки працюють узгоджено і не “ламаються” при розширенні. Багато публічних ботів не мають таких перевірок, через що часто падають або ведуть себе непередбачувано при збільшенні навантаження. IUI же спроектований з запасом: **кешування** запобігає надмірному зверненню до API, **rate limit** (обмеження частоти) захищає від спаму, **retry-логіка** автоматично повторює запити у разі тимчасового збою. Все це – ознаки production-ready системи, чим не можуть похвалитися любительські рішення.
- **Порівняння з MLBB-платформами:** Існують окремі сайти/додатки для відстеження статистики MLBB (наприклад, офіційний MLBB Leaderboards або фан-сайти з гайдами). Однак вони, як правило, **англомовні і розраховані на глобальну аудиторію**. IUI створено **“для своїх”** – з урахуванням українських реалій, мови та спільноти. Це проявляється навіть у дрібницях: всі тексти – українською, жарти – зрозумілі нашому гравцю, турніри – локальні, зручний час і формат. Таким чином, бот закриває нішу, яка донедавна була порожня. Замість того, щоб адаптуватися до англомовних ресурсів, українські гравці отримали рідний продукт, який їх об'єднує. Та ще й **мобільність Telegram** – бот завжди під рукою, повідомлення приходять миттєво, не треба відкривати окремі програми. Це величезна перевага, особливо для молодшої аудиторії, що і так проводить час у месенджерах.
- **Розширення спільноти та унікальний UX:** Завдяки інтерактивності бот не лише надає інформацію, а й створює **спільноту**. Наприклад, наявність *“чат-кімнати”* (групи Telegram, згаданої як t.me/mlbb_ukraine) і інтеграція бота з нею означає, що користувачі можуть разом обговорювати, а бот виступає як модератор або джерело даних у чаті. Це більше, ніж просто інструмент – це платформа для комунікації геймерів. У публічних MLBB-платформах такої соц. складової немає, або вона розрізнена (форуми, дискорд). Тут же все згуртовано навколо бота.

- **Технологічна інноваційність:** Проект впроваджує найсучасніші технології, які рідко побачиш в Telegram-ботах: асинхронний Python, SQLAlchemy 2.0 з типізацією, GPT-4 (ще й з модальністю Vision), Cloudinary CDN для медіа, Sentry-моніторинг, CI/CD з автоматичним деплоєм на Heroku. Це фактично переносить рівень якості, притаманний стартапам або комерційним продуктам, у волонтерський (або фанатський) проект. Такий набір технологій забезпечує надійність та масштабованість: бот може витримати велику кількість користувачів, його легко оновлювати і розширювати новими фічами, мінімізовано ризики витоків пам'яті чи падіння під навантаженням. Унікальність ще й в тому, що розробка повністю **open-source** (репозиторій доступний на GitHub), отже спільнота може долучатися, пропонувати зміни. Це нестандартно для кіберспорт-інструментів, які зазвичай пропріетарні.

Підсумовуючи, Telegram-бот IUI (MLBB-BOSS) є новаторським рішенням на перетині ігрової аналітики, соціального хабу та AI-технологій. Він не просто копіює існуючі сервіси, а пропонує якісно новий досвід: **персонального помічника для гравця Mobile Legends**, доступного в будь-який момент. Саме поєднання різномірних функцій (турніри + профілі + чатбот) та глибока інтеграція з AI роблять його особливим і на даний момент неповторним. Команда заклала міцний фундамент, тож у майбутньому бот може стати ще крутішим – додати автогенерацію мемів, внутрішньоігрові активності (квести, конкурси), підтримку інших ігор тощо. Але вже зараз IUI підтверджує своє гасло: *“перший український eSports-бот з AI, мем-режимом і підтримкою українських геймерів”* – проект, який запам'ятається надовго і задає нову планку для спільноти.
