НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В. о. завідувача кафедри ОТ
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою "Комп'ютерні системи та мережі"

спеціальності 123 "Комп'ютерна інженерія"

на тему: Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері

Виконав : студент __4__ курсу, групи _ІО-11
(шифр групи)

_____Домашенко Іван Сергійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Пономаренко А. М._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент кафедри ОТ Пономаренко А.М.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __доц., к.т.н., доцент кафедри Коган Алла Вікторівна_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

"Комп'ютерні системи та мережі"

спеціальності 123 "Комп'ютерна інженерія"

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ОТ
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Домашенка Івана Сергійовича

1. Тема проєкту *Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері*
керівник проєкту Пономаренко А. М., асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року No 1705-с

2. Термін здачі студентом закінченого проєкту 10.06.2025

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Розділ 1. Огляд існуючих рішень та технологій розробки інформаційно довідкових сервісів

Розділ 2. Огляд технологій та інструментів для розробки системи

Розділ 3. Деталі розробки системи.

Розділ 4. Дослідження та аналіз розробленої системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Пономаренко А. М.		

7. Дата видачі завдання «15» грудня 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
	Затвердження теми проєкту	*10.12.2024-15.12.2024*	
	Вивчення та аналіз завдання	*15.12.2024-14.04.2025*	
.	*Розробка архітектури та загальної структури системи*	*14.04.2025-28.04.2025*	
.	*Розробка структур окремих підсистем*	*28.04.2025-05.05.2025*	
	Програмна реалізація системи	*5.05.2025-12.05.2025*	
	Оформлення пояснювальної записки	*12.05.2025-19.05.2025*	
	Захист програмного продукту	02.06.2025	
	Передзахист	09.06.2025	
	Захист	18.06.2025	

Студент-дипломник _____ Іван ДОМАШЕНКО
(підпис)

Керівник проєкту _____ Артем ПОНОМАРЕНКО
(підпис)

АНОТАЦІЯ

У даній бакалаврській дипломній роботі представлено процес розробки інформаційно-довідкового сервісу для відеогри "Genshin Impact" під назвою "Помічничок мандрівника", спрямованого на надання користувачам необхідної інформації та інструментів, пов'язаних із цією популярною відеогрою. Здійснено аналіз основних потреб цільової аудиторії та розроблено функціональність бота, що охоплює перегляд даних про персонажів, зброю, артефакти, відомостей щодо щоденних ігрових подій та ресурсів, актуальних ігрових банерів, а також додатковий розважальний контент.

В рамках проєкту було реалізовано інформаційно-довідковий сервіс для відеогри з використанням мови програмування Python та бібліотеки python-telegram-bot для ефективної взаємодії з Telegram Bot API. Детально описано ключові аспекти розробки, включаючи обробку текстових команд і натискань кнопок, механізми навігації по меню, реалізацію функцій відображення інформації та застосування локалізації для забезпечення підтримки різних мов. Створений бот має на меті забезпечити зручний, оперативний та актуальний доступ до інформації для гравців Genshin Impact.

ANNOTATION

This bachelor's thesis presents the development process of an Information and Reference Service for the video game "Genshin Impact," entitled "Traveler's Helper," aimed at providing users with the necessary information and tools related to this popular video game. An analysis of the core needs of the target audience was conducted, and the bot's functionality was developed to encompass browsing data about characters, weapons, artifacts, information on daily in-game events and resources, current game banners, as well as additional entertaining content.

Within the project, an Information and Reference Service for the video game was implemented using the Python programming language and the python-telegram-bot library for effective interaction with the Telegram Bot API. Key aspects of the development are detailed, including the processing of text commands and button presses, menu navigation mechanisms, the implementation of information display functions, and the application of localization to ensure support for different languages. The created bot aims to provide convenient, timely, and up-to-date access to information for Genshin Impact players.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екзем	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Інформаційно-довідковий сервіс для	3		
			відеогри з інтерфейсом у месенджері			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Інформаційно-довідковий сервіс для	115		
			відеогри з інтерфейсом у месенджері			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Інформаційно-довідковий сервіс для	1		
			відеогри з інтерфейсом у месенджері			
			Схема внутрішньої архітектури			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Інформаційно-довідковий сервіс для	1		
			відеогри з інтерфейсом у месенджері			
			Схема сценаріїв використання			
			системи			
	А4	***ІАЛЦ.467200.006 Д3***	Інформаційно-довідковий сервіс для	1		
			відеогри з інтерфейсом у месенджері			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Інформаційно-довідковий сервіс для	67		
			відеогри з інтерфейсом у месенджері			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	
Розроб		Домашенко І.С.			Інформаційно-довідковий сервіс для відеогри з інтрефейсом у месенджері Опис альбому
Перев		Пономаренко А.М.			

Літ.	Аркуш	Аркушів
		1

НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері*»

Київ – 2025

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 2

ПІДСТАВИ ДЛЯ РОЗРОБКИ 2

МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ 2

ДЖЕРЕЛА РОЗРОБКИ 2

ТЕХНІЧНІ ВИМОГИ 2

ЕТАПИ РОЗРОБКИ 3

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку інтерактивного Telegram-бота, призначеного для надання інформації гравцям популярної відеогри Genshin Impact. Областю застосування цієї системи є надання швидкого та зручного доступу до інформації про ігровий контент безпосередньо в месенджері Telegram. Розроблена система може бути корисна як для новачків, які тільки починають знайомство з грою, так і для досвідчених гравців, які шукають конкретну інформацію або бажають оптимізувати свій ігровий процес.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп’ютерної інженерії», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної роботи є розробка Telegram-бота, який надаватиме гравцям Genshin Impact актуальну інформацію про ігрові об'єкти, зокрема персонажів та зброю та ін.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційна інформація про гру Genshin Impact, публічні бази даних та вікі-ресурси, присвячені грі, документація до бібліотеки Python Telegram Bot, науково-технічна література з розробки чат-ботів та веб-сервісів.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту:

- Простий та інтуїтивно-зрозумілий інтерфейс користувача на базі Telegram.

- Забезпечення можливості вибору мови інтерфейсу (українська та, можливо, інші).
- Надання користувачам можливості вибору категорії інформації (персонажі, зброя).
- Реалізація навігації між меню за допомогою кнопок "Назад".
- Надання вичерпної та зрозумілої документації для розробленого продукту.

5.2. Вимоги до програмного забезпечення:

- Операційна система: Windows, Linux або macOS.
- Середовище розробки: Будь-яке зручне IDE для Python (наприклад, PyCharm, VS Code).
- Мова програмування: Python 3.x.

5.3. Вимоги до апаратної частини:

- Комп'ютер, здатний запускати середовище розробки Python та підтримувати підключення до мережі Інтернет.
- Для роботи бота в режимі реального часу: сервер або віртуальний приватний сервер (VPS) з постійним підключенням до Інтернету (хоча для демонстрації може бути використано локальний запуск).

