Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ АДМІНІСТРУВАННЯ ІГРОВОГО КОНТЕНТУ З ІНТЕГРАЦІЄЮ В TELEGRAM (СЕРВЕРНА ЧАСТИНА)»

Виконав: студент IV курсу, групи 1ІСТ-21б спеціальності 126 – Інформаційні системи та технології
(шифр і назва спеціальності)
Гончар С. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент
Коцюбинський В. Ю.
(прізвище та ініціали)
«17» 06 2025 р.

Рецензент: д.т.н. проф. каф. КСУ Юхимчук М.С.
(прізвище та ініціали)
«17» 06 2025 р.

Допущено до захисту
завідувач кафедри АІІТ
«18» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 126 – Інформаційні системи та технології
Освітньо-професійна програма Інтелектуальні інформаційні системи

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
д.т.н., проф. Бісікало О. В.
«18» 06 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гончару Сергію Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка клієнт-серверної системи адміністрування ігрового контенту з інтеграцією в Telegram (серверна частина)»
керівник роботи Коцюбинський Володимир Юрійович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «20» березня 2025 року №97
2. Термін подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи:
- наявність адміністративного функціоналу для керування користувачами та сервером;
- можливість реєструвати нових та редагувати наявних адміністраторів системи в Telegram;
- можливість відслідковувати серверні події;
- можливість відповідати на звернення користувачів до служби підтримки.
4. Зміст текстової частини (перелік питань, які потрібно розробити) провести огляд існуючих систем управління контентом та користувачами, дослідити методи реалізації систем адміністрування ігрового контенту, розробити систему адміністрування ігрового контенту з інтеграцією в Telegram з використанням сучасних методів
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) Діаграма розгортання системи адміністрування через Telegram-бот, діаграма основних класів системи, діаграма активності процесу додавання нового користувача системи, діаграма активності обробки текстової команди Telegram-ботом

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Коцюбинський В. Ю., доц. каф. АІІТ		

7. Дата видачі завдання 13.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Термін виконання		Примітка
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БКР	03.10.2025	09.10.2025	вик
2	Аналіз предметної області та опрацювання літературних джерел.	12.03.2025	29.03.2025	вик
3	Постановка задач для вирішення в БКР	01.04.2025	26.04.2025	вик
4	Вирішення поставлених задач	29.04.2025	10.05.2025	вик
5	Підтвердження достовірності отриманих результатів	13.05.2025	24.05.2025	вик
7	Оформлення пояснювальної записки та графічної частини	27.05.2025	31.05.2025	вик
8	Перевірка ПЗ на відповідність вимогам	03.06.2025	07.06.2025	вик
9	Попередній захист, доопрацювання, та рецензування БКР	10.06.2025	15.06.2025	вик
10	Захист БКР ЕК	19.06.2025	19.06.2025	вик

Студент _____ Гончар С. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коцюбинський В. Ю.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 82 сторінок формату А4, в тому числі і додатків, на яких є 14 рисунків, 2 таблиці, список використаних джерел містить 24 найменування.

Дана бакалаврська робота присвячена розробці систем адміністрування ігрового контенту та управління користувачами та їх інтеграції в Telegram на мові програмування C# з використанням бібліотеки Telegram.Bot SDK та базою даних MariaDB.

У ході роботи було проведено аналіз предметної області, досліджено сучасні підходи до побудови клієнт-серверної архітектури, розглянуто аналоги CRM- та CMS-систем, а також інтерфейси для віддаленого адміністрування. Отримані результати дозволили сформувати технічні вимоги та спроєктувати систему, що складається з ряду модулів, серед яких: модуль управління користувачами, модерації репортів, логування дій та управління сервером. Усі компоненти взаємодіють через Telegram-бот в режимі реального часу.

Ключові слова: адміністрування, ігровий сервер, Telegram-бот, C#, Telegram.Bot SDK, MariaDB, клієнт-серверна архітектура, CMS, CRM, репорти, модерація, логування, система керування користувачами.

ANNOTATION

The bachelor's thesis consists of 82 pages of A4 format, including appendices, which contain 14 figures, 2 tables, the list of used sources contains 24 names.

This bachelor's thesis are devoted to the development of game content administration and user management systems and their integration into Telegram in the C# programming language using the Telegram.Bot SDK library and the MariaDB database.

During the work, an analysis of the subject area was conducted, modern approaches to building a client-server architecture were studied, analogues of CRM and CMS systems were considered, as well as interfaces for remote administration. The results obtained allowed us to form technical requirements and design a system consisting of a number of modules, including: a user management module, report moderation, action logging and server management. All components interact through the Telegram bot in real time.

Keywords: administration, game server, Telegram bot, C#, Telegram.Bot SDK, MariaDB, client-server architecture, CMS, CRM, reports, moderation, logging, user management system.

ЗМІСТ

ВСТУП

Актуальність. Сучасна ігрова індустрія, як і решта напрямів сфери інформаційних технологій, демонструє стрімкий розвиток. Також це стосується мультиплеєрних рольових ігор (Role Play, RP), серед яких особливе місце посідають проєкти на платформі Rage MP. Ця модифікація дозволяє створювати та підтримувати багатокористувацькі сервери для гри GTA V, з повноцінною реалізацією рольової механіки, економіки, фракцій і взаємодії між гравцями.

Зі зростанням складності ігрових проєктів, виникає необхідність у зручних та ефективних інструментах адміністрування. Традиційні панелі керування часто є громіздкими, вимагають додаткового програмного забезпечення або мають обмежену функціональність. У зв'язку з цим все більшої популярності набуває інтеграція адміністративних інструментів у месенджери, зокрема Telegram, що дозволяє забезпечити швидкий доступ до ключових функцій управління сервером безпосередньо з мобільного пристрою.

Метою роботи є покращення процесу взаємодії користувачів системи шляхом використання Telegram-боту і забезпечення адміністраторам системи доступу до інструментів керування сервером Rage MP.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі підходи до побудови систем адміністрування контенту та користувачів, зокрема з використанням месенджерів;
- визначити вимоги до функціональності та архітектури системи у контексті адміністрування ігрового сервера;
- обґрунтувати вибір архітектурного підходу, засобів реалізації та структур бази даних;
- реалізувати Telegram-бота з підтримкою обробки адміністративних команд;
- забезпечити авторизацію та захист доступу до функцій бота;
- реалізувати логування дій, обробку звернень (репортів) та інші адміністративні функції;

- провести тестування системи на відповідність функціональним вимогам, стабільність роботи та коректність обробки команд;
- проаналізувати можливості масштабування та подальшого розвитку системи.

Об'єктом дослідження виступає процес адміністрування рольового ігрового контенту на сервері Rage MP, а **предметом роботи** — методи реалізації інтерактивної клієнт-серверної взаємодії через Telegram API.

Практичне значення від роботи полягає у тому, що розроблене рішення дозволяє:

1. розширити функціональні можливості адміністрування без розробки окремих веб-інтерфейсів;
2. знизити навантаження на адміністративний персонал;
3. забезпечити гнучкий та безпечний доступ до ключових функцій керування сервером.

Методи дослідження – аналіз і систематизація інформації для вивчення предметної області та сучасних підходів до побудови клієнт-серверних систем; моделювання для формалізації логіки взаємодії між основними компонентами; проєктування програмного забезпечення із застосуванням загальноприйнятих принципів побудови архітектури; порівняльний аналіз для вибору оптимальних технологій та інструментів, що відповідають вимогам функціональності, надійності та масштабованості.

Результати роботи впроваджено в проєкт «Unreal RP», що функціонує на платформі Rage MP. Реалізована система використовується для дистанційного адміністрування ігрового сервера через Telegram-бота, зокрема для керування гравцями та сервером, моніторингу аномальної активності (збоїв, використання гравцями забороненного програмного забезпечення, внутрішніх помилок), а також доступу до внутрішньоігрової системи репортів. Крім того, розроблене рішення може бути адаптоване для використання в інших аналогічних проєктах, де серверна логіка реалізована мовою C#.

1 ОГЛЯД ІСНУЮЧИХ КЛІЄНТ-СЕРВЕРНИХ СИСТЕМ АДМІНІСТРУВАННЯ РЕСУРСАМИ ТА КОНТЕНТОМ

1.1 Дослідження актуальності систем адміністрування, що базуються на технічних вимогах клієнта

У сучасних умовах розвитку цифрових технологій більшість інформаційних систем — від електронної комерції до корпоративних платформ — потребують ефективних інструментів адміністрування. Вони дозволяють здійснювати контроль над ресурсами, користувачами, даними та контентом у режимі реального часу. Стрімке розширення функціональних можливостей таких систем обумовлює підвищення вимог до гнучкості, доступності, інтеграційних можливостей та зручності у користуванні. Як зазначає Fowler M. у своїй праці "Patterns of Enterprise Application Architecture", правильно спроєктована адміністративна система є ключовим компонентом стабільного функціонування будь-якого сервісу, особливо в умовах високого навантаження та мінливої структури даних [1].

Традиційно системи адміністрування застосовуються у веб-середовищах, таких як CMS для управління контентом (наприклад, Drupal, WordPress), або CRM-системи для підтримки бізнес-процесів. Проте останніми роками спостерігається зростаюча потреба у створенні індивідуальних адміністративних систем для ігрової індустрії — зокрема, в сегменті мультиплеєрних рольових ігор. Такі ігри часто функціонують на модифікованих ігрових платформах, як-от Rage Multiplayer (Rage MP), що дозволяє створювати багатокористувацькі сервери для гри Grand Theft Auto V із глибоким рівнем кастомізації внутрішньоігрової логіки.

Згідно з даними спільноти RageMP [2], типовим викликом для адміністраторів таких серверів є відсутність адаптованих систем керування, які враховували б специфіку геймплею та потребу у швидкому реагуванні на внутрішньоігрові події. Зазвичай адміністративний функціонал реалізовується у вигляді

внутрішньоігрових панелей керування, що вимагає перебування адміністратора у грі, рідше можна зустріти рішення, створені у вигляді веб-застосунків, однак вони теж не дають достатньої мобільності користуватися такими сайтами з мобільних пристроїв щонайменше незручно, інколи майже неможливо, а створення окремого мобільного додатку для кожної з популярних операційних систем потребує додаткових ресурсів. Водночас зростає популярність Telegram як універсального засобу взаємодії — його API дозволяє реалізовувати ботів для миттєвого адміністрування у зручному чат-інтерфейсі [3].

Проблеми, які виникають у класичних адміністративних системах у контексті ігрових серверів, включають:

- відсутність мобільного доступу до ключових функцій;
- складність налаштувань без спеціалізованих знань;
- несумісність з внутрішніми структурами ігрового світу (гравці, економіка, фракції);
- відсутність захищеної інтеграції з популярними каналами комунікації, такими як Telegram або Discord.

У зв’язку з цим все більше поширення отримують індивідуальні рішення, що реалізують клієнт-серверну архітектуру з використанням чат-ботів як інтерфейсу. Системи адміністрування нового покоління повинні відповідати таким технічним критеріям:

1. Гнучка архітектура, що підтримує модульне розширення та налаштування під сценарії конкретного сервера (наприклад, окремі команди для взаємодії з транспортом, фракціями тощо).
2. Інтеграція з Telegram Bot API для спрощення доступу до адміністративних функцій із мобільних пристроїв та зменшення часу реакції.
3. Інтерфейс, орієнтований на простоту, який не вимагає глибоких технічних знань з боку адміністратора.

Варто зауважити, що подібний підхід використовується не лише в ігровій сфері. Системи на базі Telegram-ботів активно впроваджуються й у бізнесі, логістиці та навіть державному секторі. Як зазначає Iancu M. у книзі Hands-On

Telegram Bot Development, чат-боти — це ефективний спосіб автоматизованої взаємодії з системою через знайомий інтерфейс, що дозволяє скоротити навантаження на адміністраторів і мінімізувати час навчання [4].

Таким чином, розвиток та ускладнення сучасних рольових ігор створює попит на розробку адаптованих систем адміністрування, які враховують як специфіку ігрового середовища, так і потребу у швидкій, безпечній та мобільній взаємодії. Telegram-інтегровані системи відповідають цим вимогам і дозволяють суттєво підвищити ефективність адміністрування серверів Rage MP, що робить їх актуальними та перспективними об'єктами дослідження.

1.2 Аналіз існуючих систем управління ресурсами та контентом

Управління ресурсами та контентом є ключовим елементом будь-якої сучасної інформаційної системи, зокрема у сфері адміністрування ігрових серверів. Ефективна система управління дозволяє підтримувати актуальність даних, здійснювати контроль над доступом, модерувати вміст та забезпечувати злагоджену роботу всієї інфраструктури.

Системи такого типу можуть мати різні форми реалізації — від класичних веб-панелей до інтеграцій із зовнішніми платформами, такими як месенджери або хмарні сервіси. При цьому особливої ваги набувають рішення, адаптовані до специфіки конкретного проєкту або сервера, що враховують індивідуальні вимоги користувачів та адміністрації.

Серед базових рішень такого типу варто виділити CRM (Customer Relationship Management) — системи управління взаємодією з користувачами, які дозволяють вести облік, відслідковувати активність, обробляти звернення та забезпечувати підтримку. У контексті ігрових систем це може охоплювати облік гравців, фіксацію порушень, роботу з репортами тощо [5].

Інший важливий клас — CMS (Content Management System), системи керування контентом. Вони забезпечують створення, редагування й публікацію інформаційного наповнення, організацію структури даних, керування правами

доступу, а також можливість інтеграції з базами даних, API та іншими службами [6].

У даному підрозділі буде розглянуто існуючі підходи до побудови систем управління ресурсами та контентом, їх функціональні можливості, переваги й недоліки. Це дозволить виявити ключові обмеження поточних рішень і обґрунтувати необхідність розробки власної клієнт-серверної системи з інтеграцією в Telegram для потреб адміністрування рольового ігрового контенту.

1.2.1 Аналіз систем управління контентом на прикладі Drupal

Drupal — це одна з найпотужніших та найбільш гнучких систем керування контентом (CMS) з відкритим кодом [7]. Вона орієнтована на розробку складних, масштабованих веб-додатків, у тому числі сайтів державних установ, корпоративних порталів, освітніх платформ та спільнот із великою кількістю користувачів.

Drupal розповсюджується безкоштовно під ліцензією GNU GPL і має активну спільноту розробників. Вона дозволяє створювати повністю кастомізовані типи контенту, визначати складну структуру даних, управляти правами доступу та формувати API для інтеграцій з іншими системами.

Дане рішення має модульну архітектуру, де основна функціональність Drupal розширюється за допомогою додаткових модулів, що підключаються до базового ядра. Система дозволяє створювати різні типи контенту (Content Types) з власними наборами полів, такими як текст, дата, медіа чи списки. Для категоризації контенту використовується механізм Taxonomy, що ґрунтується на словниках і термінах. Засобом виведення даних є інструмент Views, який дозволяє формувати унікальні подання контенту у вигляді списків, таблиць чи фільтрів без необхідності програмування. Відокремлення логіки від візуального представлення забезпечується завдяки Theme layer, який використовує Twig-шаблони для формування інтерфейсу (рис 1.1).