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2024-15.12.2024
Вивчення та аналіз завдання	15.12.2024-14.04.2025
Розробка архітектури та загальної структури системи	14.04.2025-28.04.2025
Розробка структур окремих частин системи	28.04.2025-05.05-2025
Програмна реалізація системи	05.05.2025-12.05.2025
Виправлення помилок	12.05.2025-19.05.2025
Оформлення пояснювальної записки	19.05.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері*»

Київ – 2025

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ІНФОРМАЦІЙНО-ДОВІДКОВИХ СЕРВІСІВ 7

1.1 Огляд існуючих чат-ботів для ігор 7

1.1.1 Аналіз функціональності ботів 7

1.1.2 Технологічна база ігрових чат-ботів 9

1.2 Порівняльний аналіз існуючих рішень для Інформаційно-довідкових сервісів 10

1.2.1 Веб-сайти та вікі-ресурси 11

1.2.2 Мобільні додатки 12

1.2.3 Чат-боти 15

ВИСНОВОК ДО РОЗДІЛУ 1 18

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СИСТЕМИ 20

2.1 Мова програмування Python 20

2.1.1 Загальна характеристика мови Python 20

2.1.2 Обґрунтування вибору мови Python 22

2.2.2 Бібліотека telegram.ext 25

2.3 Бібліотеки для асинхронного програмування 26

2.4 Бібліотеки для роботи з даними та файлами 28

2.4.1 Бібліотека json 29

2.4.2 Бібліотека os 29

2.4.3 Бібліотека collections 30

2.5 Бібліотеки для роботи з розкладом 31

2.6 Бібліотеки для роботи з часом 33
2.7 Бібліотеки для локалізації 35
2.8 Інші використані бібліотеки 36
2.8.1 Бібліотека sys 36
Бібліотека sys надає доступ до деяких змінних та функцій, які тісно взаємодіють з інтерпретатором Python та його оточенням 36
2.8.2 Бібліотека random 37
ВИСНОВОК ДО РОЗДІЛУ 2 39
РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ 41
3.1 Обробник команди /start 41
3.2 Розробка основних обробників бота 44
3.2.1 Обробка персонажів 44
3.2.2 Обробка інформації про зброю 50
3.2.3 Обробка інформації про артефакти 57
3.2.4 Обробка інформації про ворогів 63
3.2.5 Обробка меню "Цікавинки" 71
3.2.6 Обробка меню "Що сьогодні" 81
3.3 Обробка повідомлень користувача handle_message 89
ВИСНОВОК ДО РОЗДІЛУ 3 96
РОЗДІЛ 4. ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ. 98
4.1 Методологія тестування 98
4.2 Функціональне тестування за розділами меню 98
4.2.1 Тестування головного меню та загальних команд 98
4.2.2 Тестування розділу "Персонажі" 101

4.2.3 Тестування розділу "Зброя" 103
4.2.4 Тестування розділу "Артефакти" 105
4.2.5 Тестування розділу "Що сьогодні" 106
4.2.6 Тестування розділу "Цікавинки" 109
4.2.7 Тестування розділу "Битва" 112
4.3 Тестування локалізації 114
4.4 Тестування зручності використання (Usability Testing) 114
4.5 Тестування надійності та обробки помилок 115
ВИСНОВОК ДО РОЗДІЛУ 4 117
ВИСНОВКИ 119
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 121

ВСТУП

Сфера онлайн-розваг переживає безпрецедентне зростання, залучаючи до віртуальних світів мільйони ентузіастів по всьому світу. Особливе місце серед жанрового різноманіття займають пригодницькі рольові ігри з відкритим світом, які пропонують гравцям досліджувати великі та деталізовані ігрові простори, взаємодіяти з різноманітними персонажами, виконувати квести та розвивати своїх героїв. Однією з таких ігор, що здобула величезну популярність та визнання, є *Genshin Impact*. Її складний ігровий світ, розгалужена система персонажів, зброї та механік створюють високий попит на зручні й швидкі інструменти доступу до інформації, здатної допомогти гравцям у їхніх віртуальних подорожах.

В епоху стрімкого розвитку цифрових технологій та зростання популярності онлайн-ігор, зокрема всесвітньо відомої *Genshin Impact*, особливо актуальним стає впровадження інтелектуальних помічників, здатних оперативно надавати довідкову інформацію. У цьому контексті чат-боти — інтелектуальні агенти, що взаємодіють з користувачами у текстовому чи голосовому форматі, — набувають особливої цінності. Вони вже зарекомендували себе як ефективний інструмент у сферах обслуговування клієнтів, освіти, торгівлі, й ігрова індустрія не є винятком. Для гравців чат-боти можуть стати незамінними асистентами, що надають швидкий доступ до інформації про персонажів, спорядження, ігрові події, а також допомагають знайти спільноти однодумців.

Актуальність цієї роботи обумовлена прагненням гравців мати під рукою інтерактивного помічника, який би надавав інформацію без потреби звертатися до веб-браузерів чи сторонніх додатків. Метою дослідження є розробка Telegram-бота, що забезпечує миттєвий доступ до ключових ігрових даних — про героїв, зброю, артефакти та інші елементи гри.

Об’єктом дослідження є процес розробки та функціонування чат-бота для *Genshin Impact*, а предметом – методи та засоби надання інформаційної

підтримки гравцям. У процесі реалізації було застосовано методи аналізу наявних рішень, проектування інтерфейсу, розробки програмного забезпечення мовою Python з використанням бібліотеки python-telegram-bot, а також тестування створеної системи.

Практична цінність результатів полягає у створенні корисного інструменту, який покликаний оптимізувати ігровий процес і покращити загальні враження від гри. Розроблений бот може бути впроваджений у Telegram та вільно використовуватись широким колом користувачів.

Структура роботи охоплює огляд наявних рішень і технологій створення чат-ботів у сфері ігрової інформації, опис архітектури та проектування бота, детальний виклад процесу його реалізації, а також аналіз результатів тестування з рекомендаціями щодо подальшого вдосконалення.

РОЗДІЛ 1.
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ІНФОРМАЦІЙНО-ДОВІДКОВИХ СЕРВІСІВ

1.1 Огляд існуючих чат-ботів для ігор

Чат-боти вже давно перестали бути новинкою — сьогодні вони стали невіддільною частиною багатьох сфер нашого життя: від банківських послуг і технічної підтримки до онлайн-магазинів та освітніх платформ. Не стала винятком і ігрова індустрія, де боти активно використовуються для підвищення зручності взаємодії з ігровими системами. Їхнє застосування в іграх є надзвичайно широким: від надання довідкової інформації про ігрові об'єкти до автоматизації рутинних дій, як-от відстеження ресурсів, планування прокачки персонажів або навіть нагадувань про щоденні завдання. Такі інструменти дозволяють гравцям зосередитися на найцікавіших аспектах гри, делегуючи пошук і структурування інформації зручному цифровому помічнику.

1.1.1 Аналіз функціональності ботів

Функціональність ігрових чат-ботів може значно відрізнятися залежно від жанру гри, цільової аудиторії та конкретних потреб ігрової спільноти. Такі боти здатні виконувати широкий спектр завдань, від простого надання довідкової інформації до реалізації складних систем підтримки гравців у реальному часі. Найпоширеніші функції, які реалізуються в ігрових ботах, включають:

Довідкова інформація

Це одна з базових функцій, яку реалізують більшість ігрових ботів. Вона передбачає доступ до детальної інформації про персонажів (характеристики, навички, історія, способи отримання), ігрові предмети (опис, властивості, рідкість, місця знаходження), локації, завдання (квести) та інший важливий контент. Така функціональність особливо корисна для новачків, які тільки починають грати, а також для досвідчених гравців, які потребують швидкого

доступу до точної інформації без необхідності самостійно шукати її в інтернеті.

Ігрові інструменти

До цієї категорії відносяться різні утиліти, які допомагають гравцеві оптимізувати ігровий процес. Це можуть бути калькулятори для розрахунку шкоди або ресурсів, необхідних для прокачування персонажів, планувальники розвитку (наприклад, побудова шляху покращення персонажа або зброї), трекери прогресу для ведення обліку виконаних завдань чи досягнень. Такі інструменти особливо цінні для ігор зі складною механікою та великою кількістю ігрових об’єктів, як-от Genshin Impact.

Соціальна взаємодія

Багато чат-ботів підтримують функції, пов’язані з комунікацією між гравцями. Наприклад, пошук союзників для спільного проходження підземель, обмін повідомленнями, організація івентів, турнірів або внутрішньоігрових подій. Це дозволяє формувати активну спільноту прямо всередині месенджера, де працює бот.

Сповіщення

Боти можуть автоматично надсилати гравцеві повідомлення про актуальні події в грі: оновлення, нові банери персонажів або зброї, тимчасові івенти, досягнення в грі, відновлення енергії, доступність щоденних винагород тощо. Це допомагає гравцям залишатися в курсі важливих подій, навіть коли вони не перебувають у грі.

Автоматизація дій

У межах дозволених ігровими правилами функцій, боти можуть автоматизувати деякі рутинні дії, як-от планування щоденних завдань, нагадування про збори ресурсів, перевірку стану персонажів або енергії. Хоча повна автоматизація гри зазвичай заборонена (і може бути розцінена як порушення умов користування), допоміжна автоматизація — цілком легальний і зручний інструмент для гравців.

1.1.2 Технологічна база ігрових чат-ботів

Створення ефективного чат-бота вимагає використання сучасних технологій, здатних забезпечити швидку реакцію на запити, підтримку структурованого діалогу, обробку запитів природною мовою та взаємодію з ігровими API або базами даних. Основні підходи включають:

1. Скриптові боти: Це найпростіші типи ботів, логіка яких базується на чітко прописаних сценаріях і командних шаблонах. Вони реагують на фіксовані запити, наприклад */character Xiangling* або *!help*. Хоча такі боти не розуміють контекст або природну мову, їх легко реалізувати, і вони підходять для базового функціоналу. Розроблений у даній роботі "Помічничок мандрівника Genshin Impact" на початковому етапі використовує переважно скриптовий підхід, де навігація здійснюється через структуровані меню та натискання кнопок. Користувач взаємодіє з ботом, обираючи послідовно тип зброї, а потім її рідкість, щоб отримати відповідний список. Команди як такі не використовуються, а вся взаємодія відбувається через інтерактивні клавіатури.
2. Боти на основі машинного навчання (ML) та обробки природної мови (NLP): Це більш просунуті рішення, які здатні "розуміти" запити, сформульовані природною мовою: "Покажи мені мечі 5★" або "Хто з персонажів має елемент Кріо?". Такі боти можуть навчатися з часом, покращуючи свої відповіді на основі поведінки користувачів. Їх розробка є складнішою, але вони забезпечують вищу гнучкість і кращий користувацький досвід. На даному етапі розробки "Помічничок мандрівника Genshin Impact" не використовує технології ML та NLP для обробки природної мови. Взаємодія з ботом повністю базується на структурованих меню та кнопках. Однак, у майбутньому розширення функціональності бота може включати інтеграцію елементів NLP для більш гнучкого та інтуїтивно зрозумілого спілкування з користувачем.

3. Інтеграція з API ігрових платформ: Деякі ігри надають відкриті API, що дозволяє чат-ботам отримувати актуальну інформацію прямо з ігрового сервера. Це може включати дані про гравців, персонажів, інвентар тощо. Хоча не всі ігри, включаючи Genshin Impact, мають офіційні API, деякі спільноти створюють власні неофіційні бази даних, якими також можуть користуватися боти. "Помічничок мандрівника Genshin Impact" наразі не використовує пряму інтеграцію з офіційними чи неофіційними API гри. Інформація про персонажів та зброю зберігається локально у файлах Python у структурованому форматі (словники та списки). Оновлення даних вимагає ручного внесення змін до цих файлів. У майбутньому можлива інтеграція з зовнішніми базами даних або API для автоматичного оновлення інформації.
4. Інтеграція з месенджерами та соціальними мережами: Найбільш поширеною платформою для ігрових ботів є Telegram та Discord, адже саме тут формуються активні геймерські спільноти. Розробники обирають ці платформи через їхню відкритість, наявність API, підтримку кнопок, меню, повідомлень з форматуванням та інші зручні функції для реалізації бота. "Помічничок мандрівника Genshin Impact" розроблено спеціально для платформи Telegram. Використання Telegram Bot API надає можливість створювати зручний інтерфейс на основі кнопок та інтерактивних меню, що спрощує навігацію та вибір потрібної інформації для користувачів.

1.2 Порівняльний аналіз існуючих рішень для Інформаційно-довідкових сервісів

На момент написання цієї роботи, існує розгалужена екосистема сторонніх розробок, покликаних задовольнити інформаційні потреби гравців Genshin Impact. Ці рішення варіюються за формою, функціональністю та зручністю використання, включаючи веб-сайти, мобільні додатки та чат-боти, доступні в різних месенджерах.

1.2.1 Веб-сайти та вікі-ресурси

Ці онлайн-платформи, представлені офіційною HoYoWiki та численними фан-сайтами, такими як Honey Impact, Genshin.gg, а також різноманітними форумами та спільнотами, є безцінними джерелами практично будь-якої інформації про Genshin Impact.

Їхні переваги незаперечні:

- Вичерпність інформації: Ці ресурси часто містять надзвичайно детальну та структуровану інформацію, яка охоплює всі аспекти гри. Гравці можуть знайти тут глибокі аналізи характеристик персонажів, докладні описи навичок, таблиці з параметрами зброї на різних рівнях прокачки, керівництва з проходження складних квестів, карти локацій з позначенням важливих об'єктів, відомості про артефакти та їхні бонуси, механізми елементальних реакцій та багато іншого. Інформація часто підтримується активною спільнотою, яка оперативно оновлює дані після виходу нових патчів та ігрових подій.
- Можливість порівняння та аналізу: Веб-сайти та вікі часто надають інструменти для порівняння різних ігрових елементів (наприклад, характеристики зброї або навички персонажів), що допомагає гравцям приймати обґрунтовані рішення щодо розвитку своїх персонажів та вибору екіпірування.
- Візуалізація даних: Багато з цих ресурсів використовують зображення, відео та інтерактивні елементи (наприклад, інтерактивні карти), що полегшує сприйняття складної інформації.

Однак, незважаючи на ці значні переваги, використання веб-сайтів та вікі-ресурсів має суттєві недоліки з точки зору оперативного отримання інформації, особливо в контексті активного спілкування в месенджерах:

- Необхідність залишати месенджер: Для отримання потрібної інформації гравцеві необхідно відволіктися від свого поточного спілкування в Telegram або іншій платформі, відкрити браузер та перейти на

відповідний веб-сайт. Цей процес може бути незручним, особливо якщо інформація потрібна швидко під час обговорення ігрової ситуації з іншими гравцями.