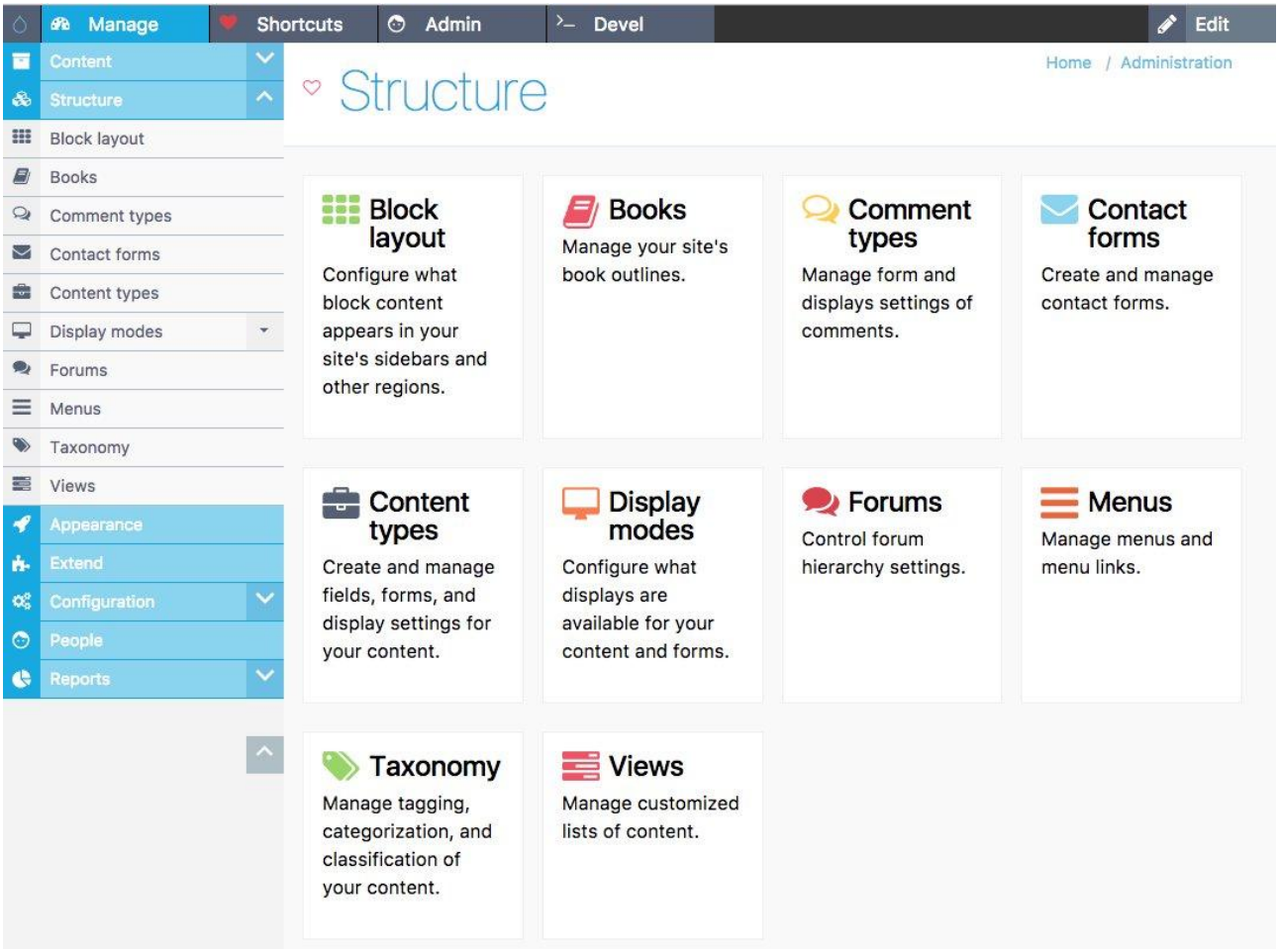

Рисунок 1.1 – Інтерфейс адміністративної панелі Drupal

Основними функціональними можливостями Drupal є:

1) Керування користувачами: гнучка система ролей і прав доступу, підтримка авторизації через OAuth, LDAP;

2) Інтерфейс редагування: Візуальні редактори (CKEditor), автозбереження, медіа-менеджери;

3) API: Підтримка REST, JSON:API, GraphQL для headless-архітектури;

4) Вбудована підтримка кількох мов, автоматичне перемикання;

5) Безпека: Розширене логування, контроль доступу, активна робота Security Team;

6) Модулі інтеграції: Підключення до сторонніх сервісів (Mailchimp, Google Analytics, Telegram через API тощо)

Серед основних переваг Drupal варто виділити гнучкість і масштабованість, що дозволяє використовувати систему як для невеликих сайтів, так і для великих корпоративних порталів; високий рівень безпеки, завдяки чому вона використовується навіть державними структурами, включаючи уряд США; розширюваність за рахунок тисяч безкоштовних модулів та активної спільноти розробників; а також орієнтованість на API, що дає змогу створювати headless-рішення та легко інтегрувати систему з чат-ботами, зокрема Telegram. Водночас до недоліків відносяться високий поріг входу — система складніша для освоєння порівняно з WordPress або Joomla; складність адміністрування без відповідного досвіду, оскільки багато налаштувань потребують технічної підготовки; і ресурсоємність, що вимагає більш потужного серверного середовища при масштабуванні.

Drupal може бути використаний як потужна backend-платформа для адміністрування ігрового контенту, особливо якщо розглядати його як headless CMS, що надає дані Telegram-боту через API. Це дозволяє зберігати інформацію про гравців, персонажів, фракції тощо, керувати доступом адміністраторів через панель, виводити дані до Telegram-бота через JSON API та легко розширювати логіку через кастомні модулі.

Проте, через складність налаштування та надлишкову гнучкість, Drupal краще використовувати у великих проєктах або у випадках, коли планується багато видів контенту та взаємодій.

1.2.2 Аналіз систем управління взаєминами із клієнтами на прикладі HubSpot CRM

HubSpot CRM — це хмарна система управління взаєминами з клієнтами (Customer Relationship Management), яка забезпечує комплексне ведення бази контактів, автоматизацію бізнес-процесів, маркетингову активність і комунікацію з клієнтами. HubSpot широко використовується в бізнесі, маркетингу, продажах і службах підтримки, особливо серед малих і середніх компаній [8]. Сер-

віс працює за моделлю SaaS (Software as a Service), тобто користувач отримує доступ до всіх інструментів через веб-інтерфейс без необхідності локального встановлення. Система умовно поділена на кілька функціональних блоків: CRM, маркетинг, продажі, підтримка, аналітика. Базовий функціонал CRM доступний безкоштовно, а інші модулі — на умовах підписки.

Архітектура та структура HubSpot CRM передбачає використання основних об’єктів (Entities), до яких належать контакти, компанії, угоди (deals), тікети та завдання, що формують базову модель даних. Ключовим елементом взаємодії є пайплайни — візуалізовані послідовності етапів роботи з угодами або зверненнями клієнтів (наприклад, «новий», «в роботі», «успішно завершено»), які дозволяють відстежувати стан кожного процесу (рис. 1.2).

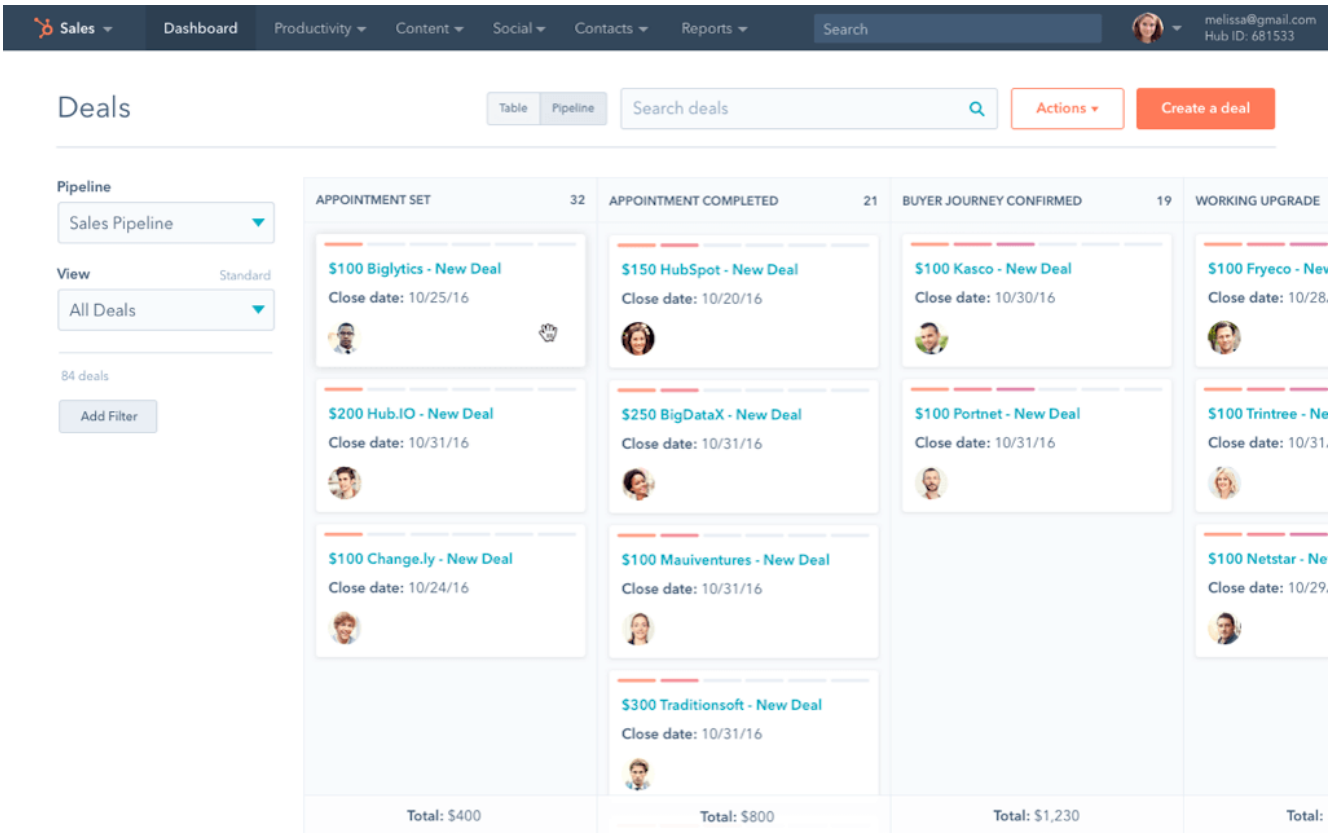

Рисунок 1.2 – Інтерфейс HubSpot CRM

Уся інформація про взаємодію з клієнтом — дзвінки, електронні листи, повідомлення — автоматично збирається в хронологічній стрічці, що забезпечує цілісне уявлення про історію спілкування. Також система підтримує чис-

ленні інтеграції, включаючи поштові сервіси, чати, сайти та месенджери; можливе з'єднання з Telegram через API або сторонні сервіси, що значно розширює функціональні можливості CRM.

Основні функціональні можливості HubSpot CRM включають ведення контактної бази з детальною інформацією про користувачів і повною історією взаємодій; управління угодами через створення пайплайнів, відстеження етапів продажів і прогнозування доходу; можливості email-маркетингу, що охоплюють створення розсилок, шаблонів та аналітику відкриттів і кліків; підтримку онлайн-чату та чат-ботів із можливістю автоматизації сценаріїв спілкування на сайті; аналітичні інструменти для формування звітів щодо воронок продажів, конверсій і активності менеджерів; доступ до REST API та Webhooks для інтеграції зі сторонніми сервісами, включаючи Telegram-ботів; а також автоматизацію робочих процесів для обробки подій, повідомлень і задач.

Перевагами HubSpot CRM є інтуїтивно зрозумілий інтерфейс, що підходить навіть користувачам без попереднього досвіду з CRM-системами; хмарна модель, яка не потребує встановлення і забезпечує доступ з будь-якого пристрою; наявність безкоштовної базової версії з підтримкою обліку контактів, угод і листування; широка підтримка інтеграцій із CMS, поштою та месенджерами, включно з Telegram (через Zapier, Make тощо); а також розвинена система автоматизації, яка дозволяє реалізовувати складні сценарії реагування на події.

Серед недоліків варто відзначити обмеження безкоштовного тарифного плану, де ключові функції аналітики та автоматизації доступні лише в платній версії; неадаптованість до ігрових сценаріїв, оскільки система орієнтована на класичні бізнес-процеси і не враховує специфіку ігрових механік (фракції, інвентар тощо); а також складність інтеграції з Telegram, яка потребує використання сторонніх інструментів або розробки власного API-зв'язку.

HubSpot CRM може частково використовуватися як система для ведення обліку користувачів, історії їх активностей або адміністративних дій. Наприклад, якщо адміністраторам потрібно бачити хронологію взаємодії з гравцем, фіксувати звернення, порушення чи запити.

Проте в умовах ігрового сервера, де контент має складну ієрархію (NPC, локації, фракції, скрипти тощо), HubSpot не забезпечує достатньої кастомізації. Тому воно радше розглядається як орієнтир або база для моделювання системи адміністрування, а не пряме рішення для ігрового середовища.

1.2.3 Аналіз панелей керування ігровими серверами на прикладі Pterodactyl Panel

Pterodactyl Panel — це сучасна, безкоштовна система з відкритим кодом для адміністрування ігрових серверів [9]. Вона побудована на основі Docker-контейнерів і призначена для надання повного контролю над запуском, управлінням, моніторингом та налаштуванням серверів із різними ігровими двигунами (рис 2.3).

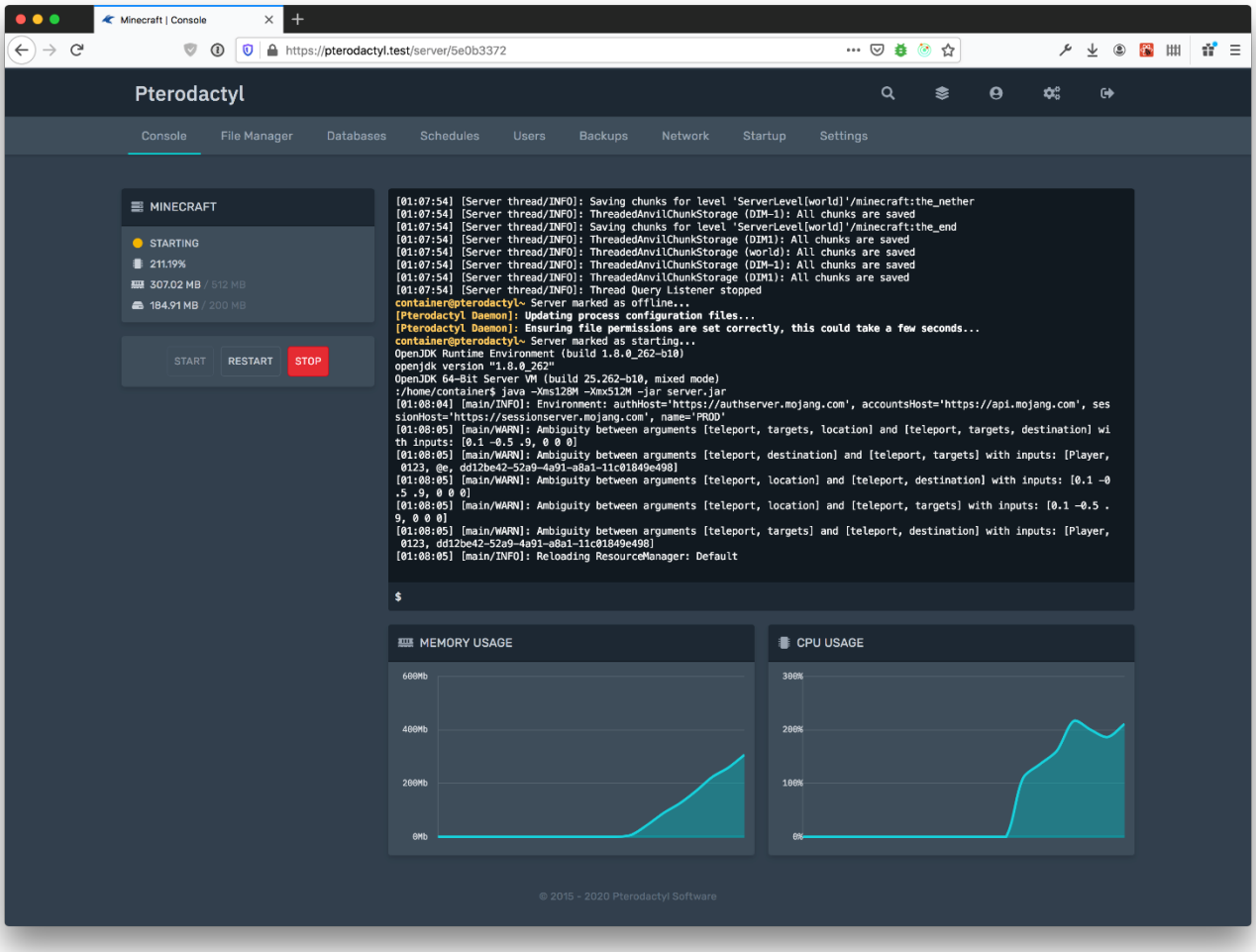

Рисунок 2.3 – Консоль керування сервером в Pterodactyl Panel

Pterodactyl Panel орієнтована на хостинг ігрових серверів, зокрема Minecraft, CS:GO, ARK, Rust, але також може використовуватись і для нестандартних сценаріїв, зокрема рольових серверів на базі Rage MP або аналогічних платформ. Вона складається з двох основних компонентів:

1. Панель (frontend + backend) — написана на PHP (Laravel) і надає зручний веб-інтерфейс.
2. Daemon (Wings) — агент на сервері, що виконує безпосередні дії через Docker.

Основні функціональні можливості Pterodactyl Panel включають централізоване керування ігровими серверами з можливістю їх запуску, перезапуску, зупинки та видалення; інтегрований файловий менеджер для перегляду, редагування та завантаження файлів; доступ до інтерактивної консолі для керування процесами гри в реальному часі; моніторинг використання ресурсів (CPU, оперативної пам’яті, дискового простору); система прав доступу, що дозволяє гнучко призначати ролі користувачам і адміністраторам; планувальник завдань для автоматизації дій (наприклад, резервне копіювання чи рестарт серверів); REST API для інтеграції з веб-сайтами, ботами або іншими інтерфейсами; а також механізми безпеки, зокрема ізоляція контейнерів, аудит дій користувачів і контроль доступу.

Перевагами Pterodactyl Panel є повний веб-контроль над серверами з будь-якого пристрою; масштабованість, що дозволяє обслуговувати десятки серверів одночасно; відкритий вихідний код і безкоштовне використання зі спільнотою, що активно підтримує проект; гнучкість у налаштуванні шаблонів запуску для різних ігрових рушіїв; а також можливість інтеграції з кастомними клієнтами або Telegram-ботами через API.

До недоліків можна віднести відсутність підтримки внутрішньоігрової логіки (гравці, персонажі, фракції не обробляються — лише інфраструктура серверів); відсутність мобільного додатку чи інтерфейсу у вигляді месенджера — керування можливе лише через браузер; а також потребу в технічній підго-

товці для налаштування, зокрема знань у галузі Docker, Linux і DevOps-практик.

Pterodactyl Panel демонструє ідею централізованого управління серверними ресурсами, однак не забезпечує керування ігровим контентом у вузькому сенсі (наприклад, гравцями, адміністративними командами, економікою гри). У контексті розробки Telegram-бота для адміністрування RP-сервера Rage MP, Pterodactyl може виконувати допоміжну роль — наприклад, для перезапуску серверу чи створення резервних копій, але не як основний інструмент управління контентом.

Найбільш релевантним її елементом є REST API, який може бути використаний для інтеграції з Telegram-інтерфейсом або власним бекендом, що розширює функціональність класичної панелі за межі її призначення.