- Пошук інформації вручну: Навіть на добре структурованих веб-сайтах гравцеві часто доводиться використовувати пошукові системи або складну систему навігації, щоб знайти конкретну інформацію. Це може зайняти певний час, особливо якщо гравець не знає точної назви потрібного об'єкта або терміну.
- Великий обсяг нерелевантної інформації: Веб-сайти та вікі часто містять величезну кількість інформації, яка може бути не потрібна гравцеві в даний момент. Просіювання цього масиву даних у пошуках конкретної відповіді може бути неефективним та стомлюючим.
- Реклама та складний інтерфейс: Деякі фан-сайти можуть містити велику кількість реклами, що ускладнює сприйняття контенту. Інтерфейс деяких вікі може бути перевантаженим та не завжди інтуїтивно зрозумілим, особливо для новачків.
- Відсутність миттєвої відповіді: На відміну від чат-бота, який може надати швидку відповідь на конкретний запит, використання веб-сайтів вимагає самостійного пошуку та аналізу інформації.

Таким чином, хоча веб-сайти та вікі-ресурси є незамінним джерелом глибоких знань про Genshin Impact, їхнє використання може бути неоптимальним у ситуаціях, коли гравцеві потрібен швидкий та зручний доступ до конкретної інформації безпосередньо в процесі спілкування або гри. Саме цю проблему покликані вирішити чат-боти, інтегровані в звичні для гравців месенджери.

1.2.2 Мобільні додатки

Окрім веб-сайтів та вікі-ресурсів, на ринку існують різноманітні мобільні додатки, розроблені незалежними ентузіастами та невеликими командами, які пропонують гравцям Genshin Impact агреговану довідкову

інформацію та корисні інструменти. Ці додатки прагнуть забезпечити зручний доступ до даних безпосередньо з мобільних пристроїв, якими більшість гравців користуються для спілкування та ігрового процесу.

Переваги мобільних додатків:

- Інтерфейс, оптимізований для мобільних пристроїв: Однією з ключових переваг мобільних додатків є їхній користувацький інтерфейс, спеціально розроблений для невеликих екранів та сенсорного керування. Це може значно покращити зручність навігації та перегляду інформації порівняно з веб-сайтами, особливо на смартфонах.
- Інтегровані інструменти: Багато мобільних додатків пропонують не лише довідкову інформацію, але й додаткові інструменти, що полегшують ігровий процес. Це можуть бути:
    - Калькулятори шкоди: Дозволяють гравцям оцінювати потенційну шкоду від атак персонажів з урахуванням різних параметрів.
    - Планувальники розвитку персонажів: Допомагають відстежувати необхідні ресурси для прокачки персонажів, зброї та талантів, а також показують їхнє місцезнаходження.
    - Трекери прогресу: Дають змогу відстежувати виконання щоденних завдань, збір смоли, прогрес у дослідженнях та інші важливі аспекти гри.
    - Інструменти для складання команд: Можуть надавати рекомендації щодо оптимальних комбінацій персонажів для різних випробувань.
    - Інтерактивні карти: Спрощують пошук ресурсів, скарбів та інших важливих об'єктів у відкритому світі.
- Push-сповіщення: Мобільні додатки часто мають функцію push-сповіщень, яка може інформувати гравців про початок нових ігрових подій, нагадувати про щоденні завдання або повідомляти про інші

важливі оновлення в грі. Це допомагає залишатися в курсі подій, не пропускаючи важливих можливостей.

- Офлайн-доступ до інформації: Деякі мобільні додатки можуть надавати можливість доступу до певної частини інформації в режимі офлайн, що може бути корисним за відсутності стабільного інтернет-з'єднання.

Проте, використання мобільних додатків для Genshin Impact також має певні недоліки та потенційні ризики:

- Необхідність встановлення додаткового програмного забезпечення: Завантаження та встановлення додаткового застосунку може бути не завжди бажаним для користувачів, особливо якщо вони мають обмежений обсяг пам'яті на своїх пристроях або не хочуть встановлювати багато сторонніх програм.
- Ризики конфіденційності та безпеки даних: Оскільки більшість цих додатків є неофіційними розробками, існує певний ризик щодо конфіденційності особистих даних користувачів. Деякі додатки можуть запитувати надмірні дозволи на доступ до інформації на пристрої, що викликає обґрунтовані побоювання щодо їхнього використання.
- Актуальність та достовірність інформації: Інформація в неофіційних додатках може не завжди бути актуальною або достовірною, оскільки вона залежить від зусиль розробників щодо її оновлення після виходу нових патчів та ігрових змін.
- Стабільність роботи та підтримка: Неофіційні додатки можуть бути менш стабільними у роботі та не мати постійної технічної підтримки. З часом вони можуть ставати несумісними з новими версіями операційних систем або гри.
- Потенційна наявність реклами: Для підтримки розробки деякі безкоштовні мобільні додатки можуть містити рекламу, що може погіршувати користувацький досвід.

Таким чином, мобільні додатки є зручним інструментом для отримання інформації про Genshin Impact безпосередньо на мобільних пристроях та пропонують корисні інтегровані інструменти. Однак, користувачам слід бути обережними при виборі та використанні неофіційних додатків, враховуючи потенційні ризики для конфіденційності та стабільності роботи. Розробка чат-бота в месенджері, яким вже користуються гравці, може стати більш безпечною та зручною альтернативою для швидкого доступу до ключової інформації.

1.2.3 Чат-боти

У популярних серед геймерської спільноти месенджерах, таких як Telegram та Discord, сформувалася певна екосистема чат-ботів, розроблених силами ентузіастів для задоволення інформаційних потреб гравців Genshin Impact. Ці боти стали своєрідними цифровими помічниками, які надають гравцям можливість отримувати різноманітну інформацію про гру безпосередньо в звичному середовищі спілкування.

Функціональність цих чат-ботів може варіюватися в широкому діапазоні, залежно від задуму розробників та їхніх технічних можливостей:

- Базові інформаційні боти: Найпростіші з цих ботів реагують на конкретні команди, надаючи обмежений набір інформації. Наприклад, користувач може ввести команду на кшталт /character <ім'я_персонажа> або /weapon <назва_зброї> і отримати у відповідь базові характеристики, опис або зображення відповідного ігрового об'єкта. Ці боти часто використовують заздалегідь підготовлені бази даних або прості API для видачі інформації.
- Боти з розширеними можливостями фільтрації: Більш складні чат-боти надають користувачам можливість фільтрувати інформацію за різними критеріями. Наприклад, можна запросити список усіх п'ятизіркових персонажів певного елементу (/characters element=cryo rarity=5) або перелік усіх мечів чотиризіркової рідкості (/weapons type=sword

rarity=4). Це значно полегшує пошук конкретної інформації серед великої кількості ігрових об'єктів.

- Боти з порадами щодо складання команд: Деякі чат-боти пропонують гравцям рекомендації щодо формування оптимальних команд персонажів для різних ігрових ситуацій, враховуючи елементальні реакції, синергію навичок та ролі персонажів у команді. Ці боти можуть базуватися на аналізі популярних стратегій спільноти або власних алгоритмах оцінки ефективності команд.
- Боти для відстеження ігрових подій: Окремий клас ботів спеціалізується на наданні інформації про поточні та майбутні ігрові події, такі як нові банери персонажів та зброї, тимчасові івенти, нагороди за входження та інші важливі оновлення. Вони можуть надсилати сповіщення про початок нових подій або нагадувати про їх завершення.
- Боти з інтеграцією з іншими сервісами: Деякі розробники намагаються інтегрувати своїх ботів з іншими онлайн-сервісами, такими як HoYoLAB (офіційна платформа miHoYo) для отримання інформації про прогрес гравця, стан смоли або щоденні завдання. Однак, така інтеграція часто є неофіційною і може бути нестабільною через відсутність офіційного API.