1.3 Висновки до розділу

Аналіз існуючих систем адміністрування та керування ресурсами та контентом показав, що, не зважаючи на різноманіття доступних рішень, жодна з таких систем не задовольняє повною мірою специфічні індивідуальні потреби адміністрування рольових ігрових серверів, зокрема таких, що працюють на платформі Rage MP.

CMS-системи, як-от Drupal, демонструють високий рівень гнучкості у структурі контенту й надають широкі можливості інтеграції через API, проте вони складні у налаштуванні та не адаптовані до динамічної ігрової логіки. CRM-рішення, подібні до HubSpot, зручні у роботі з користувачами й автоматизацією, однак орієнтовані на бізнес-процеси, а не на специфіку ігрових сценаріїв. Панелі керування, як-от Pterodactyl, забезпечують потужний технічний контроль над серверами, але не працюють з внутрішнім ігровим контентом та не передбачають мобільного або чат-інтерфейсу для адміністраторів.

Отже, існує чіткий розрив між функціональністю доступних інструментів та реальними вимогами адміністрування RP-серверів, які потребують швидко-

го, зручного, адаптивного і захищеного інтерфейсу для взаємодії з ігровим світом у режимі реального часу.

Це дозволяє зробити обґрунтований висновок, що є необхідною розробка спеціалізованої клієнт-серверної системи адміністрування ігрового контенту з інтеграцією в Telegram, яка відповідатиме таким вимогам:

1. Наявність адміністративного функціоналу для управління сервером та користувачами;
2. Підтримка системи ролей та прав доступу для адміністраторів різного рівня;
3. Реалізація захищеного з’єднання між Telegram-ботом та серверною частиною системи;
4. Ведення журналу дій (логування) з можливістю фільтрації та перегляду історії подій;
5. Підтримка масштабованої бази даних для збереження інформації про гравців, адміністративні дії та внутрішньоігрові процеси;
6. Можливість реагувати на звернення гравців до адміністрації (репорти).

2 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Вибір мови програмування для реалізації серверної частини системи

Вибір мови програмування для реалізації серверної частини клієнт-серверної системи є ключовим фактором, що визначає не лише технічну ефективність розробки, але й можливості подальшої інтеграції з ігровим сервером Rage MP, а також підтримку вже існуючих функціональних модулів.

Платформа Rage MP офіційно підтримує дві мови серверної розробки — C# та JavaScript (Node.js). Обидва варіанти мають певні переваги, однак остаточний вибір потребує зваженого аналізу як технічних характеристик, так і особливостей реалізації адміністрування в рамках ігрового середовища.

Node.js є одним із найпоширеніших рішень для створення серверної логіки в середовищі Rage MP, насамперед завдяки активній спільноті, частим оновленням та широкому інструментарію для JavaScript-розробки [10]. Платформа базується на асинхронній неблокуючій моделі введення/виведення, що робить її особливо ефективною у випадках обробки великої кількості одночасних клієнтських запитів. Це актуально для ігрових серверів, де інтерфейс адміністратора має оперативно реагувати на події в реальному часі.

Значною перевагою є багатий екосистемний простір — зокрема, наявність таких бібліотек, як:

1) express — для побудови REST API;
2) node-fetch, axios — для зовнішніх HTTP-запитів;
3) node-telegram-bot-api — для роботи з Telegram Bot API;
4) sequelize або mongoose — для взаємодії з базами даних.

Завдяки цьому Node.js забезпечує швидку розробку, низький поріг входу та доступність великої кількості рішень із відкритим кодом.

Проте JavaScript, як мова із динамічною типізацією, має й низку обмежень. Зокрема, відсутність жорсткого контролю над типами даних підвищує

ризик помилок у логіці системи, що ускладнює її підтримку та масштабування при зростанні складності проєкту. Попри наявність рішень на кшталт TypeScript, які вводять статичну типізацію, вони додають складності до екосистеми й не завжди є виправданими у невеликих проектах.

Таким чином, Node.js добре підходить для побудови гнучких, швидких API та Telegram-ботів, але може бути менш придатним для реалізації структурованої логіки керування ігровим контентом, яка потребує суворої типізації та внутрішньої узгодженості даних.

Мова **Python** хоч і не підтримується нативно сервером Rage MP, є одним із найпопулярніших виборів для створення адміністративних бекендів, скриптів автоматизації, REST API та веб-сервісів. Його висока читабельність, лаконічний синтаксис та наявність розвинених бібліотек (наприклад, Flask [11], FastAPI [12], SQLAlchemy, Requests, Telethon) забезпечують ефективну реалізацію функціоналу взаємодії між інтерфейсом адміністратора (наприклад, Telegram-ботом) та серверною логікою.

Python також часто використовується для побудови панелей керування, створення систем логування, ведення статистики гравців та обробки адміністративних запитів. Завдяки високому рівню абстракції Python дозволяє реалізувати значний обсяг функціональності за відносно короткий час.

Основним недоліком у контексті ігрових серверів є відсутність прямої інтеграції з ігровим оточенням Rage MP, що передбачає необхідність проміжного шару або API між Python-сервером і ігровим сервером. Це ускладнює архітектуру, збільшує затримки в обробці запитів і підвищує вимоги до синхронізації між компонентами системи. Також Python не є найкращим вибором для задач із високою частотою запитів або жорсткими вимогами до продуктивності у реальному часі.

Отже, Python є зручним інструментом для побудови допоміжного функціоналу, але не оптимальним ядром для серверної частини, що має тісно взаємодіяти з ігровим середовищем.

Мова C# є повноцінно підтримуваною серверною мовою в екосистемі Rage MP, згідно з офіційною документацією [13]. Це дає змогу безпосередньо взаємодіяти з усіма ігровими об'єктами, подіями та методами через API, наданий самою платформою. Така інтеграція відкриває можливість для повноцінної побудови серверної частини, яка безпосередньо керує ігровим процесом, не потребуючи проміжних рівнів або обхідних архітектур.

C# — це статично типізована, об’єктно-орієнтована мова, що поєднує високу продуктивність, чітку типізацію та підтримку сучасних парадигм програмування. Завдяки структурованості та суворому контролю типів вона особливо ефективна у великих і складних програмних системах, де важливо забезпечити стабільність і передбачуваність логіки взаємодії між компонентами [14]. Поглиблений розгляд принципів роботи C# у середовищі CLR, управління пам’яттю, обробки подій, багатопотоковості й асинхронного програмування представлений у праці Джеффрі Ріхтера «CLR via C#», яка є авторитетним джерелом у .NET-екосистемі [15]. Завдяки таким характеристикам мова забезпечує широкі можливості для реалізації складних серверних систем, зокрема:

1. створювати модульну серверну архітектуру;
2. реалізовувати захищені REST API через ASP.NET Core [16];
3. використовувати ORM (наприклад, Entity Framework) для доступу до бази даних;
4. легко інтегрувати Telegram-бота за допомогою бібліотек Telegram.Bot або TLSharp.

Додатковою перевагою є можливість глибокого повторного використання наявного коду, якщо основний ігровий модуль вже реалізовано на C#. Це сприяє зменшенню складності взаємодії між адміністративною системою та ігровим сервером, забезпечуючи єдність логіки, збереження стилю програмування та відсутність дублювання функціоналу.

Таким чином, C# поєднує високу продуктивність, можливості для побудови складних API та повну сумісність з серверною логікою Rage MP. Це робить її оптимальним вибором для реалізації адміністративної системи з

Telegram-інтерфейсом, яка повинна безпосередньо керувати ігровим середовищем.

Незважаючи на популярність Node.js у середовищі Rage MP, вибір на користь C# обумовлений перевагами цієї мови у розробці структурованих, стабільних серверних рішень, де важливо дотримуватись суворої логіки типів, інкапсуляції та повторного використання коду. Для системи адміністрування, що має обробляти запити з Telegram-бота, взаємодіяти з базою даних та синхронізуватись із внутрішньою логікою ігрового модуля, C# є ефективним вибором як з точки зору надійності, так і продуктивності.

Варто зазначити, що наявність уже реалізованих з використанням мови C# компонентів у складі ігрового компоненту також є важливою передумовою вибору. Вона дозволяє забезпечити повторне використання існуючої логіки, зменшити дублювання коду, уникнути надмірної складності в інтеграції і, відповідно, підвищити загальну ефективність розробки системи.

Таким чином, мова програмування C# була обрана як оптимальний варіант для реалізації серверної частини системи адміністрування ігрового контенту, що інтегрується з Telegram. Такий вибір забезпечує як високий рівень контролю над логікою системи, так і ефективну взаємодію з ігровим сервером Rage MP, а також дає можливість побудови масштабованої та безпечної серверної інфраструктури.

2.2 Вибір фреймворку для реалізації серверної частини системи

Фреймворк є основою для розробки логіки серверної частини будь-якої інформаційної системи. Його вибір має ґрунтуватися на сумісності з основними платформами, продуктивності, гнучкості у розширенні, доступності документації та підтримки інтеграцій з зовнішніми сервісами. У контексті клієнт-серверної системи адміністрування рольового ігрового контенту, реалізованої на платформі Rage MP, важливим критерієм є тісна інтеграція з ігровим сервером та підтримка асинхронної взаємодії з Telegram.

У результаті технічного аналізу для реалізації серверної логіки було обрано GTANetworkAPI як нативний API-фреймворк для сервера Rage MP, а також Telegram.Bot SDK — офіційну .NET-бібліотеку для взаємодії з Telegram Bot API.

GTANetworkAPI — це офіційний програмний інтерфейс для розробки серверної частини ігрових модулів на платформі RAGE Multiplayer з використанням мови C#. Фреймворк забезпечує прямий доступ до внутрішньоігрових подій, гравців, транспортних засобів, об’єктів, а також дозволяє обробляти серверні події та створювати кастомну логіку [17].

Основні можливості GTANetworkAPI:

1) обробка подій підключення, виходу, смерті, телепортації, взаємодії з об’єктами тощо;
2) виклик клієнтських подій із серверної частини;
3) зберігання кастомних даних у гравця чи транспортного засобу;
4) підтримка C#-структур, словників, асинхронної логіки;
5) можливість побудови кастомних команд і систем доступу.

Фреймворк працює як вбудована серверна платформа, а не як окремий веб-сервер, що дозволяє безпосередньо реалізовувати логіку Telegram-бота в контексті ігрового процесу. Це суттєво знижує затримки в обробці команд і спрощує архітектуру.

Telegram.Bot SDK — це офіційна бібліотека для створення Telegram-ботів мовою C# [18]. Вона підтримує всі основні функції Telegram Bot API, включаючи надсилання та обробку повідомлень, inline-команди, callback-запити, клавіатури, а також підтримує асинхронне програмування (async/await) [4].

Telegram.Bot SDK було обрано з наступних причин:

1) повна підтримка Telegram Bot API відповідно до офіційної документації;
2) можливість запуску обробника повідомлень у фоні паралельно з логікою гри;
3) підтримка Polling і Webhook-режимів;

4) хороша інтеграція з типізованими структурами C#;
5) відсутність потреби у проміжному сервері або окремому фреймворку (наприклад, ASP.NET Core).

Таким чином, Telegram.Bot SDK дозволяє вбудувати функціонал Telegram-бота безпосередньо в модуль ігрового сервера, зберігаючи просту структуру проекту та мінімізуючи залежності.

2.3 Вибір технологій збереження та обробки даних

Збереження та обробка даних є критично важливою складовою будь-якої інформаційної системи, особливо у випадках, коли система взаємодіє з великою кількістю користувачів і потребує надійного механізму обліку, логування та обробки запитів. У контексті клієнт-серверної системи адміністрування ігрового контенту необхідно забезпечити збереження та доступ до інформації про гравців та інші ігрові сутності, адміністративні дії, події у грі, а також реалізувати логування використання функціоналу Telegram-боту.

Для забезпечення надійного зберігання й обробки даних у системах адміністрування зазвичай застосовуються два основні підходи до проєктування сховищ: реляційні (SQL) та нереляційні (NoSQL) бази даних. У залежності від структури даних, обсягу навантаження та вимог до масштабованості обирається відповідний тип СУБД.

Серед популярних СУБД можна виокремити Oracle, MySQL, Microsoft SQL Server, PostgreSQL та MongoDB. Ці рішення вже багато років поспіль стабільно посідають провідні місця у світових рейтингах (рис. 2.1). Згідно з аналітикою DB-Engines [19], зазначені СУБД демонструють найвищі показники поширення, активності спільноти та технічної підтримки. Їх обирають як великі корпоративні проєкти, так і розробники вебсервісів, зокрема через високу продуктивність, масштабованість і широкі можливості інтеграції. Серед лідерів також спостерігається стабільне зростання популярності MariaDB — відкритої

реляційної СУБД, що є форком MySQL і вирізняється високою продуктивністю, сумісністю з існуючими рішеннями та активною спільнотою розробників.

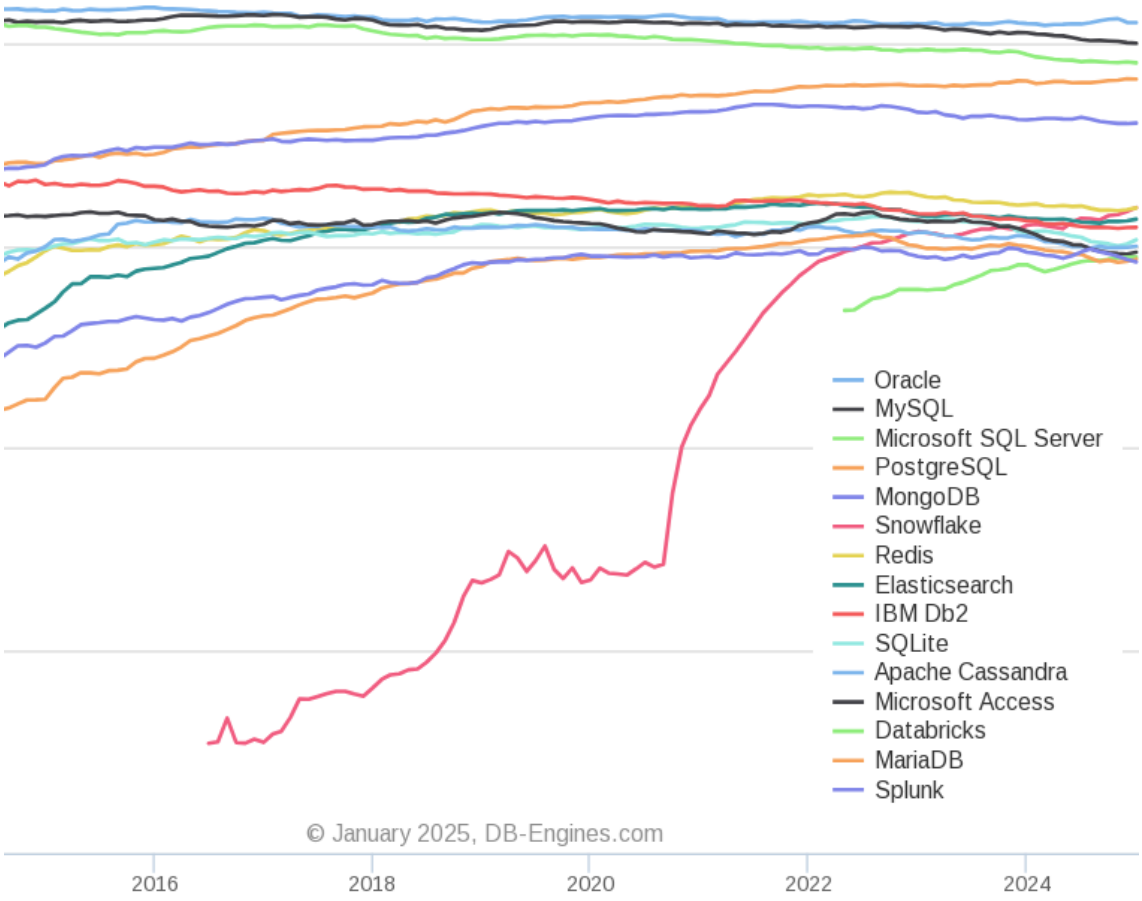

Рисунок 2.1 – Динаміка популярності СУБД у 2015–2025 роках

MariaDB є одним з найпопулярніших і найнадійніших форків MySQL, що забезпечує повну сумісність зі стандартом SQL, високу продуктивність, масштабованість і відкриту ліцензію [20]. Вона активно підтримується спільнотою та компанією MariaDB Corporation, що гарантує регулярні оновлення, виправлення безпеки та розширення функціональності.

Основні переваги MariaDB:

1) Продуктивність – підтримка індексів, оптимізованих JOIN-операцій, кешування запитів і можливість налаштування реплікації забезпечують стабільну роботу навіть при високому навантаженні, що важливо для активного ігрового сервера.

2) Сумісність з .NET – використання офіційного клієнта MySqlConnector або Pomelo.EntityFrameworkCore.MySql дозволяє легко підключити MariaDB до C#-середовища через ORM Entity Framework Core або безпосередні SQL-запити.

3) Масштабованість і підтримка великих обсягів даних – MariaDB дозволяє ефективно зберігати дані про велику кількість гравців [21], логів адміністративних дій, запити до Telegram-бота, а також забезпечує зручну побудову аналітики.

4) Відкритість та гнучкість – завдяки відкритому коду та повній сумісності з SQL-стандартами, MariaDB дозволяє міграцію та кастомізацію без обмежень, що часто трапляється у комерційних СУБД.

5) Сумісність із хостингом – MariaDB доступна на більшості VPS та хмарних платформ, що спрощує її використання у продакшн-середовищі.