Важливо підкреслити, що переважна більшість чат-ботів для Genshin Impact є результатом зусиль окремих ентузіастів або невеликих груп розробників. Вони не мають офіційної підтримки від компанії miHoYo, розробника гри. Це має свої наслідки:

- Нестабільність роботи: Робота таких ботів може бути нестабільною, вони можуть періодично виходити з ладу через технічні проблеми або зміни в API месенджерів.
- Негарантована актуальність інформації: Актуальність інформації, яку надають ці боти, повністю залежить від відповідальності та

можливостей їхніх розробників. З виходом нових оновлень гри дані в ботах можуть застарівати, якщо їх вчасно не оновити.

- Потенційні ризики безпеки: Хоча більшість ботів розробляються з добрими намірами, існує певний ризик використання неперевірених сторонніх розробок, особливо якщо вони запитують доступ до особистої інформації або інших чутливих даних (хоча більшість інформаційних ботів цього не роблять).

Незважаючи на ці потенційні недоліки, чат-боти залишаються популярним інструментом серед гравців Genshin Impact завдяки своїй зручності та швидкості надання інформації безпосередньо в месенджері. Розроблений у даній роботі "Помічничок мандрівника Genshin Impact" також належить до цієї категорії, прагнучи надати користувачам простий та ефективний спосіб отримання ключової інформації про персонажів та зброю через інтуїтивно зрозумілий інтерфейс Telegram.

ВИСНОВОК ДО РОЗДІЛУ 1

У результаті проведеного аналізу існуючих інформаційно-довідкових рішень для гри Genshin Impact можна виокремити кілька ключових напрямів, у яких користувачі отримують довідкову інформацію: офіційні та фанатські вебсайти, мобільні додатки, плагіни для браузерів, а також боти в месенджерах. Кожен з цих форматів має свої переваги та обмеження.

Вебресурси (офіційні сайти, вікі тощо) пропонують найбільш повну та структуровану інформацію, однак мають суттєвий недолік — вони не інтегровані у повсякденну комунікацію користувача. Доводиться щоразу відкривати браузер, вручну шукати дані, перемикатися між вкладками, що знижує загальну зручність використання.

Мобільні додатки, хоч і забезпечують швидший доступ до інформації, часто потребують встановлення, займають місце на пристрої, можуть містити рекламу або непотрібний функціонал, а також нерідко оновлюються із запізненням, що призводить до появи неактуальних даних.

Чат-боти, зокрема в Telegram, мають низку переваг:

- інтеграція безпосередньо у месенджер, яким користуються щодня;
- швидкий доступ до інформації за допомогою команд без потреби перемикатися між програмами;
- можливість персоналізації, збереження історії запитів, контекстної відповіді тощо;
- проста реалізація пошуку за запитами, фільтрації та сортування даних.

Зважаючи на вищезазначене, вибір на користь створення Telegram-бота є обґрунтованим і доцільним. Такий інструмент здатен поєднати зручність доступу, швидкість реакції, структурованість інформації та гнучкість функціоналу. Крім того, Telegram надає добре задокументоване API, що значно полегшує технічну реалізацію проєкту.

У підсумку, Telegram-бот для довідкової підтримки гравців Genshin Impact має потенціал стати ефективним, доступним та інтерактивним

рішенням, яке усуває недоліки інших форматів та повністю відповідає потребам цільової аудиторії.

РОЗДІЛ 2.
ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Мова програмування Python

Python, розроблений Гвідо ван Россумом і вперше представлений у 1991 році, є високорівневою, інтерпретованою мовою програмування загального призначення. Його філософія дизайну наголошує на читабельності коду завдяки значному використанню відступів для структурування блоків коду.

2.1.1 Загальна характеристика мови Python

Історія розвитку та ключові особливості Python:

Python бере свій початок як хобі-проєкт ван Россума, натхненний мовою ABC. З часом він еволюціонував у потужну та універсальну мову, що підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, імперативне та функціональне програмування. Ключові особливості Python включають:

- Інтерпретованість: Код Python виконується построково інтерпретатором, що спрощує розробку та налагодження.
- Динамічна типізація: Типи змінних визначаються під час виконання, що зменшує обсяг шаблонного коду.
- Велика стандартна бібліотека: Python постачається з обширною бібліотекою модулів для виконання різноманітних завдань, від роботи з файлами та мережами до обробки тексту та даних.
- Простий та читабельний синтаксис: Синтаксис Python розроблено таким чином, щоб бути максимально зрозумілим та легким для вивчення, що сприяє швидкій розробці та полегшує читання коду іншими розробниками.
- Кросплатформність: Код Python може виконуватися на різних операційних системах (Windows, macOS, Linux) без значних змін.

Переваги мови Python для розробки Telegram-ботів:

Python зарекомендував себе як одна з найпопулярніших мов для розробки ботів, зокрема для платформи Telegram. Його переваги в цьому контексті є очевидними:

- Простота та швидкість розробки: Лаконічний синтаксис Python дозволяє швидко прототипувати та розробляти функціональних ботів, що є важливим у динамічному середовищі розробки.
- Велика кількість бібліотек: Існує кілька потужних бібліотек Python, спеціально розроблених для роботи з Telegram Bot API, серед яких особливо виділяється python-telegram-bot. Ці бібліотеки значно спрощують взаємодію з API, надаючи зручні інструменти для обробки оновлень, надсилання повідомлень, керування клавіатурами та іншими функціями Telegram.
- Асинхронність: Python підтримує асинхронне програмування за допомогою бібліотеки asyncio, що є критично важливим для розробки ботів, які мають обробляти велику кількість одночасних запитів без блокування основного потоку виконання.

Сфера застосування мови Python:

Універсальність Python зумовила його широке застосування в багатьох сферах, включаючи:

- Веб-розробка: Фреймворки Django та Flask є одними з найпопулярніших інструментів для створення веб-сайтів та веб-додатків.
- Аналіз даних та машинне навчання: Бібліотеки Pandas, NumPy, SciPy, Scikit-learn та TensorFlow зробили Python лідером у галузі науки про дані та штучного інтелекту.
- Автоматизація та скрипти: Python часто використовується для написання скриптів для автоматизації рутинних завдань, системного адміністрування та обробки файлів.
- Розробка ігор: Бібліотеки Pygame дозволяють створювати прості та середні за складністю ігри.

- Наукові обчислення: Python є популярним вибором для наукових досліджень завдяки своїм потужним бібліотекам для математичних обчислень та моделювання.

2.1.2 Обґрунтування вибору мови Python

Вибір мови програмування є одним з найважливіших рішень у процесі розробки будь-якого програмного забезпечення. Для проєкту "Помічничок мандрівника Genshin Impact" мова Python була обрана з кількох ключових причин:

- Простота синтаксису та швидкість розробки: Читабельний та лаконічний синтаксис Python значно прискорює процес написання коду та зменшує кількість потенційних помилок. Це особливо важливо на початкових етапах розробки та при швидкій ітерації функціональності бота.
- Велика екосистема бібліотек для різних завдань: Як було зазначено раніше, наявність спеціалізованих бібліотек для роботи з Telegram API (python-telegram-bot) та інструментів для асинхронного програмування (asyncio) значно полегшує розробку ботів. Крім того, стандартна бібліотека Python надає безліч корисних модулів для роботи з файлами, обробки тексту та інших загальних завдань.
- Активна спільнота та доступність ресурсів: Python має велику та активну спільноту розробників по всьому світу. Це означає наявність великої кількості документації, навчальних матеріалів, онлайн-курсів та форумів, де можна знайти відповіді на питання та отримати допомогу у вирішенні проблем. Для розробника-початківця або досвідченого програміста це є значною перевагою.