PostgreSQL — потужна об’єктно-реляційна СУБД з відкритим кодом, яка підтримує більшість стандартів SQL та розширена низкою функцій, що виходять за межі класичної реляційної моделі [22].

PostgreSQL має низку вагомих переваг, серед яких потужна система транзакцій із повною підтримкою принципів ACID, вбудована підтримка збереження й обробки структурованих і напівструктурованих даних (JSON, XML), а також геоданих через модуль PostGIS. Система дозволяє створювати власні типи даних, функції та тригери, що забезпечує гнучкість і розширюваність у побудові складних додатків. Вона також вирізняється високою стабільністю при роботі з великими обсягами даних. Водночас, до недоліків PostgreSQL можна віднести складніші налаштування та адміністрування у порівнянні з MariaDB, а також меншу розповсюдженість серед хостинг-платформ за замовчуванням, що може ускладнити початкове розгортання для недосвідчених користувачів.

У порівнянні з MariaDB, PostgreSQL краще підходить для проєктів, де необхідна глибока обробка даних, складні запити та розширені типи. У свою чергу, MariaDB є більш легкою, швидкою в налаштуванні і цілком достатньою для класичної системи адміністрування контентом.

MongoDB — документно-орієнтована NoSQL СУБД, яка зберігає дані у форматі BSON (аналог JSON). Вона орієнтована на високу продуктивність, гнучкість структури даних та горизонтальну масштабованість [23].

Переваги:

1) Гнучка схема зберігання: немає фіксованої структури таблиць;
2) Ідеально підходить для динамічних, слабоструктурованих або змінних даних;
3) Висока швидкість запису й читання;
4) Зручна робота з об'єктами JSON-подібної структури.

Недоліки:

1) Відсутність жорстких зв’язків між даними (як у SQL);
2) Менша відповідність ACID-принципам (у порівнянні з класичними СУБД);
3) Потребує ретельного контролю цілісності даних на рівні прикладної логіки.

У порівнянні з MariaDB, MongoDB добре підходить для систем з великою кількістю неуніфікованих або гнучких даних (наприклад, кастомні налаштування, тимчасові дані, журнали), однак не є оптимальним вибором для структурованих облікових записів, ролей, прав доступу, де реляційна модель більш природна. Порівняльна характеристика досліджуваних СУБД наведена у таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика СУБД

Критерій	**MariaDB**	**PostgreSQL**	**MongoDB**
Тип	Реляційна (SQL)	Об'єктно-реляційна	Документна (NoSQL)
Гнучкість структури	Середня	Висока	Дуже висока
Продуктивність	Висока	Висока	Висока при записі
Масштабованість	Вертикальна	Вертикальна	Горизонтальна
Підтримка транзакцій	Повна (ACID)	Повна (ACID)	Часткова
Сумісність із .NET	Висока	Висока	Висока (через драйвери)
Найкраще для	Табличні дані	Аналітичні запити	Динамічний контент

З урахуванням характеру даних, які зберігаються у цьому проєкті (таблична структура, чітка логіка доступу, зв’язки між об’єктами), використання MariaDB є оптимальним. Вона поєднує простоту, продуктивність та сумісність із C#-середовищем і Telegram-інтеграцією. До того ж у вже існуючому модулі проєкту база даних також реалізована на MariaDB, і вона містить інформацію, з якою працюватиме Telegram-бот. Це дозволяє повторно використати наявні структури, уникнути дублювання даних і спростити архітектуру, оскільки не потрібно налаштовувати окрему базу для адміністративної системи.

2.4 Висновки до розділу

У ході аналізу технологій, придатних для реалізації серверної частини клієнт-серверної системи адміністрування ігрового контенту з інтеграцією в Telegram, було обґрунтовано доцільність використання мови програмування C#, нативного API-фреймворку GTANetworkAPI та бібліотеки Telegram.Bot SDK. Обрані інструменти повністю відповідають специфіці платформи Rage MP, забезпечують прямий доступ до ігрової логіки й дозволяють інтегрувати Telegram як адміністративний інтерфейс без потреби у проміжному сервері.

Для реалізації збереження даних у системі було обрано реляційну СУБД MariaDB, яка забезпечує оптимальний баланс між продуктивністю, надійністю, масштабованістю та сумісністю з екосистемою .NET. Проведене порівняння з альтернативними SQL- (PostgreSQL) і NoSQL- (MongoDB) базами даних підтвердило доцільність вибору MariaDB саме для системи з чіткою структурою доступу, суворою логікою зв’язків між об’єктами та необхідністю ефективної обробки транзакцій.

Таким чином, сформований технологічний стек дозволяє побудувати стабільну, масштабовану та безпечну серверну частину [1], що буде повністю відповідати вимогам інтеграції з ігровим сервером Rage MP і забезпечить ефективне адміністрування через Telegram-бота.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНИХ ЗАДАЧ

3.1 Вибір архітектури системи

Клієнт-серверна архітектура — це одна з найпоширеніших моделей організації взаємодії між компонентами в інформаційних системах [24]. Вона передбачає поділ системи на два основні логічні блоки: клієнт, що ініціює запити та взаємодіє з користувачем, і сервер, який обробляє ці запити, виконує бізнес-логіку та керує доступом до ресурсів (зокрема, бази даних). Така модель дозволяє досягти гнучкості, масштабованості та відокремлення відповідальностей, що є критично важливими для підтримки складних і динамічних систем.

Класична клієнт-серверна модель може бути доповнена розбиттям на шари, кожен із яких відповідає за свою частину обробки даних — від інтерфейсу до логіки та зберігання. У сучасних розробках часто використовується тришарова архітектура, яка включає:

- Presentation Layer (клієнтський рівень) — взаємодія з користувачем;
- Application/Logic Layer (бізнес-логіка) — обробка запитів, виконання правил;
- Data Layer (рівень даних) — доступ до сховищ, збереження/отримання інформації (рис. 3.1).

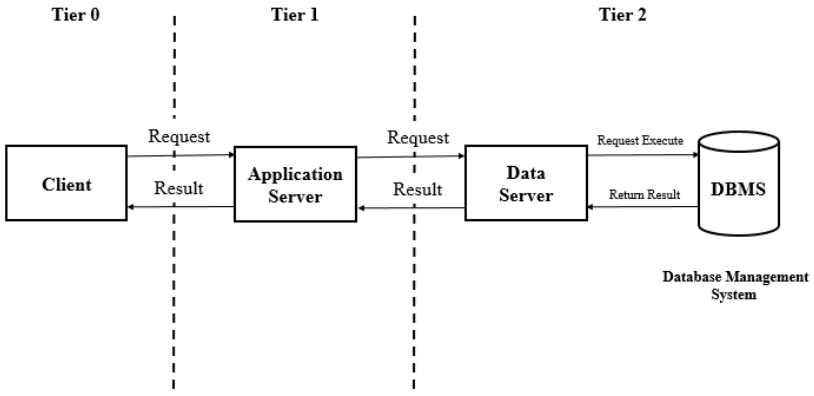

Рисунок 3.1 – Схема 3-шарової клієнт-серверної архітектури

У контексті реалізації системи адміністрування ігрового контенту з інтеграцією в Telegram архітектурне рішення повинно забезпечити одночасну роботу декількох підсистем, серед яких: ігровий сервер на платформі Rage MP, внутрішня логіка обробки команд адміністратора, та інтерфейс на основі Telegram-бота.

З огляду на технічні та функціональні вимоги, для реалізації було обрано модульну архітектуру з асинхронним обміном даними, де серверна частина ігрового моду виконує роль центрального вузла, а Telegram-бот виступає як зовнішній інтерфейс взаємодії з адміністративною логікою.

Система складається з декількох основних рівнів, які взаємодіють між собою для забезпечення віддаленого управління ігровим сервером:

1. Рівень клієнта (інтерфейс взаємодії з користувачем)

Представлений Telegram-ботом, який виконує роль користувацького інтерфейсу для адміністраторів. Він забезпечує:

- обробку текстових команд;
- відправку відповідей у чат;
- взаємодію через кнопки (callback);
- передачу введених даних у серверну частину.

2. Рівень обробки запитів (обробники подій)

Відповідає за первинну обробку повідомлень і дій користувача:

- аналізує тип оновлення (команда чи callback);
- виконує перевірку доступу;
- диспетчеризує виклики до відповідних команд;
- зберігає й обробляє тимчасові стани (наприклад, підтвердження дій).

3. Рівень команд (бізнес-функції)

Складається з набору окремих логічних одиниць (команд), кожна з яких:

- реалізує певну адміністративну функцію;
- має обмеження за ролями;
- може підтримувати підтвердження дії;

- повертає результат виконання на клієнтський рівень.

4. Сервісний рівень (бізнес-логіка)

Забезпечує реалізацію функціональності, що потребує:

- взаємодії з базою даних;
- складної перевірки прав чи умов;
- роботи з внутрішньоігровими об'єктами;
- централізованого логування.

Цей рівень ізольований від деталей Telegram API й відповідає лише за логіку.

5. Рівень доступу до даних (DAO / ORM)

Інкапсулює операції над базою даних:

- отримання й оновлення інформації про користувачів, гравців, налаштування;
- збереження журналів;
- читання й запис службових параметрів.

Взаємодія з базою виконується через ORM (LinqToDB), що гарантує безпечний і контрольований доступ до даних.

6. Сховище даних (база даних)

База даних зберігає всю необхідну інформацію для роботи системи:

- облікові записи адміністраторів Telegram;
- історію адміністративних дій;
- внутрішньоігрові параметри (репорти, транспорт, фракції);
- системні налаштування.

У якості СУБД використовується MariaDB.

У системі застосовано асинхронну модель взаємодії, у якій Telegram-бот ініціює запити, а ігровий сервер у відповідь виконує відповідну дію. Це дозволяє уникнути блокування головного потоку гри, реалізувати одночасну роботу з кількома адміністраторами та масштабувати логіку за рахунок розділення відповідальностей між модулями.

Причини вибору архітектури:

- Чітке розділення обов’язків — кожен рівень має свою зону відповідальності;
- Легше підтримувати та тестувати;
- Гнучкість у зміні реалізації окремих рівнів без впливу на інші;
- Масштабованість — можна замінити Telegram-бот іншим клієнтом без зміни бізнес-логіки.

Обрана архітектура забезпечує оптимальний баланс між простотою реалізації, ефективністю взаємодії та можливістю масштабування системи у майбутньому. Завдяки інтеграції Telegram-бота безпосередньо в логіку серверного модуля, система адміністрування отримує зручний, мобільний і функціональний інтерфейс управління контентом гри без необхідності додаткових проміжних сервісів.

В результаті аналізу можливої реалізації архітектури системи автором була запропонована діаграма розгортання системи, зображена у додатку А.1.

3.2 Розробка структури бази даних

Для забезпечення ефективної роботи клієнт-серверної системи адміністрування ігрового контенту, що взаємодіє з Telegram-ботом, було спроєктовано реляційну структуру бази даних, яка відповідає особливостям ігрової логіки та адміністративних функцій спираючись на наступні вимоги:

1) Зберігання інформації про користувачів (гравців та адміністраторів);
2) Відображення ролей, прав доступу та статусів;
3) Логування дій користувачів і Telegram-команд;
4) Масштабованість для подальшого розширення структури.

В результаті було створено таблиці:

1) Users – Дані про користувачів боту (ChatID, Name, Role, LogSettings):

```
CREATE TABLE `Users` (
  `Id` INT AUTO_INCREMENT PRIMARY KEY,
  `ChatId` BIGINT NOT NULL UNIQUE,
```

```
    `Name` VARCHAR(100) NOT NULL,
    `Role` TINYINT NOT NULL,
    `LogSettings` VARCHAR(50)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2) Logs – Лог використання Telegram-боту (Time, AdminID, Action, Target):

```
CREATE TABLE `Logs` (
    `Id` INT AUTO_INCREMENT PRIMARY KEY,
    `Time` DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    `AdminId` BIGINT NOT NULL,
    `Action` VARCHAR(255) NOT NULL,
    `Target` VARCHAR(255)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Окрім зазначених таблиць, Telegram-бот, як частина більшої системи має доступ до необхідних для виконання адміністративних функцій даних з інших, уже існуючих в системі таблиць, зокрема:

1) Accounts – дані про облікові записи гравців.
2) Characters – дані про ігрових персонажів гравців.
3) Reports та ReportMessages – дані про запити гравців до служби підтримки.

Запропонована структура бази даних відповідає вимогам до системи адміністрування RP-сервера, дозволяє гнучко масштабувати функціональність, інтегрується з MariaDB та забезпечує ефективну обробку запитів як із боку Telegram-бота, так і внутрішнього ігрового серверу.

3.3 Розробка серверної частини

Серверна частина системи реалізована на мові C# з використанням середовища .NET Core та фреймворку GTANetworkAPI, що дозволяє безпосередньо працювати з ігровим сервером Rage MP. Основними задачами серверного мо-

дуля є обробка команд адміністратора, взаємодія з базою даних, логування подій та забезпечення захищеної інтеграції з Telegram API.

Функціональна структура серверної частини поділяється на такі компоненти:

1) Рівень обробки Telegram-запитів – приймає повідомлення та команди з Telegram-бота;

2) Рівень взаємодії з користувачем - реалізація логіки обробки команд;

3) Сервіси бізнес-логіки – перевірка прав, виконання дій, генерація відповідей;

4) Data Access Layer – забезпечує доступ до даних, зокрема до MariaDB через ORM (LinqToDB);

5) Інфраструктурний рівень та утиліти – допоміжні сервіси: генерація клавіатур, шаблонів повідомлень.

Для реалізації цієї структури сформовано наступні пакети:

- Commands – містить реалізації окремих адміністративних команд, які викликаються з Telegram (наприклад, /kick, /ban, /save).
- Handlers – обробники вхідних запитів: MessageHandler і CallbackHandler. Вони аналізують повідомлення та маршрутизують їх до відповідних команд.
- Services – бізнес-логіка системи. Забезпечує виконання основних дій, перевірку прав, обробку команд, логування тощо.
- DAO (Data Access Object) – рівень доступу до бази даних. Відповідає за запити до таблиць MariaDB.
- Models – класи-моделі, які описують структуру сутностей (користувачі, ролі, репорти тощо).
- Utils – допоміжні інструменти: шаблонізатор, генератор клавіатур, форматування повідомлень.

Між пакетами існують наступні зв’язки:

- Пакет Commands викликає функції з Services.

- Handlers використовують сервіси, утиліти та моделі для обробки запитів.
- Services звертаються до DAO та працюють з Models.
- DAO оперує даними, які подаються у вигляді моделей (рис. 3.2).

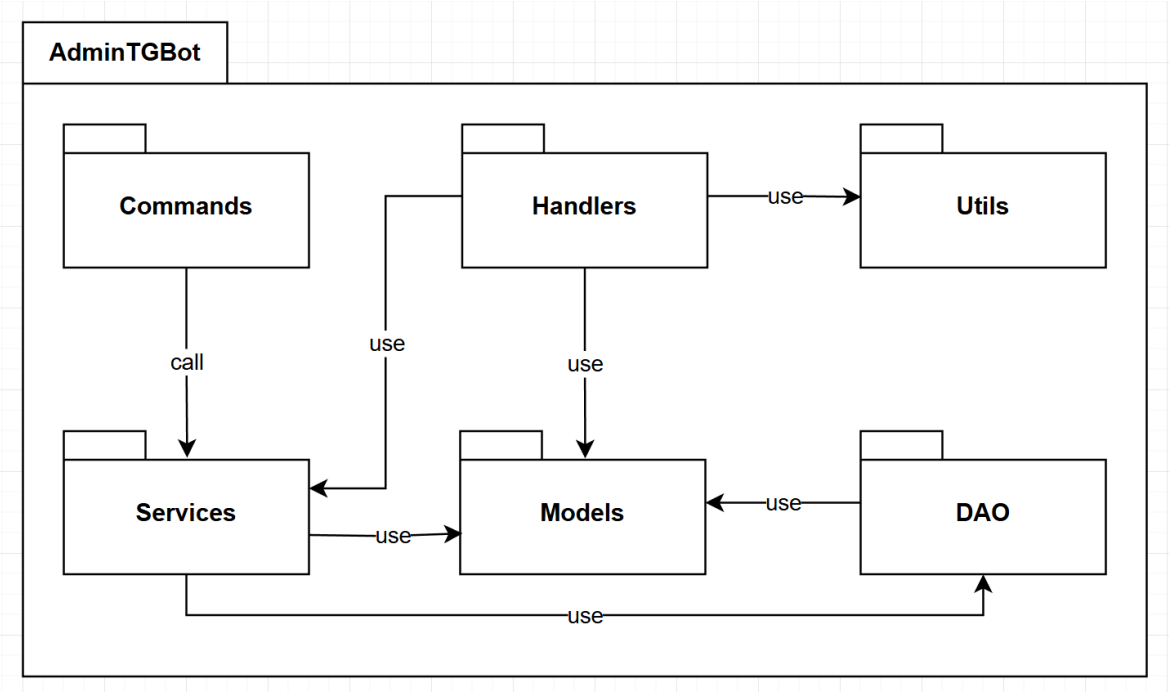

Рисунок 3.2 – UML діаграма основних пакетів системи

Інтерфейс адміністратора реалізований через Telegram-бота, що взаємодіє з сервером у режимі polling. Усі команди Telegram транслюються в дії, які впливають на ігрову логіку або базу даних. Наприклад, адміністратор через бот може перевірити інформацію про гравця, видати попередження або змінити статус фракції.