У підсумку, вибір Python для розробки "Помічничка мандрівника Genshin Impact" є стратегічно виправданим завдяки його простоті, потужній екосистемі бібліотек та великій спільноті, що сприяє ефективній та швидкій

розробці функціонального та зручного Telegram-бота для гравців Genshin Impact.

2.2 Основні бібліотеки для розробки Telegram-бота

Для ефективної розробки Telegram-бота "Помічничок мандрівника Genshin Impact" ключову роль відіграють дві основні бібліотеки: telegram та telegram.ext. Вони надають необхідні інструменти для взаємодії з Telegram Bot API, обробки подій та керування роботою бота.

2.2.1 Бібліотека telegram

Бібліотека telegram є низькорівневою обгорткою над Telegram Bot API. Вона надає прямий доступ до всіх методів та об'єктів API, дозволяючи розробникам гнучко керувати взаємодією з серверами Telegram.

Огляд основних модулів та класів:

- Update: Цей клас представляє собою оновлення, отримане від Telegram. Оновлення можуть включати нові повідомлення, редагування повідомлень, реакції, запити зворотного виклику з inline-клавіатур, inline-запити та багато іншого. Об'єкт Update містить інформацію про подію та користувача, який її ініціював.
- Bot: Клас Bot є основним об'єктом для взаємодії з Telegram Bot API. Він інкапсулює токен вашого бота та надає методи для надсилання повідомлень, редагування їх, керування клавіатурами, завантаження та надсилання файлів та виконання інших дій, передбачених API. Наприклад, Bot.send_message(), Bot.edit_message_text(), Bot.send_photo().
- ReplyKeyboardMarkup: Цей клас використовується для створення клавіатур, які відображаються під полем введення тексту у користувача. Вони можуть містити звичайні кнопки з текстом або кнопки для запиту геолокації чи контактних даних користувача.
- InlineKeyboardMarkup: На відміну від ReplyKeyboardMarkup, inline-клавіатури прикріплюються безпосередньо до повідомлення. Кнопки в

inline-клавіатурах можуть виконувати різні дії, такі як відкриття URL-адрес, виконання запитів зворотного виклику (CallbackQuery), надсилання inline-запитів або перемикання на інший inline-запит.

- InlineKeyboardButton: Цей клас представляє окрему кнопку в складі InlineKeyboardMarkup. Кожна кнопка повинна мати текст та один з параметрів, що визначає її дію (наприклад, url, callback_data, switch_inline_query_current_chat).
- InputMediaPhoto: Цей клас використовується для надсилання фотографій як частину медіа-групи. Він дозволяє об'єднувати кілька фотографій в одне повідомлення для кращого візуального представлення інформації.
- InputFile: Клас InputFile використовується для завантаження та надсилання локальних файлів (зображень, документів, аудіо, відео) через API. Він обертає файловий об'єкт, роблячи його придатним для передачі методам Bot.

Принципи взаємодії з Telegram Bot API через об'єкт Bot:

Взаємодія з Telegram Bot API через бібліотеку telegram відбувається шляхом виклику методів об'єкта Bot. Кожен метод відповідає певному API-запиту. Наприклад, щоб надіслати текстове повідомлення користувачеві, використовується метод bot.send_message(chat_id=user_id, text='Привіт!'). Для надсилання inline-клавіатури до повідомлення використовується параметр reply_markup при виклику методів надсилання або редагування повідомлень, якому передається об'єкт InlineKeyboardMarkup.

Використання різних типів клавіатур та медіа-вкладень:

У проєкті "Помічничок мандрівника Genshin Impact" використовуються обидва типи клавіатур для забезпечення зручної навігації та взаємодії користувача з ботом. ReplyKeyboardMarkup може використовуватися для відображення основного меню бота, а InlineKeyboardMarkup є ідеальним для багаторівневих меню вибору (наприклад, вибір елементу, потім рідкості

персонажа або зброї) завдяки своїй здатності оновлювати повідомлення без спаму чату новими повідомленнями. Клас InputMediaPhoto застосовується для надсилання зображень персонажів, зброї або артефактів, покращуючи візуальне сприйняття інформації. InputFile використовується для завантаження цих зображень з локального сховища бота.

2.2.2 Бібліотека telegram.ext

Бібліотека telegram.ext є надбудовою над бібліотекою telegram, яка значно спрощує процес розробки ботів, надаючи високорівневі абстракції для обробки оновлень та керування станом бота. Вона використовує патерн "обробник" (handler), що дозволяє організувати логіку бота у вигляді набору незалежних функцій, які реагують на певні типи оновлень.

Огляд обробників:

- Application (або Updater): Це центральний об'єкт telegram.ext, який отримує оновлення від Telegram та передає їх відповідним обробникам. Application є більш сучасним та рекомендованим способом керування ботом. Updater є застарілим класом, але все ще може зустрічатися в існуючому коді.
- CommandHandler: Цей обробник використовується для обробки команд, які починаються зі слеша (/). Він пов'язує певну команду (наприклад, /start, /character) з функцією-обробником, яка буде викликана при отриманні цієї команди.
- MessageHandler: Цей обробник відповідає за обробку текстових повідомлень або інших типів повідомлень (фото, відео, аудіо тощо) на основі заданих фільтрів. Фільтри визначають, які типи повідомлень повинен обробляти даний обробник.
- CallbackQueryHandler: Цей обробник призначений для обробки запитів зворотного виклику, які генеруються натисканням на кнопки в inline-клавіатурах. Він дозволяє боту реагувати на конкретні дії користувача, пов'язані з inline-кнопками.

Фільтри (filters) для обробки різних типів оновлень:

Модуль filters в telegram.ext надає набір попередньо визначених фільтрів (filters.TEXT, filters.COMMAND, filters.PHOTO) та дозволяє створювати власні фільтри для точного визначення того, які оновлення повинен обробляти певний MessageHandler. Це забезпечує гнучкість у керуванні логікою бота.

Контекст обробки (CallbackContext):

CallbackContext — це об'єкт, який передається до функцій-обробників. Він використовується для зберігання та обміну даними між різними обробниками протягом життєвого циклу обробки одного оновлення або між послідовними взаємодіями з користувачем. Це дозволяє підтримувати стан розмови та персоналізувати взаємодію.

Запуск та керування ботом (Application.run_polling() або Updater.start_polling()):

Для запуску бота та початку отримання оновлень від Telegram використовуються методи Application.run_polling() (для Application) або Updater.start_polling() (для Updater). Ці методи запускають цикл опитування (polling), під час якого бот періодично надсилає запити до серверів Telegram для отримання нових оновлень. Після отримання оновлення воно передається до відповідних обробників на основі визначених правил (команди, фільтри, запити зворотного виклику).

У проєкті "Помічничок мандрівника Genshin Impact" бібліотека telegram.ext є основним інструментом для організації логіки бота, обробки команд користувача та реакції на натискання кнопок у меню вибору. Використання обробників та фільтрів значно спрощує структуру коду та робить його більш керованим.