Завдяки такій структурі, серверна частина є гнучкою до змін і розширень — нові команди можна швидко додавати без ризику порушити існуючу функціональність. Всі сервіси взаємодіють між собою через чітко визначені інтерфейси, що відповідає принципам розділення відповідальностей (SRP) та інверсії залежностей (DIP), рекомендованих у сучасному об’єктно-орієнтованому програмуванні.

3.3.1 Підключення та налаштування Telegram API

Для реалізації Telegram-бота використано бібліотеку Telegram.Bot SDK, яка забезпечує повну підтримку Telegram Bot API у середовищі .NET. Вона є офіційною, стабільною та активно підтримується спільнотою. Бібліотека надає доступ до всіх можливостей Telegram-ботів, включаючи надсилання повідомлень, обробку команд, роботу з клавіатурами, повідомленнями callback, inline-режимом та файлами.

Ініціалізація, запуск та основні методи для функціонування Telegram-бота реалізовані у центральному класі BotSystem. Усі основні класи розробленої системи наведено у додатку А.2.

Для реалізації задуму було зроблено наступні кроки:

1. **Створення Telegram-бота**

Через сервіс BotFather у Telegram було створено нового бота та отримано унікальний токен доступу (botToken), що зберігається у Main.ServerSettings.TelegramBotToken та використовується для ініціалізації клієнта (рис. 3.3).

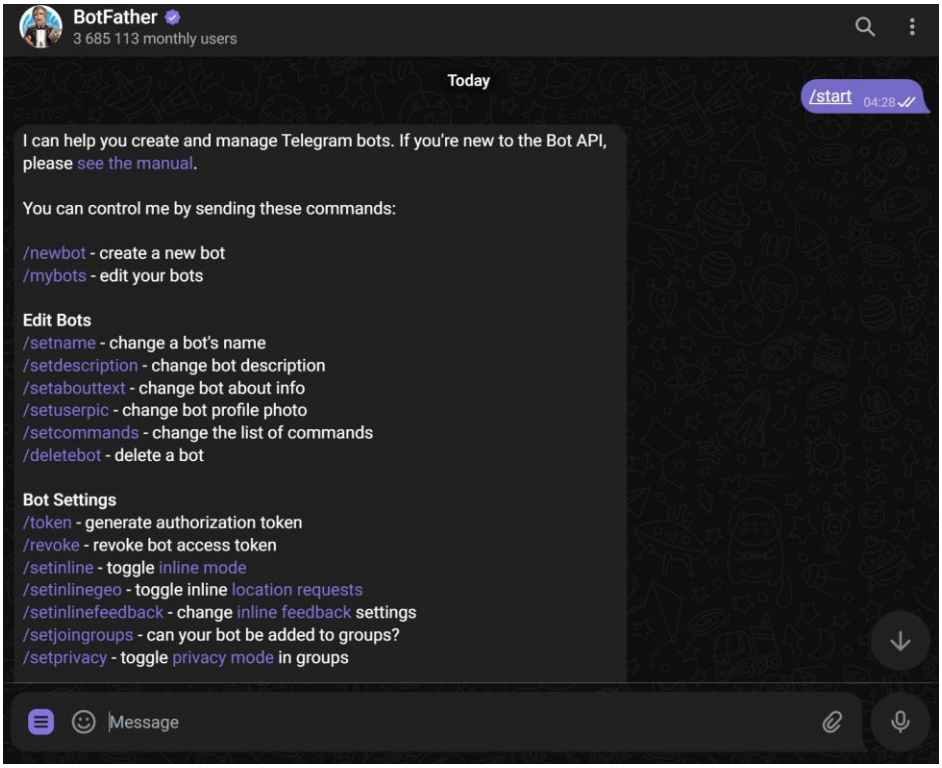

Рисунок 3.3 – Сервіс для створення Telegram-ботів BotFather

2. **Ініціалізація бота у C#**

Ініціалізація бота реалізується шляхом створення нового екземпляру класу TelegramBotClient із використанням токена доступу, як зазначено нижче:

```
private static readonly string BotToken =
Main.ServerSettings.TelegramBotToken;
private static readonly TelegramBotClient BotClient = new
TelegramBotClient(BotToken);
```

3. **Налаштування Polling-режиму**

Запуск бота відбувається разом із запуском сервера:

```
    [ServerEvent(Event.ResourceStart)]
        public static void OnResourceStart(){
            SendMessageAsync(Templ.GetText(Msg.ServerStarting));

            CommandDispatcher.RegisterAll();

            NAPI.Task.Run(async () =>
            {
                await SendMessageAsync(Templ.GetText(Msg.ServerReady));
                StartReceiving();
                foreach (var chatId in Users.Select(x => x.ChatId))
                {
                    await
CommandDispatcher.Get<CMD_Help>().Execute(Array.Empty<string>(), chatId);
                }
            });
        }
```

Сервер постійно слухає нові запити через StartReceiving(), тобто використовується режим long polling —:

```
public static void StartReceiving()
{
   Task.Run(() =>
   {
      BotClient.StartReceiving(
         updateHandler: MessageHandler.GetUpdateHandler,
         pollingErrorHandler: ErrorHandler,
         receiverOptions: new ReceiverOptions(),
         cancellationToken: Cts.Token
      );
   });
}
```

4. **Обробка оновлень**

Функція GetUpdateHandler визначає тип оновлення та залежно від цього встановлює, як саме потрібно обробляти оновлення:

```
private static async Task GetUpdateHandler(ITelegramBotClient botClient, Update update,
      CancellationToken cancellationToken)
   {
      if (update.Type == UpdateType.Message && update.Message?.Text != null)
         await MessageHandler.HandleTextMessage(update.Message);
      else if (update.Type == UpdateType.CallbackQuery && update.CallbackQuery != null)
```

```
            await CallbackHandler.HandleCallbackQuery(update.CallbackQuery);
        else
            throw new ArgumentOutOfRangeException();
    }
```

5. **Обробка команд**

Команди, що вводяться як текст у чаті Telegram, обробляються методом HandleTextMessage класу MessageHandler. Після валідації та визначення типу команди викликається відповідна реалізація через метод Execute(...) інтерфейсу ICommand, який реалізує кожна команда окремо:

```
    public static async Task HandleTextMessage(Message message)
    {
        var chatId = message.Chat.Id;
        var text = message.Text.Trim();

        if (!await BotSystem.IsAllowedChatId(chatId)) return;

        if (PendingCommands.ContainsKey(chatId))
        {
            await HandlePendingCommand(chatId, text);
            return;
        }

        string command = text.StartsWith("/") ? text[1..].ToLower() : "";

        if (!await CommandDispatcher.CheckCommand(command, chatId)) return;
```

```
            if (InputCommandHints.TryGetValue(command, out var hint))
            {
                PendingCommands[chatId] = command;
                await BotSystem.SendMessageToUserAsync(chatId, hint,
KeyboardBuilder.CancelButton);
            }
            else
            {
                await CommandHandlers[command].Invoke(new[] { "" }, 
chatId);
                await BotSystem.SendMenuAsync(chatId, 
Templ.GetText(Msg.CommandExecuted),

KeyboardBuilder.BuildMenuKeyboard(MenuType.MainMenu));
            }
        }
```

6. **Обробка callback-відповідей**

Команди, що потребують підтвердження дії, реалізовані через callback-кнопки Telegram. Після відправки запиту бот чекає підтвердження від адміністратора. Обробка відбувається централізовано через CallbackHandler, який викликає метод OnConfirm(...) у відповідної команди:

```
        public static async Task HandleCallbackQuery(CallbackQuery 
callbackQuery) {
            var chatId = callbackQuery.Message.Chat.Id;
        if (!ValidateCallback(callbackQuery, out var errorMessage, out var ac-
tion, out var userId)) {
            await BotSystem.AnswerCallbackQueryAsync(callbackQuery.Id, errorMessage);
```

```
            return;
        }

        if (!await BotSystem.IsAllowedChatId(chatId)) return;

        if (userId != chatId) {
            await BotSystem.AnswerCallbackQueryAsync(callbackQuery.Id,
                Templ.GetText(Msg.CallbackConfirmDenied));
            return;
        }

        if (!PendingCommands.TryGetValue(chatId, out var rawData)) {
            await BotSystem.AnswerCallbackQueryAsync(callbackQuery.Id,
                Templ.GetText(Msg.CallbackNotFound));
            return;
        }

        switch (action) {
            case "confirm":
                await ProcessConfirmedAction(chatId, callbackQuery.Message.MessageId, rawData);
                break;
            case "cancel":
                await CancelCallback(callbackQuery);
                break;
        }
        PendingCommands.Remove(chatId);
    }
```

7. **Захист команд**

В системі реалізовано перевірку прав адміністратора через порівняння Telegram ID з базою users. Доступ надається лише тим адміністраторам, які мають мінімально допустиму роль для використання конкретної команди.

```
public static bool IsAllowed(long chatId, Role requiredRole){
    var user = BotSystem.Users.FirstOrDefault(u => u.ChatId == chatId);
    return user != null && user.Role >= requiredRole;
}
```

8. **Інтерактивне меню команд (Reply-клавіатура)**

Для зручної взаємодії адміністратора з Telegram-ботом реалізовано reply-клавіатуру, яка дублює команди у вигляді інтерактивних кнопок. Клавіатура генерується залежно від меню (наприклад, MainMenu, ServerMenu, PlayerMenu тощо) за допомогою класу KeyboardBuilder.

При натисканні на кнопку користувач фактично надсилає текст команди, яка обробляється так само, як і введена вручну — через метод HandleTextMessage у класі MessageHandler, що спричиняє виклик методу Execute() відповідної команди. Такий підхід дозволяє мінімізувати помилки введення, пришвидшити виконання адміністративних дій, адаптувати інтерфейс під різні сценарії використання через меню (рис. 3.4).

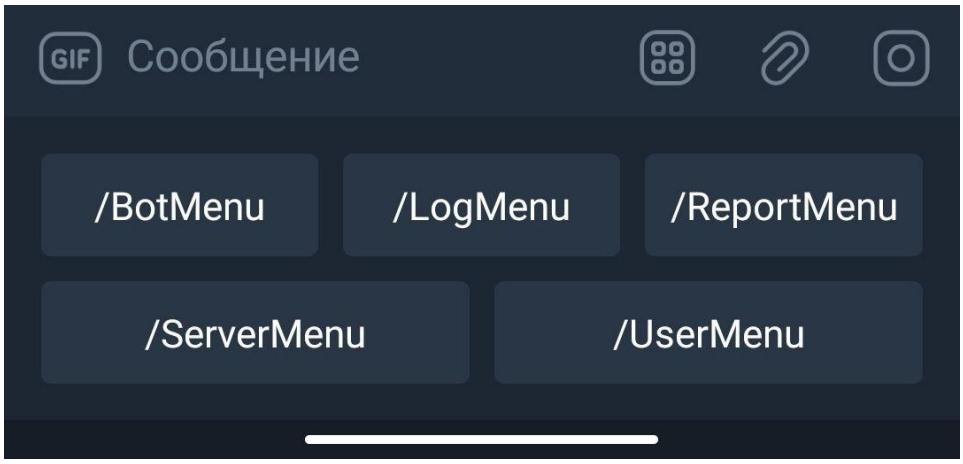

Рисунок 3.4 – Головне меню Telegram-бота

3.3.2 Розробка модуля реєстрації користувачів системи та редагування прав доступу

Модуль керування користувачами Telegram-бота забезпечує базову систему автентифікації та авторизації адміністративного персоналу. Основні функції реалізовані у сервісному рівні (BotService) та включають:

1. Реєстрацію нових користувачів з прив’язкою до їхнього ChatId, імені та відповідної ролі (наприклад, Admin, SerniorAdmin, ProjectManager);
2. Редагування користувачів (зміна ролі, ChatId або імені, видалення користувача);
3. Зберігання інформації про користувачів у базі даних (таблиця users) через ORM LinqToDB;
4. Перевірку прав доступу до команд на основі ролі користувача, яка відбувається перед виконанням будь-якої дії.

Інтерфейс користувача реалізований у вигляді меню (BotMenu), що дозволяє керувати обліковими записами безпосередньо через Telegram — додавати нових користувачів, змінювати їхні ролі або видаляти з системи. Процес роботи даного модулю представлено у додатку А.3.

Захист реалізовано через валідацію Telegram ID, а також порівняння ролі з мінімально необхідною для виконання конкретної команди. Це дозволяє гарантувати, що лише авторизовані особи мають доступ до критичних функцій сервера.

3.3.3 Розробка модуля логування подій ігрового сервера в реальному часі

Для забезпечення прозорості адміністративної діяльності та оперативного моніторингу внутрішньоігрових подій було реалізовано модуль логування в реальному часі. Основна мета модуля — збір, збереження та, за потреби, надси-

лання адміністраторам інформації про події, що відбуваються на сервері (наприклад, вхід/вихід гравців, бан/кік, помилки, спроби зловживань тощо).

Функціональність модуля включає:

1. Автоматичне фіксування подій у базі даних (таблиця logs) із зазначенням часу, типу події, ініціатора та опису дії;
2. Паралельне записування в лог-файл для резервного зберігання або швидкого перегляду;
3. Вивід логів у Telegram за запитом адміністратора (через підменю LogMenu), а також налаштування автоматичного надсилання при виникненні критичних подій;
4. Можливість завантажити лог-файл за певний період безпосередньо через Telegram-бота.

Основна логіка обробки реалізована у сервісі LogService, який виконує централізовану обробку та розсилання повідомлень. Автоматичне оповіщення про серверні події відбувається на основі налаштувань користувачів бота, щоб вони, могли обрати потрібні види сповіщень, уникаючи спаму непотрібними повідомленнями. Для цього реалізовано метод SendToTg:

```
public static void SendLogToTG(string text, nLog.Type logType)
{
      var users = BotSystem.Users.Where(u =>
u.LogSettings.Contains((int)logType)).ToList();

        foreach (var user in users)
        {
         BotSystem.SendMessageToUserAsync(user.ChatId, text);
        }
}
```

SendToTG() Викликається, коли за допомогою класу cLog відправляється будь-яке сповіщення:

```
public class cLog {
    private nLog Logger;

    public cLog(nLog logger)
    {
        Logger = logger;
    }

    public void Write(string text, nLog.Type logType = nLog.Type.Info)
{
        var msg = Logger.Write(text, logType);
        LogService.SendLogToTG(msg, logType);
    }
}
```

Таким чином, модуль логування забезпечує адміністрації інструменти для постійного контролю над ігровим процесом та швидкого реагування у випадку порушень чи несправностей (рис. 3.5).

```
private static readonly cLog Log = new cLog(new nLog( _reference: "TG_BOT"));

1 usage  Kevin GC
private static Task ErrorHandler(ITelegramBotClient botClient, Exception exception,
    CancellationToken cancellationToken)
{
    Log.Write($"Помилка Telegram API: {exception.Message}");
    return Task.CompletedTask;
}
```

Рисунок 3.5 – Приклад створення та використання об’єкту класу cLog

3.3.4 Реалізація адміністративного функціоналу для управління сервером та користувачами

Для зручності та ефективності адміністрування реалізацію відповідного функціоналу було структуровано за принципом розділення відповідальностей:

- ServerMenu — відповідає за команди управління сервером, такі як надсилання глобальних повідомлень, ініціалізація новин, збереження стану тощо. Меню представлено у вигляді Telegram-клавіатури, яка містить кнопки для виклику відповідних команд. Для обробки запитів, пов'язаних із сервером, використовується сервіс ServerService, який реалізує необхідну логіку — включаючи виклик внутрішніх методів сервера (наприклад, InitNews, SaveServer), перевірку прав користувача відповідно до його ролі та запис усіх дій до системи логування.
- PlayerMenu — дозволяє виконувати дії, пов'язані з гравцями: перегляд онлайну, кік, бан, хардбан, вивід списку адміністраторів тощо. Усі ці команди реалізовані у відповідних класах (CMD_Kick, CMD_Ban, CMD_HardBan, CMD_Admins, CMD_Players), а логіка обробки запитів винесена у PlayerService. Цей сервіс взаємодіє з базою даних та API ігрового сервера, перевіряє активність гравців і їхній статус, а також забезпечує перевірку ролей адміністратора, що виконує команду.

Кожна команда має відповідну реалізацію в окремому класі, який реалізує інтерфейс ICommand або, для команд, що потребують підтвердження, — IConfirmableCommand. Вони виконуються після введення текстової команди або натискання відповідної кнопки в меню, після чого ініціюється метод Execute, що викликає відповідну логіку сервісного рівня. Якщо команда вимагає додаткової взаємодії (наприклад, підтвердження дії через інтерфейс Telegram), то обробка продовжується через CallbackHandler.

Такий підхід дозволяє забезпечити:

- модульність та чіткий розподіл обов'язків;
- можливість масштабування без порушення цілісності системи;

- повторне використання бізнес-логіки в інших частинах проєкту (наприклад, у внутрішньоігровій адміністративній панелі);

Таким чином, реалізована структура не лише спрощує адміністрування сервером, а й дозволяє гнучко реагувати на зміну вимог до системи або додавання нового функціоналу без ризику порушення існуючих механізмів. У додатку А.4 зображена діаграма активності для процесу обробки команд користувача.

3.3.5 Розробка системи модерації внутрішньо-ігрового чату технічної підтримки

Система модерації чату технічної підтримки реалізована як окремий функціональний модуль Telegram-бота, що дозволяє адміністраторам оперативно реагувати на звернення гравців (репорти), не заходячи безпосередньо в гру.

Інформація про звернення зберігається в базі даних у двох таблицях:

- Report — містить основні дані про звернення (ідентифікатор, час створення, статус, автор тощо);
- ReportMessages — зберігає повну історію повідомлень у рамках кожного звернення, включаючи відповіді адміністрації та уточнення користувачів.