2.3 Бібліотеки для асинхронного програмування

В умовах розробки Telegram-бота, який має обробляти велику кількість одночасних запитів від користувачів, ефективне керування ресурсами та забезпечення неблокуючої роботи є критично важливим. Для цього в Python

використовується механізм асинхронного програмування, реалізований за допомогою бібліотеки asyncio.

Бібліотека asyncio надає інфраструктуру для написання одночасного коду, використовуючи синтаксис async/await. Вона є основою для асинхронного програмування в Python і дозволяє виконувати кілька задач, які можуть очікувати на зовнішні події (наприклад, відповідь від сервера Telegram), без блокування основного потоку виконання програми.

Принципи асинхронного програмування в Python:

Асинхронне програмування відрізняється від традиційного послідовного (синхронного) виконання коду. Замість того, щоб чекати завершення кожної операції перед переходом до наступної, асинхронний код може призупинити виконання поточної задачі, коли вона очікує на зовнішню подію, і передати керування іншій готовій до виконання задачі. Коли перша задача отримує необхідні дані, вона може бути відновлена.

Ключові концепції asyncio включають:

- Події (Event Loop): Серце asyncio, що керує виконанням асинхронних задач. Цикл подій відстежує готові до виконання корутини та виконує їх по черзі.
- Корутини (Coroutines): Спеціальні функції, які можуть призупиняти та відновлювати своє виконання. Вони оголошуються за допомогою ключового слова async.
- Очікування (Awaitables): Об'єкти, які можуть бути "очікувані" (awaited). Корутини є одним з типів awaitables. Коли використовується ключове слово await перед awaitable, виконання поточної корутини призупиняється до завершення awaitable, після чого виконання корутини відновлюється.
- Завдання (Tasks): Корутини, загорнуті в об'єкти Task, які керуються циклом подій. Завдання можуть виконуватися паралельно.

Використання asyncio для обробки одночасних запитів та неблокуючих операцій:

У контексті Telegram-бота, бібліотека asyncio є надзвичайно корисною для обробки великої кількості одночасних запитів від користувачів. Коли бот отримує оновлення від Telegram API, кожне оновлення може бути оброблено як окрема асинхронна задача. Якщо під час обробки задачі боту необхідно виконати операцію, яка може зайняти певний час (наприклад, відправити запит до зовнішньої бази даних, завантажити файл), використання await дозволяє призупинити виконання поточної задачі та дати можливість циклу подій обробляти інші оновлення або виконувати інші асинхронні задачі.

Бібліотека python-telegram-bot, яка використовується у проєкті, інтегрована з asyncio. Обробники подій (функції, які обробляють команди, повідомлення, запити зворотного виклику) можуть бути визначені як асинхронні функції (async def). При отриманні оновлення, python-telegram-bot створює асинхронну задачу для його обробки та передає керування циклу подій asyncio. Це дозволяє боту залишатися чутливим до нових запитів, навіть коли він зайнятий обробкою попередніх.

Завдяки використанню asyncio, бот "Помічничок мандрівника Genshin Impact" може ефективно обробляти велику кількість користувачів одночасно, не блокуючи свій основний потік виконання та забезпечуючи швидку та чуйну взаємодію. Кожна дія користувача обробляється асинхронно, що робить роботу бота більш плавною та продуктивною.

2.4 Бібліотеки для роботи з даними та файлами

Ефективне керування та обробка даних є ключовим аспектом розробки бота "Помічничок мандрівника Genshin Impact", оскільки бот надає інформацію про велику кількість ігрових об'єктів. Для цієї мети використовуються кілька стандартних та сторонніх бібліотек Python, які спрощують роботу з даними та файловою системою.

2.4.1 Бібліотека json

Бібліотека json у Python надає інструменти для роботи з даними у форматі JSON (JavaScript Object Notation). JSON є легким форматом обміну даними, який легко читається та генерується як людьми, так і машинами. Він часто використовується для передачі даних між веб-серверами та клієнтами, а також для зберігання структурованих даних у файлах.

Використання для зчитування та обробки даних у форматі JSON:

У проєкті "Помічничок мандрівника Genshin Impact" бібліотека json використовується для зчитування інформації про персонажів та зброю з окремих файлів (character_data.py, weapon_data.py). Хоча назви файлів мають розширення .py, вони, містять дані, представлені у форматі JSON (у вигляді рядків Python), які потім обробляються за допомогою функцій бібліотеки json.

Основні функції бібліотеки json, що використовуються для цієї мети, включають:

- json.load(f): Ця функція зчитує дані з відкритого файлового об'єкта f, що містить JSON-форматований текст, та перетворює їх у відповідні об'єкти Python (словники, списки, рядки, числа, булеві значення, None).
- json.loads(s): Ця функція приймає рядок s, що містить JSON-форматовані дані, та парсить його у об'єкт Python.

Після зчитування даних з файлів за допомогою json.load() або json.loads(), інформація про персонажів та зброю стає доступною у вигляді структурованих об'єктів Python (зазвичай, це вкладені словники та списки), що дозволяє легко отримувати доступ до конкретних атрибутів (назви, характеристики, навички тощо) та використовувати ці дані для надання інформації користувачам бота.

2.4.2 Бібліотека os

Бібліотека os надає спосіб використовувати залежні від операційної системи функціональні можливості, такі як взаємодія з файловою системою. Вона включає функції для створення, видалення, перейменування файлів та

каталогів, перевірки їх існування, отримання інформації про них, а також для роботи зі шляхами до файлів.

Робота з файловою системою (наприклад, для формування шляхів до зображень):

У контексті бота "Помічничок мандрівника Genshin Impact" бібліотека os може використовуватися для різних завдань, пов'язаних з файловою системою, зокрема для:

- Формування шляхів до файлів: При роботі зі статичними ресурсами, такими як зображення персонажів та зброї, необхідно мати правильні шляхи до цих файлів на сервері, де запущено бот. Функції os.path.join() дозволяють безпечно та платформо-незалежно об'єднувати частини шляху до файлу.
- Перевірка існування файлів: Перед спробою надіслати зображення користувачеві, бот може використовувати функції os.path.exists() для перевірки, чи існує відповідний файл зображення на диску.
- Робота з каталогами: У випадку більш складної організації файлів, бібліотека os може використовуватися для переліку файлів у каталозі, створення нових каталогів тощо.

Використання os забезпечує надійну та кросплатформну роботу бота з локальними файлами, що є важливим для надання візуальної інформації користувачам.

2.4.3 Бібліотека collections

Бібліотека collections надає спеціалізовані типи контейнерів, які є альтернативою вбудованим контейнерам Python (таким як dict, list, set, tuple) та надають додаткову функціональність.

Використання defaultdict для організації даних:

Одним з корисних класів, що надається бібліотекою collections, є defaultdict. defaultdict схожий на звичайний словник (dict), але відрізняється тим, що при спробі доступу до відсутнього ключа він не викликає помилку

KeyError, а замість цього створює новий ключ зі значенням за замовчуванням, яке визначається при створенні defaultdict.

У проєкті "Помічничок мандрівника Genshin Impact" defaultdict може використовуватися для організації даних про персонажів або зброю за певними критеріями, наприклад, за елементом, рідкістю тощо. Це може спростити процес фільтрації та пошуку інформації для надання користувачам. Наприклад, можна створити defaultdict(list), де ключами будуть елементи, а значеннями - списки персонажів цього елементу. При спробі додати нового персонажа до елементу, який ще не представлений як ключ, буде автоматично створено новий ключ зі значенням у вигляді порожнього списку.