Для взаємодії з репортами в Telegram реалізовано ReportMenu, яке при відкритті відображає список усіх відкритих звернень. Кожне звернення може бути оброблено в один клік — адмін може прочитати суть запиту, відповісти або закрити звернення. Додатково, бот інформує про нові звернення в реальному часі (рис. 3.6).

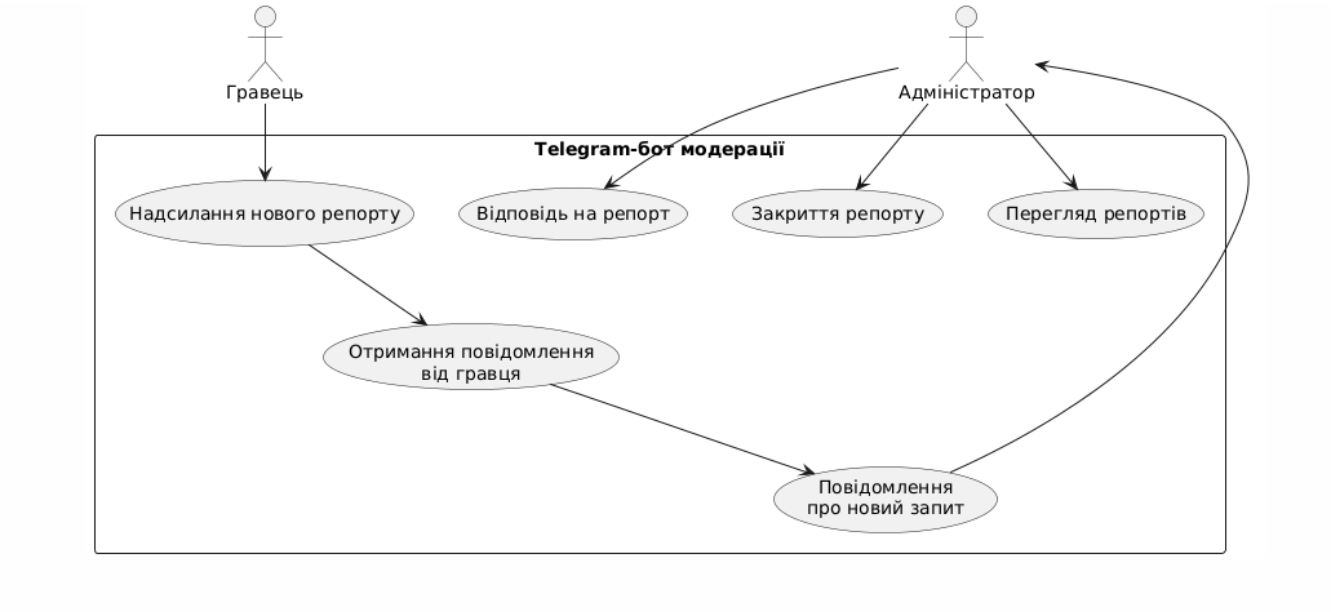

Рисунок 3.6 — Use-case діаграма роботи модуля обробки репортів

Внутрішня логіка обробки запитів реалізована через окремі команди та сервіс, які:

- отримують список актуальних репортів із бази;
- забезпечують форматування повідомлень для Telegram;
- дозволяють відповісти на запит (відповідь зберігається в базі як новий запис у ReportMessages);
- дозволяють позначити звернення як закрите.

Уся взаємодія із гравцем здійснюється з урахуванням його ID у грі, а логування дій адміністратора ведеться для контролю та аналітики.

Такий підхід дозволяє:

- здійснювати підтримку гравців без постійної присутності в грі;
- зберігати всю історію комунікацій;
- прискорити реакцію на технічні чи поведінкові порушення.

Варто зазначити, що дане рішення ніяк не конфліктує з уже наявним у грі меню репортів, дозволяючи одночасно працювати адміністраторам як і перебуваючи на ігровому сервері, так і використовуючи Telegram-бота а також може розширюватися додатковим функціоналом за потреби, як-от можливість здійснювати адміністративні функції над гравцем під час розгляду його звернення.

3.4 Тестування розробленої системи

Тестування програмного забезпечення — це цілеспрямований процес верифікації та валідації програмного продукту, який полягає у перевірці того, що розроблена система відповідає вимогам замовника, функціонує належним чином за різних умов і не містить критичних помилок. Метою тестування є виявлення дефектів, зниження ризику збоїв у реальному середовищі, забезпечення якості, стабільності та безпеки програмного забезпечення.

У життєвому циклі розробки тестування займає центральне місце та може виконуватись на різних етапах: від перевірки окремих модулів до повноцінного тестування інтегрованої системи. Залежно від методики, тестування також поділяється на функціональне, інтеграційне, регресійне, навантажувальне та інші види, що допомагають охопити як логіку виконання, так і поведінку системи в критичних ситуаціях.

Відповідно до способу проведення, тестування поділяється на:

- автоматизоване тестування — виконується за допомогою спеціалізованих фреймворків, скриптів або програм, що дозволяє повторно запускати тести з мінімальними витратами часу, особливо ефективне при регресійних перевірках;
- ручне тестування — передбачає безпосередню участь тестувальника, який взаємодіє з інтерфейсом програми, оцінює її поведінку, вводить різні комбінації даних і фіксує результати вручну.

Ручне тестування залишається важливим етапом, особливо у проєктах із динамічним інтерфейсом, логічними сценаріями або складною взаємодією з користувачем, де автоматизація не завжди доцільна чи економічно виправдана.

Головною метою тестування є перевірка коректної та стабільної роботи функціоналу клієнт-серверної системи адміністрування, зокрема — взаємодії між Telegram-ботом, ігровим сервером та базою даних. Особливу увагу приділено виконанню адміністративних команд, правильності перевірки ролей, об-

робці помилок, логуванню подій, а також надійній передачі даних між компонентами системи.

Тестування має підтвердити:

- відповідність реалізованого функціоналу технічному завданню;
- стійкість системи до неправильної або несподіваної поведінки користувача (наприклад, недійсні команди, відсутність параметрів тощо);
- коректну обробку запитів Telegram API та асинхронних операцій;
- безпечне збереження даних у MariaDB без втрати інформації.
- Кожне оновлення або зміна у бізнес-логіці має супроводжуватись відповідними перевірками, щоб уникнути регресій.
- Вторинними цілями тестування є:
- виявлення всіх дефектів, що можуть негативно вплинути на досвід адміністратора;
- оцінка ризиків, пов’язаних із порушенням цілісності даних, логуванням або ролями доступу;
- документування помилок та інформування розробників;
- перевірка зручності використання системи (юзабіліті), особливо з мобільних пристроїв;
- підтвердження захищеності доступу (наприклад, несанкціонований доступ до команд з боку користувача без відповідної ролі);
- забезпечення узгодженості роботи Telegram-бота з різними версіями клієнта Telegram (десктоп, мобільний, веб).

Таким чином, тестування не лише виявляє помилки, а й формує впевненість у готовності системи до реального використання в умовах живого проєкту.

Для перевірки всіх ключових аспектів роботи системи було застосовано ручне тестування, яке дозволило врахувати специфіку Telegram-інтерфейсу, асинхронної взаємодії та бізнес-логіки адміністративних команд.

Рішення про відмову від автоматизованого тестування на користь ручного було прийнято оскільки:

- система має розвинений інтерфейс через Telegram, взаємодія з яким зручно перевіряється вручну;
- основна частина команд пов'язана з обробкою станів гри в реальному часі, які складно відтворити автоматизовано без повноцінної емуляції ігрового середовища;
- обсяг функціоналу порівняно компактний, що дає змогу оперативно перевірити всі ключові сценарії вручну;
- ручне тестування дозволяє одночасно здійснювати візуальну перевірку правильності відповідей бота, UI-елементів (наприклад, inline-кнопок), форматування повідомлень, що складно покрити автотестами.

Тестування охоплює такі типи:

Функціональне тестування:

- перевірка доступу до команд відповідно до ролей (перевірка RoleManager);
- тестування виконання адміністративних команд (/kick, /ban, /global тощо);
- перевірка сценаріїв із підтвердженням дій (callback);
- тестування обробки помилкових запитів (невалідні аргументи, скасування команд);
- перевірка правильності логування подій у таблицю Logs;
- перевірка коректної роботи модуля керування користувачами бота;
- перевірка коректної роботи модуля технічної підтримки

Нефункціональне тестування:

- тестування стійкості до великої кількості запитів (імітація одночасної роботи декількох адмінів);
- перевірка збереження даних після аварійного завершення сесії;
- тестування стабільності роботи Telegram-бота у фоновому режимі;
- сумісність із різними пристроями (Telegram Desktop, Android, iOS, Web);

- перевірка захищеності логіки (неможливість виконання команд без належної авторизації);
- перевірка працездатності кнопок клавіатури (callback-інтерфейсів);
- стрес-тестування для перевірки роботи системи при пікових навантаженнях.

Для виконання поставлених задач було сформовано відповідні тест-кейси (табл. 3.1), які охоплюють основні сценарії взаємодії з Telegram-ботом у рамках адміністративної системи. Тестування проводилося з урахуванням різних ролей користувачів, варіантів введення команд, а також перевірки роботи критично важливих функцій, таких як обробка репортів, авторизація користувачів, відправка глобальних повідомлень та взаємодія з базою даних.

Таблиця 3.1 – Тест-кейси

№	Назва	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Перевірка доступу без ролей	Користувач без прав розпочинає роботу з ботом	Повідомлення про відмову у доступі	Відмова в доступі отримана	Пройдено
2	Перевірка доступу за роллю	Користувач із роллю Admin виконує команду із вищим рівнем доступу	Повідомлення про відмову у доступі	Відмова в доступі отримана	Пройдено
3	Перевірка коректного виконання команд	Користувач виконує команду /help	Повідомлення із довідковою інформацією	Отримано повідомлення з довідковою інформацією	Пройдено
4	Перевірка надсилання невалідних повідомлень	Користувач надсилає повідомлення, що містить неіснуючу команду	Повідомлення про відсутність команди	Отримано повідомлення «Невідома команда»	Пройдено
5	Перевірка команд із параметрами	Користувач вводить команду /kick та вказує параметр 1	Гравець з id 1 кікнутий, отримано повідомлення	Гравець кікнутий, повідомлення отримано	Пройдено

Продовження таблиці 3.1

6	Перевірка передачі некоректних параметрів	Користувач вводить команду /kick та вказує параметр «abc»	Отримано повідомлення про введення некоректного параметра	Повідомлення надіслано, відміна операції	Пройдено
7	Перевірка коректної роботи складних команд із відповідями	Користувач вводить команду /global Тестове повідомлення	Підтвердження через кнопку, повідомлення на сервер	Повідомлення підтверджено і відправлено	Пройдено
8	Перевірка можливості додавання користувачів бота	Користувач із правами супер-адміністратора створює нового користувача	Користувача успішно створено, він може користуватися ботом	Додано нового користувача, отримано повідомлення про успіх	Пройдено
9	Перевірка можливості редагування прав користувачів	Користувач із правами супер-адміністратора змінює роль іншого адміністратора з Admin на SeniorAdmin	Користувач отримав нову роль та доступ до нових команд	Користувач отримав нову роль та доступ, повідомлення про успіх	Пройдено
10	Перевірка роботи модуля модерації звернень гравців	Користувач бере на розгляд звернення гравця, надає відповідь та закриває його	Гравець отримав відповідь, репорт закрито	Гравець отримав відповідь, репорт закрито	Пройдено
11	Перевірка коректного надходження логів	Користувач вмикає отримання усіх сповіщень	Усі серверні логи надсилаються користувачу в режимі реального часу	Користувач отримує усі логи, що надсилаються за допомогою класу cLog	Пройдено

Як бачимо, усі тест-кейси пройдено успішно, розроблена система стабільно працює й задовольняє поставлені вимоги.

ВИСНОВКИ

У бакалаврській дипломній роботі було реалізовано клієнт-серверну систему адміністрування рольового ігрового серверу з інтеграцією в Telegram, метою якої є спрощення та оптимізація управління внутрішньоігровими процесами адміністрування. В рамках дослідження було проаналізовано сучасні підходи до побудови клієнт-серверних архітектур, зокрема трирівневі моделі з виокремленням логіки обробки даних, рівня взаємодії з користувачем та рівня доступу до бази даних. Вивчено особливості використання Telegram-ботів як альтернативного інтерфейсу адміністрування у порівнянні з класичними веб-панелями.

Під час роботи були досліджені системи, що надають адміністративний функціонал для керування контентом та користувачами, зокрема CRM та CMS рішення. Було визначено їх переваги (зручність у веб-інтерфейсі, розширені панелі керування), але також виявлено недоліки — громіздкість, складність у розгортанні та залежність від сторонніх хостингів або веб-серверів. На основі цього було обґрунтовано доцільність створення Telegram-орієнтованої системи, яка є мобільною, легко інтегрується та забезпечує достатній рівень функціональності для оперативного адміністрування.

Для реалізації системи було обрано GTANetworkAPI як основний фреймворк для створення логіки серверу на платформі Rage MP, Telegram.Bot SDK для побудови інтерфейсу взаємодії з адміністраторами, а також ORM LinqToDB для доступу до реляційної бази даних MariaDB, де зберігаються всі ключові дані про користувачів, логування дій, ігрові об'єкти та системні параметри. На основі модульної багаторівневої архітектури було створено ізольовані компоненти для обробки команд, виконання бізнес-логіки, взаємодії з базою даних та утилітарних сервісів.

У процесі реалізації було розроблено гнучку систему ролей і прав доступу, систему підтвердження адміністративних дій, механізм формування динамічних клавіатур Telegram, модулі логування та модерації, а також інтеграцію з

внутрішньоігровими сутностями. Усі адміністративні дії можуть бути виконані за кілька кліків безпосередньо з мобільного пристрою, що значно підвищує ефективність і мобільність адміністрування.

Система пройшла повний цикл ручного тестування, яке підтвердило її стабільність, надійність та відповідність вимогам. У ході тестування було перевірено всі основні функції: обробка команд, взаємодія з гравцями, логування, робота з ролями, коректність відповіді Telegram-бота, ігрова інтеграція тощо. Особливу увагу приділено перевірці реакції на нестандартні ситуації, помилки введення та навантаження на систему.

Розроблене рішення демонструє високий потенціал до подальшого розвитку — розширення функціоналу, підтримки нових платформ, підключення сторонніх сервісів або створення аналітичних панелей. Таким чином, поставлені цілі було досягнуто повністю. Проєкт не лише підтвердив практичну доцільність інтеграції Telegram у систему адміністрування ігрового контенту, а й відкрив можливості для подальшого розвитку у напрямках автоматизації аналітики, інтеграції з іншими сервісами (наприклад, CMS або CRM), розширення інтерфейсу та підтримки альтернативних каналів керування (наприклад, Discord).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2013. — 560 ст.

2. RAGE Multiplayer. Офіційний сайт. URL: https://rage.mp (дата звернення: 8.05.2025)

3. Telegram Bot API documentation. URL: https://core.telegram.org/bots/api Дата звернення: 8.05.2025.

4. Iancu M. Hands-On Telegram Bot Development. Packt Publishing, 2021. — 306 ст.

5. Buttle, F., & Maklan, S. Customer Relationship Management: Concepts and Technologies. 4th ed. – London: Routledge, 2019. – 512 с.

6. Berry, M. J. Drupal 9 Module Development: Get up and running with building powerful Drupal modules and applications. – Birmingham: Packt Publishing, 2021. – 480 с.

7. Drupal. Офіційний сайт. URL: https://www.drupal.org (дата звернення: 10.05.2025)

8. Streamline Your Entire Business with a Free CRM | HubSpot. HubSpot | Software & Tools for your Business - Homepage. URL: https://www.hubspot.com/products/crm (дата звернення: 10.05.2025).

9. Pterodactyl – Game Server Management Panel. Офіційний сайт. URL: https://pterodactyl.io/ (дата звернення: 10.05.2025)

10. RAGE Multiplayer Wiki – Getting Started with Client-side. URL: https://wiki.rage.mp/wiki/Getting_Started_with_Client-side (дата звернення: 12.05.2025)

11. Welcome to Flask – Flask Documentation (3.1.x). URL: https://flask.palletsprojects.com/ (дата звернення: 12.05.2025).

12. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 12.05.2025)

13. RAGE Multiplayer Wiki – Development with Visual Studio Code. URL: https://wiki.rage.mp/wiki/Development_with_Visual_Studio_Code (дата звернення: 12.05.2025).

14. C# Guide – .NET managed language. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 12.05.2025).

15. Richter J. CLR via C# (4th Edition). Microsoft Press, 2013. — 896 ст.

16. Freeman A., Sanderson A. Pro ASP.NET Core 6. Apress, 2022. — 1080 ст.

17. RAGE Multiplayer Wiki – Server-side CSharp events. URL: https://wiki.rage.mp/wiki/Server-side_CSharp_events (дата звернення: 12.05.2025).

18. GitHub – TelegramBots/Telegram.Bot: .NET Client for Telegram Bot API. URL: https://github.com/TelegramBots/Telegram.Bot (дата звернення: 12.05.2025).

19. DB-Engines Ranking. URL: https://db-engines.com/en/ranking (дата звернення: 17.05.2025).

20. MariaDB Foundation – MariaDB.org. URL: https://mariadb.org (дата звернення: 19.05.2025).

21. McCool C., Kennedy J. Learning SQL (3rd Edition). O'Reilly Media, 2020. — 384 ст.

22. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 19.05.2025).

23. MongoDB Official Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 19.05.2025).

24. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2015. — 792 p.

ДОДАТКИ

Додаток А (обов’язковий).

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ АДМІНІСТРУВАННЯ ІГРОВОГО КОНТЕНТУ З ІНТЕГРАЦІЄЮ В TELEGRAM (СЕРВЕРНА ЧАСТИНА)»

Діаграма розгортання системи адміністрування через Telegram-бот

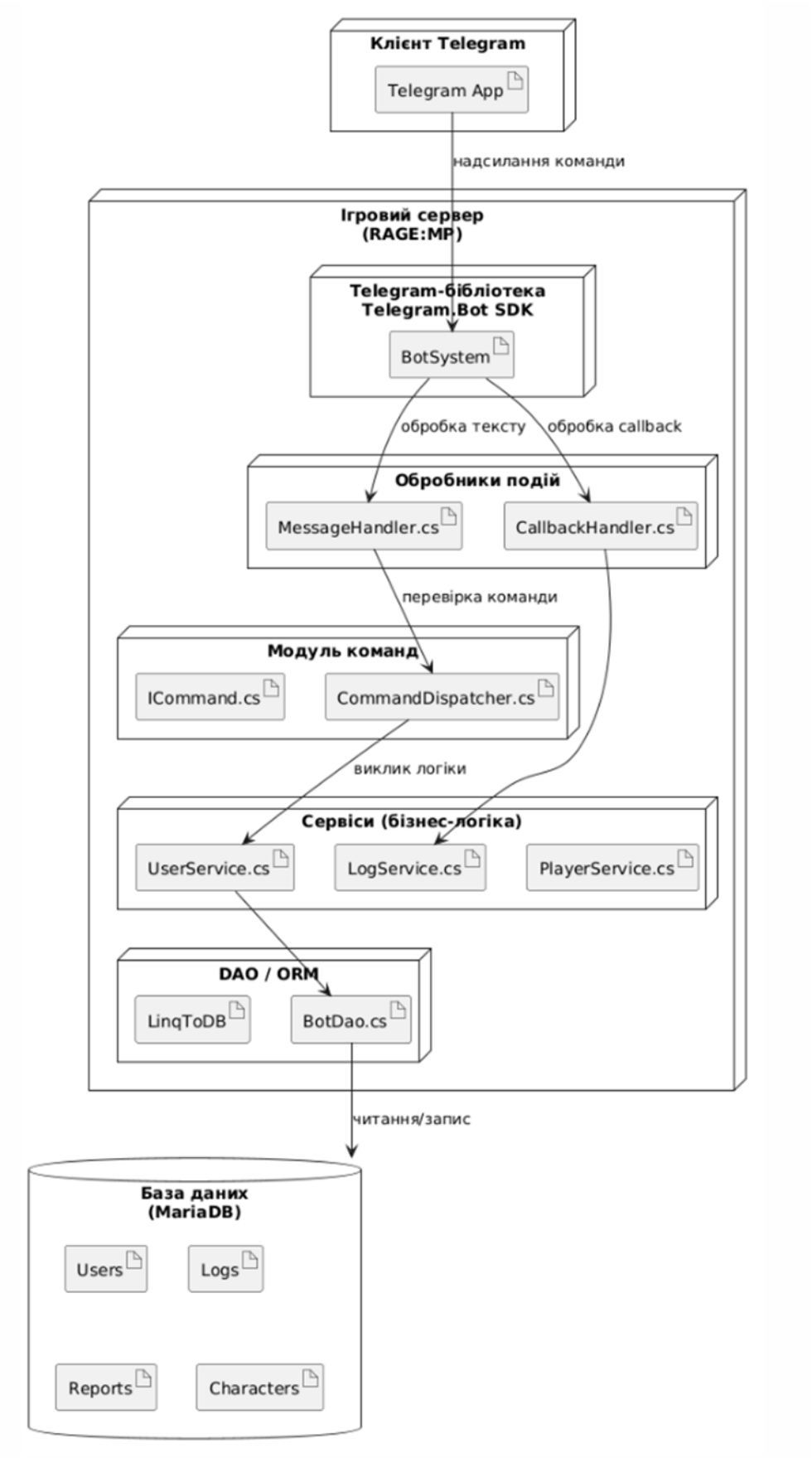

Рисунок А.1 – Діаграма розгортання системи адміністрування через Telegram-бот

Діаграма основних класів системи

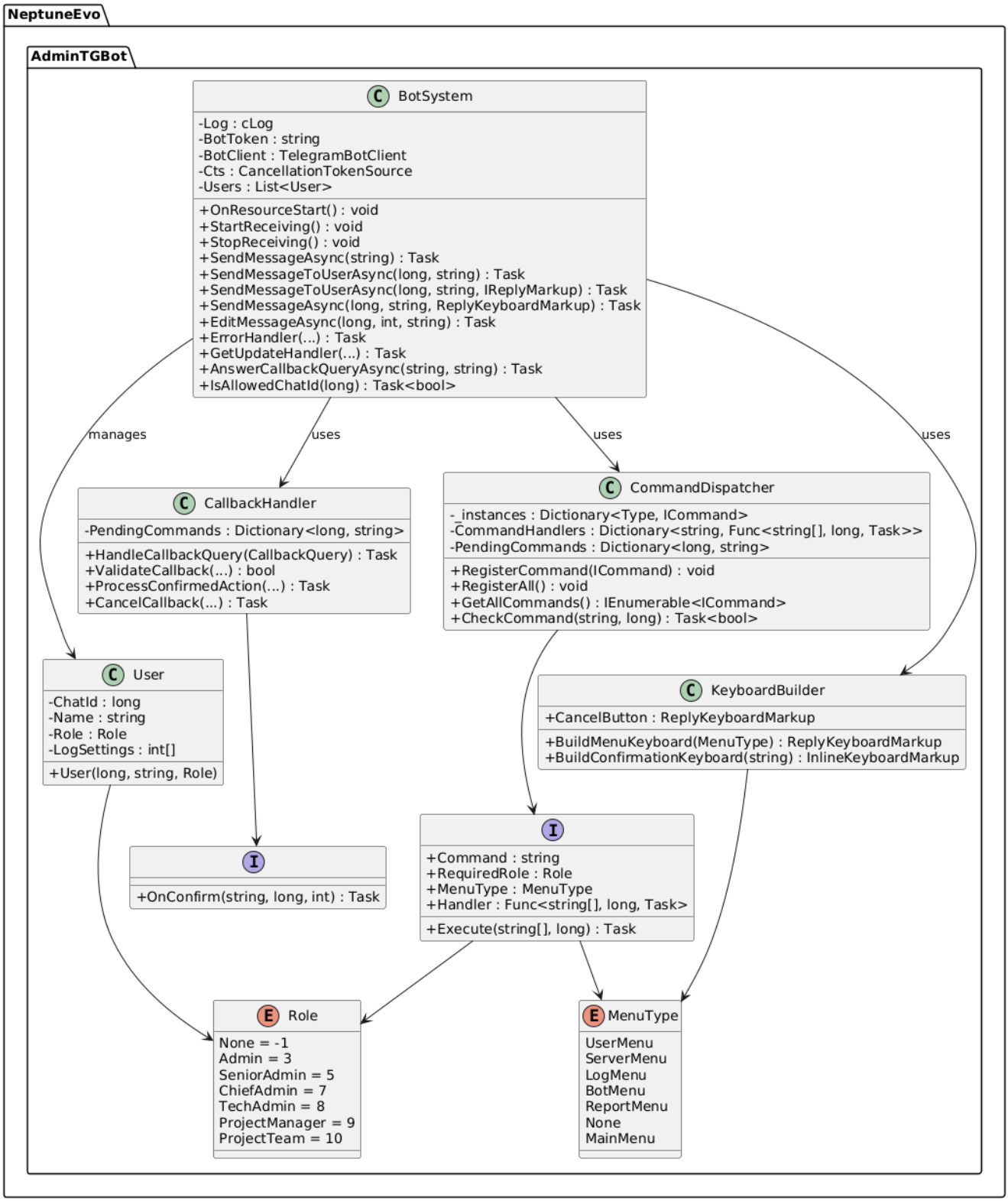

Рисунок А.2 – Діаграма основних класів системи

Діаграма активності процесу додавання нового користувача системи

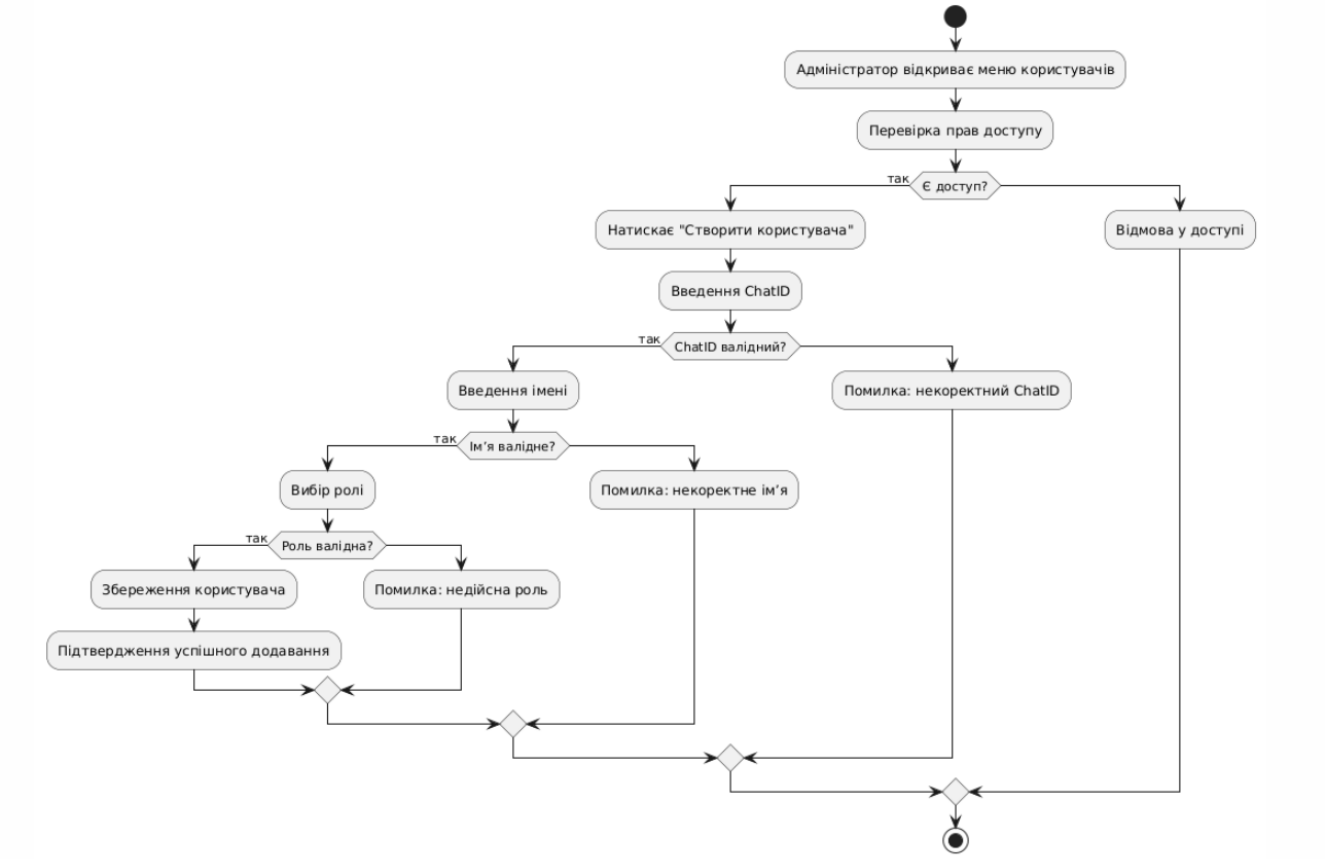

Рисунок А.3 - Діаграма активності процесу додавання нового користувача Telegram-бота

Діаграма активності обробки текстової команди Telegram-ботом

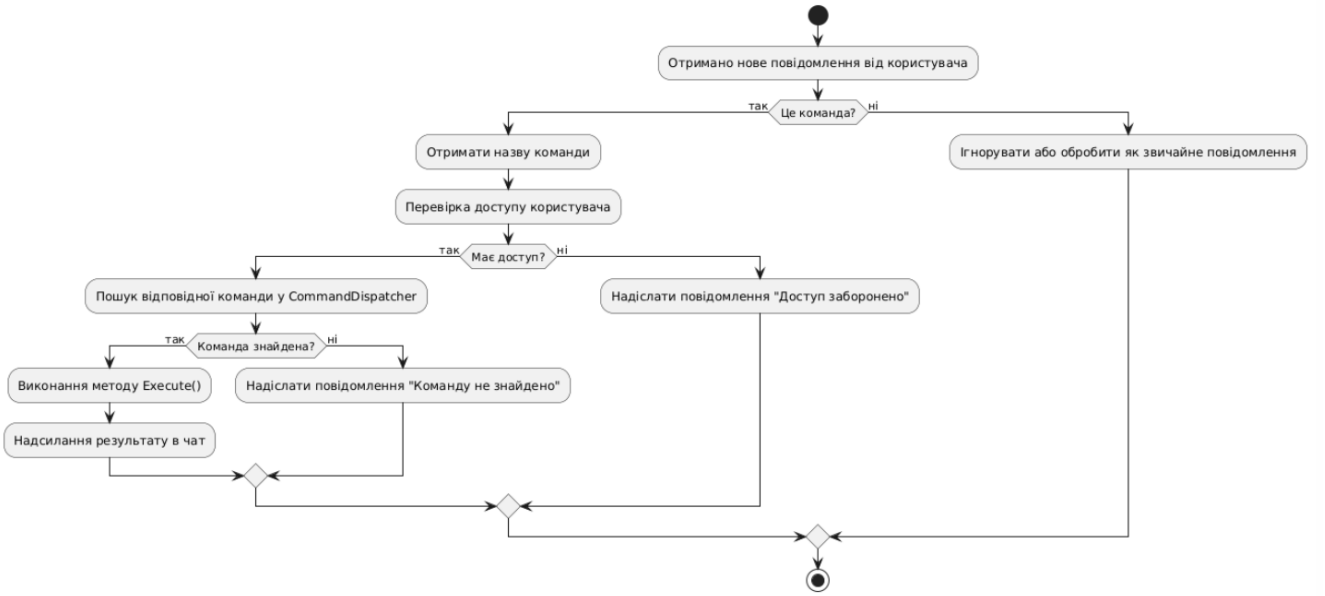

Рисунок А.4 – Діаграма активності обробки текстової команди Telegram-ботом

Додаток Б (обов'язковий). ЛІСТИНГ ВИХІДНОГО КОДУ ОСНОВНИХ КЛАСІВ СИСТЕМИ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using GTANetworkAPI;
using NeptuneEvo.AdminTGBot.Handlers;
using NeptuneEvo.AdminTGBot.Utils;
using Redage.SDK;
using Telegram.Bot;
using Telegram.Bot.Polling;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using Telegram.Bot.Types.ReplyMarkups;
using User = NeptuneEvo.AdminTGBot.AccessControl.Models.User;

namespace NeptuneEvo.AdminTGBot
{
   public static class BotSystem : Script
   {
      private static readonly cLog Log = new cLog(new nLog("TG_BOT"));

      private static readonly string
         BotToken = Main.ServerSettings.TelegramBotToken;

      private static readonly TelegramBotClient BotClient = new TelegramBotClient(BotToken);

      private static readonly CancellationTokenSource Cts = new CancellationTokenSource();



      public static List<User> Users = new List<User>();



      [ServerEvent(Event.ResourceStart)]
      public static void OnResourceStart()
      {
         SendMessageAsync(Templ.GetText(Msg.ServerStarting));

         CommandDispatcher.RegisterAll();

         NAPI.Task.Run(async () =>
```

```
        {
            await SendMessageAsync(Templ.GetText(Msg.ServerReady));
            StartReceiving();
        });
    }
    private static async Task GetUpdateHandler(ITelegramBotClient botClient, Update update,
        CancellationToken cancellationToken)
    {
        if (update.Type == UpdateType.Message && update.Message?.Text != null)
            await MessageHandler.HandleTextMessage(update.Message);
        else if (update.Type == UpdateType.CallbackQuery && update.CallbackQuery != null)
            await CallbackHandler.HandleCallbackQuery(update.CallbackQuery);
        else
            throw new ArgumentOutOfRangeException();
    }

    public static void StartReceiving()
    {
        Task.Run(() =>
        {
            BotClient.StartReceiving(
                updateHandler: GetUpdateHandler,
                pollingErrorHandler: ErrorHandler,
                receiverOptions: new ReceiverOptions(),
                cancellationToken: Cts.Token
            );
        });
    }

    public static void StopReceiving()
    {
        Cts.Cancel();
    }
        public static async Task SendMessageToUserAsync(long chatId, string message,
IReplyMarkup? replyMarkup = null)
    {
        try
        {
            await BotClient.SendTextMessageAsync(chatId, message, parseMode: ParseMode.Html,
                replyMarkup: replyMarkup);
        }
        catch (Exception ex)
        {
            Log.Write($"Помилка надсилання повідомлення в Telegram: {ex.Message}");
        }
    }
```

```
    private static Task ErrorHandler(ITelegramBotClient botClient, Exception exception,
        CancellationToken cancellationToken)
    {
        Log.Write($"Помилка Telegram API: {exception.Message}");
        return Task.CompletedTask;
    }

    public static async Task SendMenuAsync(long chatId, string message, ReplyKeyboardMarkup
keyboard)
    {
        // var keyboard = KeyboardBuilder.BuildMainMenuKeyboard();

        try
        {
            await BotClient.SendTextMessageAsync(chatId, message, replyMarkup: keyboard,
parseMode: ParseMode.Html);
        }
        catch (Exception ex)
        {
            Log.Write($"Помилка надсилання меню в Telegram: {ex.Message}");
        }
    }

    public static async Task SendMessageAsync(string message)
    {
        try
        {
            foreach (var user in Users)
            {
                await BotClient.SendTextMessageAsync(user.ChatId, message);
            }
        }
        catch (Exception ex)
        {
            Log.Write($"Помилка надсилання повідомлення в Telegram: {ex.Message}");
        }
    }

    public static async Task SendMessageToUserAsync(long chatId, string message)
    {
        try
        {
            await BotClient.SendTextMessageAsync(chatId, message, parseMode: ParseMode.Html);
        }
        catch (Exception ex)
```

```
        {
            Log.Write($"Помилка надсилання повідомлення в Telegram: {ex.Message}");
        }
    }

    public static async Task SendMessageWithKeyboardToUserAsync(long chatId, string message,
        InlineKeyboardMarkup keyboard)
    {
        try
        {
            await BotClient.SendTextMessageAsync(chatId, message, replyMarkup: keyboard,
parseMode: ParseMode.Html);
        }
        catch (Exception ex)
        {
            Log.Write($"Помилка надсилання повідомлення з кнопками в Telegram:
{ex.Message}");
        }
    }


    public static async Task EditMessageAsync(long chatId, int messageId, string newText)
    {
        try
        {
            await BotClient.EditMessageTextAsync(
                chatId: chatId,
                messageId: messageId,
                text: newText,
                parseMode: ParseMode.Html,
                replyMarkup: null
            );
        }
        catch (Exception ex)
        {
            Log.Write($"Помилка редагування повідомлення: {ex.Message}");
        }
    }

    public static async Task AnswerCallbackQueryAsync(string callbackQueryId, string text)
    {
        await BotClient.AnswerCallbackQueryAsync(callbackQueryId, text);
    }

    public static async Task<bool> IsAllowedChatId(long chatId)
    {
```

```
            var admin = Users.FirstOrDefault(u => u.ChatId == chatId);

            if (admin != null) return true;
            await SendMessageToUserAsync(chatId, Templ.GetText(Msg.CallbackAccessDenied));
            return false;

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Threading.Tasks;
using GTANetworkAPI;
using NeptuneEvo.AdminTGBot.AccessControl;
using NeptuneEvo.AdminTGBot.Commands;
using NeptuneEvo.Character;
using NeptuneEvo.Core;
using NeptuneEvo.Handles;
using NeptuneEvo.Players;
using Telegram.Bot.Types.ReplyMarkups;

namespace NeptuneEvo.AdminTGBot
{
   public class CommandDispatcher : Script
   {
      private static readonly Dictionary<Type, ICommand> _instances = new Dictionary<Type,
ICommand>();

      public static readonly Dictionary<string, Func<string[], long, Task>> CommandHandlers =
         new Dictionary<string, Func<string[], long, Task>>();

      public static readonly Dictionary<long, string> PendingCommands = new Dictionary<long,
string>();


      private static void RegisterCommand(ICommand cmd)
      {
         CommandHandlers[cmd.Command] = cmd.Execute;
      }

      public static void RegisterAll()
      {
         var commandTypes = Assembly.GetExecutingAssembly()
            .GetTypes()
            .Where(t => typeof(ICommand).IsAssignableFrom(t) && !t.IsInterface &&
!t.IsAbstract);

         foreach (var type in commandTypes)
         {
            if (!_instances.TryGetValue(type, out var instance))
               instance = (ICommand)Activator.CreateInstance(type)!;
               _instances[type] = instance;
            }
```

```
            RegisterCommand(instance);
        }
    }

    public static IEnumerable<ICommand> GetAllCommands(){
        return _instances.Values;
    }

    public static T Get<T>() where T : class, ICommand
        var type = typeof(T);
        if (_instances.TryGetValue(type, out var instance))
            return instance as T;

        throw new InvalidOperationException($"Команду {type.Name} не зареєстровано.");
    }

    private static bool TryGetCommand(string commandName, out ICommand cmd)
    {
        cmd = _instances.Values.FirstOrDefault(c => c.Command.Equals(commandName,
StringComparison.OrdinalIgnoreCase));
        return cmd != null;
    }

    public static async Task<bool> CheckCommand(string commandName, long chatId)
    {
        if (!TryGetCommand(commandName, out var cmd))
        {
            await BotSystem.SendMessageToUserAsync(chatId, "Невідома команда.");
            return false;
        }

        if (!BotDao.IsAllowed(chatId, cmd.RequiredRole))
        {
            await BotSystem.SendMessageToUserAsync(chatId, "Недостатньо прав для цієї коман-
ди.");
            return false;
        }

        return true;
    }

  }
}
```

```
using System;
using System.Threading.Tasks;
using NeptuneEvo.AdminTGBot.AccessControl.Models;
using NeptuneEvo.AdminTGBot.Utils;

namespace NeptuneEvo.AdminTGBot.Commands
{
   public interface ICommand
   {
      public string Command => "none";

      public Role RequiredRole => Role.None;

      public MenuType MenuType => MenuType.None;

      public Func<string[], long, Task> Handler => (args, chatId) => Task.Run(() => Execute(args,
chatId));

      public async Task Execute(string[] args, long chatId)
      {
         await BotSystem.SendMessageToUserAsync(chatId,
            Templ.GetText(Msg.InvalidCommandInput));
      }


   }
}


using System.Threading.Tasks;

namespace NeptuneEvo.AdminTGBot.Commands
{
   public interface IConfirmableCommand
   {
      Task OnConfirm(string payload, long chatId, int messageId);
   }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using GTANetworkAPI;
using Localization;
using NeptuneEvo.AdminTGBot.AccessControl.Models;
using NeptuneEvo.AdminTGBot.Utils;
using NeptuneEvo.Core;
using Redage.SDK;
using Telegram.Bot;
using Telegram.Bot.Polling;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using Telegram.Bot.Types.ReplyMarkups;

namespace NeptuneEvo.AdminTGBot.Handlers
{
    public class MessageHandler
    {
        private static readonly Dictionary<long, string> PendingCommands =
CommandDispatcher.PendingCommands;

        private static readonly Dictionary<string, Func<string[], long, Task>> CommandHandlers =
            CommandDispatcher.CommandHandlers;

        private static readonly Dictionary<string, string> InputCommandHints = new
Dictionary<string, string>
        {
            [Templ.GetText(Msg.KickCMD)] = "✍ Введіть Id гравця, якого збираєтесь кікнути:",
            [Templ.GetText(Msg.BanCMD)] = "✍ Введіть Id гравця, якого збираєтесь забанити:",
            [Templ.GetText(Msg.HardBanCMD)] = "✍ Введіть Id гравця, якого збираєтесь хардбанити:",
            [Templ.GetText(Msg.GlobalCMD)] = "✍ Введіть текст глобального повідомлення:"
        };


        public static async Task HandleTextMessage(Message message)
        {
            var chatId = message.Chat.Id;
            var text = message.Text.Trim();

            if (!await BotSystem.IsAllowedChatId(chatId)) return;
```

```
        if (PendingCommands.ContainsKey(chatId))
        {
            await HandlePendingCommand(chatId, text);
            return;
        }

        string command = text.StartsWith("/") ? text[1..].ToLower() : "";

        if (!await CommandDispatcher.CheckCommand(command, chatId)) return;

        if (InputCommandHints.TryGetValue(command, out var hint))
        {
            PendingCommands[chatId] = command;
            await BotSystem.SendMessageToUserAsync(chatId, hint,
KeyboardBuilder.CancelButton);
        }
        else
        {
            await CommandHandlers[command].Invoke(new[] { "" }, chatId);
            await BotSystem.SendMenuAsync(chatId, Templ.GetText(Msg.CommandExecuted),
                KeyboardBuilder.BuildMenuKeyboard(MenuType.MainMenu));
        }
    }

    private static async Task HandlePendingCommand(long chatId, string input)
    {
        if (input == Templ.GetText(Msg.Cancel))
        {
            PendingCommands.Remove(chatId);
            await BotSystem.SendMenuAsync(chatId, Templ.GetText(Msg.CommandCancelled),
                KeyboardBuilder.BuildMenuKeyboard(MenuType.MainMenu));
            return;
        }

        if (input.StartsWith("/"))
        {
            PendingCommands.Remove(chatId);
            return;
        }

        if (PendingCommands.TryGetValue(chatId, out var command))
        {
            PendingCommands.Remove(chatId);
            await CommandHandlers[command].Invoke(new[] { input }, chatId);
            await BotSystem.SendMenuAsync(chatId, Templ.GetText(Msg.CommandExecuted),
                KeyboardBuilder.BuildMenuKeyboard(MenuType.MainMenu));
```

```
        }
      }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using GTANetworkAPI;
using Localization;
using NeptuneEvo.AdminTGBot.Commands;
using NeptuneEvo.AdminTGBot.Utils;
using NeptuneEvo.Core;
using Redage.SDK;
using Telegram.Bot.Types;
using Task = System.Threading.Tasks.Task;

namespace NeptuneEvo.AdminTGBot.Handlers
{
    public class CallbackHandler
    {
        private static Dictionary<long, string> PendingCommands =
CommandDispatcher.PendingCommands;

        public static async Task HandleCallbackQuery(CallbackQuery callbackQuery)
        {
            var chatId = callbackQuery.Message.Chat.Id;

            if (!ValidateCallback(callbackQuery, out var errorMessage, out var action, out var userId))
            {
                await BotSystem.AnswerCallbackQueryAsync(callbackQuery.Id, errorMessage);
                return;
            }

            if (!await BotSystem.IsAllowedChatId(chatId))
                return;

            if (userId != chatId)
            {
                await BotSystem.AnswerCallbackQueryAsync(callbackQuery.Id,
                    Templ.GetText(Msg.CallbackConfirmDenied));
                return;
            }

            if (!PendingCommands.TryGetValue(chatId, out var rawData))
            {
```

```
            await BotSystem.AnswerCallbackQueryAsync(callbackQuery.Id,
                Templ.GetText(Msg.CallbackNotFound));
            return;
        }

        switch (action)
        {
            case "confirm":
                await ProcessConfirmedAction(chatId, callbackQuery.Message.MessageId, rawData);
                break;
            case "cancel":
                await CancelCallback(callbackQuery);
                break;
        }

        PendingCommands.Remove(chatId);
    }

    private static bool ValidateCallback(CallbackQuery query, out string errorMessage, out string
action, out long id)
    {
        errorMessage = string.Empty;
        action = string.Empty;
        id = 0;

        if (string.IsNullOrEmpty(query.Data))
        {
            errorMessage = Templ.GetText(Msg.CallbackError);
            return false;
        }

        var parts = query.Data.Split(':');
        if (parts.Length < 2)
        {
            errorMessage = Templ.GetText(Msg.CallbackFormatError);
            return false;
        }

        action = parts[0];

        if (parts.Length > 2 && long.TryParse(parts[2], out var parsedId))
            id = parsedId;
        else
            id = query.Message.Chat.Id;

        return true;
```

```
        }

        private static async Task ProcessConfirmedAction(long chatId, int messageId, string rawData)
        {
            var split = rawData.Split(':', 2);
            if (split.Length != 2)
            {
                await BotSystem.SendMessageToUserAsync(chatId, "Невірні дані підтвердження.");
                return;
            }

            var commandName = split[0];
            var payload = split[1];

            var command = CommandDispatcher.GetAllCommands()
                .FirstOrDefault(c => c.Command == commandName);

            var confirmableCommand = command as IConfirmableCommand;

            if (confirmableCommand == null)
            {
                await BotSystem.SendMessageToUserAsync(chatId,
                    $"Команда `{commandName}` не підтримує підтвердження.");
                return;
            }

            await confirmableCommand.OnConfirm(payload, chatId, messageId);

        }


        private static async Task CancelCallback(CallbackQuery query)
        {
            await BotSystem.EditMessageAsync(query.Message.Chat.Id, query.Message.MessageId,
                Templ.GetText(Msg.CallbackSentCanceled));
        }
    }
}
```

```
using NeptuneEvo.AdminTGBot.AccessControl;
using NeptuneEvo.AdminTGBot.AccessControl.Models;
using NeptuneEvo.AdminTGBot.Utils;
using Redage.SDK;

namespace NeptuneEvo.AdminTGBot.Services
{
   public static class BotService
   {
      private static cLog Log = new cLog(new nLog("TG_Bot | AdminService"));

      public static bool CreateUser(long adminId, long chatId, string name, Role role, out string
response)
      {
         var admin = BotDao.GetUser(adminId);

         if (admin == null)
         {
            response = "Адміна не знайдено.";
            Log.Write(response);
            return false;
         }

         if (!BotDao.IsAllowed(adminId, Role.ProjectTeam))
         {
            response = "Недостатньо прав.";
            Log.Write(response);
            return false;
         }

         if (BotDao.UserExists(chatId))
         {
            response = "Користувач вже існує.";
            Log.Write(response);
            return false;
         }

         BotDao.AddUser(chatId, name, role);
         response = "Користувача успішно створено.";
         Log.Write(response);
         return true;
      }

      public static bool RemoveUser(long adminId, long targetChatId, out string response)
      {
         var admin = BotDao.GetUser(adminId);
```

```
            if (admin == null)
            {
                response = "Адміністратора не знайдено.";
                return false;
            }

            if (!BotDao.UserExists(targetChatId))
            {
                response = "Користувача не знайдено.";
                return false;
            }

            BotDao.RemoveUser(targetChatId);
            response = $"Користувача {targetChatId} видалено.";
            Log.Write(response);
            return true;
        }
    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using NeptuneEvo.AdminTGBot.AccessControl.Models;

namespace NeptuneEvo.AdminTGBot.AccessControl
{
   public static class BotDao
   {
      public static bool IsAllowed(long chatId, Role requiredRole)
      {
         var user = GetUser(chatId);
         return user != null && user.Role >= requiredRole;
      }

      public static Role GetUserRole(long chatId)
      {
         var user = GetUser(chatId);
         return user?.Role ?? Role.None;
      }

      public static void SetUserRole(long chatId, Role newRole)
      {
         var user = GetUser(chatId);
         if (user != null)
            user.Role = newRole;
      }

      public static void AddUser(long chatId, string name, Role role)
      {
         BotSystem.Users.Add(new User(chatId, name, role));
      }

      public static void RemoveUser(long chatId)
      {
         var user = GetUser(chatId);
         if (user != null)
            BotSystem.Users.Remove(user);
      }

      public static List<User> GetAllUsers()
      {
         return BotSystem.Users;
      }

      public static bool UserExists(long chatId)
      {
```

```
            return GetUser(chatId) != null;
        }

        public static User GetUser(long chatId)
        {
            return BotSystem.Users.FirstOrDefault(u => u.ChatId == chatId);
        }
    }
}
```

```
using System.Collections.Generic;
using System.Linq;
using NeptuneEvo.AdminTGBot.AccessControl.Models;
using Telegram.Bot.Types.ReplyMarkups;

namespace NeptuneEvo.AdminTGBot.Utils
{
   public class KeyboardBuilder
   {

      public static ReplyKeyboardMarkup BuildMenuKeyboard(MenuType menu)
      {
         const int buttonsPerRow = 3;

         var allCommands = CommandDispatcher.GetAllCommands()
            .OrderBy(c => c.Command)
            .Where(c => c.MenuType == menu)
            .Select(c => new KeyboardButton("/" + c.Command))
            .ToList();

         var rows = new List<KeyboardButton[]>();

         for (int i = 0; i < allCommands.Count; i += buttonsPerRow)
         {
            var row = allCommands.Skip(i).Take(buttonsPerRow).ToArray();
            rows.Add(row);
         }

         return new ReplyKeyboardMarkup(rows)
         {
            ResizeKeyboard = true,
            OneTimeKeyboard = false
         };
      }



      public static ReplyKeyboardMarkup CancelButton => new ReplyKeyboardMarkup(new[]
      {
         new[] { new KeyboardButton(Templ.GetText(Msg.Cancel)) }
      })
      {
         ResizeKeyboard = true,
         OneTimeKeyboard = true
      };
```

```
        public static InlineKeyboardMarkup BuildConfirmationKeyboard(string callbackId)
        {
            return new InlineKeyboardMarkup(new[]
            {
                new[]
                {
                    InlineKeyboardButton.WithCallbackData(Templ.GetText(Msg.Yes),
$"confirm:{callbackId}"),
                },
                new[]
                {
                    InlineKeyboardButton.WithCallbackData(Templ.GetText(Msg.No),
$"cancel:{callbackId}")
                }
            });
        }
    }
}
```

Додаток В (обов'язковий).

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка клієнт-серверної системи адміністрування ігрового контенту з інтеграцією в Telegram (серверна частина)

Тип роботи: бакалаврська кваліфікаційна робота

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра АІІТ, ФІІТА, ІІСТ-21б

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 94 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☑ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Олег БІСІКАЛО, зав. кафедри АІІТ _____ (підпис)

(прізвище, ініціали, посада)

Любов ЯНЧУК, секретар кафедри АІІТ _____ (підпис)

(прізвище, ініціали, посада)

Особа, відповідальна за перевірку _____ (підпис) Костянтин ОВЧИННИКОВ

(прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ (підпис) к.т.н., доцент Коцюбинський В.Ю.

(прізвище, ініціали, посада)

Здобувач _____ (підпис) Гончар С.В.

(прізвище, ініціали)