Використання defaultdict може зробити код більш лаконічним та ефективним при роботі з даними, які потрібно групувати або індексувати за певними ознаками.

У сукупності, бібліотеки json, os та collections відіграють важливу роль у забезпеченні функціональності бота "Помічничок мандрівника Genshin Impact", відповідаючи за зчитування, обробку та організацію даних, а також за роботу з файловою системою для надання користувачам повної та візуально привабливої інформації про ігровий світ Genshin Impact.

2.5 Бібліотеки для роботи з розкладом

Для реалізації функціональності, що залежить від часу, наприклад, автоматизованого надсилання привітань з днем народження персонажів гри, у проєкті "Помічничок мандрівника Genshin Impact" використовується бібліотека apscheduler.

Apscheduler є потужною бібліотекою Python, яка надає гнучкий механізм для планування виконання завдань (jobs) у фоновому режимі в заданий час або через певні інтервали. Вона підтримує різні "backends" для зберігання інформації про заплановані завдання, включаючи in-memory, SQLite та бази даних SQL.

Застосування BackgroundScheduler для виконання завдань за розкладом:

У контексті бота "Помічничок мандрівника Genshin Impact" використовується один з шедулерів, що надаються apscheduler, а саме BackgroundScheduler. Цей шедулер запускає заплановані завдання у окремому фоновому потоці, не блокуючи основний цикл обробки оновлень бота. Це дозволяє боту залишатися чутливим до команд користувачів, навіть коли виконуються заплановані завдання.

Основний процес використання BackgroundScheduler включає наступні кроки:

1. Створення екземпляра шедулера:

```
from apscheduler.schedulers.background import BackgroundScheduler
scheduler = BackgroundScheduler()
```

2. Додавання завдань до шедулера: Завдання визначаються у вигляді функцій, які потрібно виконати, та тригерів, що визначають, коли ці функції повинні бути викликані. apscheduler підтримує різні типи тригерів:
    - date: Запускає завдання один раз у вказаний час.
    - interval: Запускає завдання через певні інтервали часу.
    - cron: Запускає завдання за розкладом, визначеним у форматі cron.

Наприклад, для надсилання привітань з днем народження щодня о 10:00 ранку можна використовувати cron-тригер:

```
from birthday_sender import send_birthday_greeting
scheduler.add_job(send_birthday_greeting, 'cron', hour=10, minute=0)
```

Тут функція send_birthday_greeting буде викликатися щодня о 10:00.

3. Запуск шедулера: Після додавання всіх необхідних завдань, шедулер потрібно запустити, щоб він почав відстежувати час та виконувати заплановані завдання:

```
scheduler.start()
```

Приклад: Надсилання привітань з днем народження:

У проєкті "Помічничок мандрівника Genshin Impact" BackgroundScheduler використовується для автоматизованого надсилання

привітань користувачам, які підписалися на цю функцію, у дні народження їхніх улюблених персонажів. Функція send_birthday_greeting (імпортована з модуля birthday_sender.py) виконує наступні дії:

- Отримує поточну дату.
- Перевіряє, чи є сьогодні дні народження будь-яких персонажів гри на основі завантаженої бази даних днів народження (load_character_birthdays, також імпортована з birthday_sender.py).
- Для кожного персонажа, у якого сьогодні день народження, формує привітання (можливо, з зображенням персонажа та побажаннями).
- Надсилає це привітання всім користувачам, які підписалися на отримання таких повідомлень.

Використання apscheduler дозволяє автоматизувати цей процес, забезпечуючи своєчасне надсилання привітань без необхідності ручного втручання, що значно покращує взаємодію з користувачами та додає інтерактивності боту. Запуск шедулера у фоновому потоці гарантує, що ця функціональність не впливає на основну роботу бота з обробки команд користувачів.

2.6 Бібліотеки для роботи з часом

Для точної обробки дат та часу, що є необхідним для визначення днів народження персонажів гри та інших функціональностей, пов'язаних з часом, у проєкті "Помічничок мандрівника Genshin Impact" використовується вбудована бібліотека Python datetime.

Бібліотека datetime надає класи для роботи з датами та часом. Вона дозволяє створювати об'єкти, що представляють конкретні моменти часу, дати, часові інтервали, а також виконувати різноманітні операції над ними, такі як порівняння, додавання, віднімання, форматування та парсинг.

Використання для обробки дат та часу (наприклад, для визначення днів народження):

У контексті бота "Помічничок мандрівника Genshin Impact", бібліотека datetime є ключовою для реалізації функціональності, пов'язаної з днями народження персонажів. Процес визначення та обробки днів народження включає наступні кроки з використанням класів та методів бібліотеки datetime:

1. Зберігання дат народження: Інформація про дні народження персонажів, швидше за все, зберігається у структурованому форматі (наприклад, у файлах JSON або в базі даних). Ці дані можуть включати місяць та день народження кожного персонажа.
2. Отримання поточної дати: Для визначення, чи є сьогодні у когось день народження, необхідно отримати поточну дату. Це робиться за допомогою класу date та його методу today():

```
from datetime import date
today = date.today()
current_month = today.month
current_day = today.day
```

3. Порівняння дат: Після отримання поточної дати, вона порівнюється з датами народження персонажів. Оскільки у базі даних днів народження можуть зберігатися лише місяць та день, порівняння відбувається саме за цими параметрами:

```
    character_birthdays = load_character_birthdays() # Припустимо, що
ця функція повертає словник {ім'я: (місяць, день)}
    for name, (month, day) in character_birthdays.items():
        if current_month == month and current_day == day:
            # Сьогодні день народження у персонажа з ім'ям name
            # Формуємо привітання та надсилаємо користувачам
            pass
```

4. Форматування дат (за потреби): Бібліотека datetime надає методи для форматування об'єктів date та datetime у зручний для читання формат. Наприклад, можна відформатувати дату народження для відображення у повідомленні користувачеві.
5. Робота з часовими зонами (якщо потрібно): У випадку, якщо бот має враховувати різні часові зони користувачів для надсилання привітань у

відповідний час, можна використовувати модуль timezone з бібліотеки datetime або сторонні бібліотеки, такі як pytz. Однак, судячи з наданого коду, базова обробка днів народження обмежується порівнянням місяця та дня.

Використання бібліотеки datetime є фундаментальним для реалізації будь-якої функціональності бота, що залежить від часу, забезпечуючи точну та надійну обробку дат та часу для коректного визначення днів народження персонажів та своєчасного надсилання привітань користувачам.

2.7 Бібліотеки для локалізації

Для забезпечення багатомовної підтримки та надання користувачам можливості взаємодії з ботом "Помічничок мандрівника Genshin Impact" їхньою рідною мовою, у проєкті реалізовано механізм локалізації. Основну роль у цьому відіграє файл localization.py.

Файл localization.py, містить структуровані дані, що відображають текстові повідомлення та елементи інтерфейсу бота різними мовами. Найімовірніше, це реалізовано за допомогою словника Python під назвою localized_texts.

Принципи реалізації багатомовності за допомогою словника localized_texts:

Словник localized_texts служить централізованим сховищем усіх текстових констант, які використовуються в боті. Ключами цього словника є коди мов (наприклад, 'uk' для української, 'en' для англійської), а значеннями є інші словники. Ці внутрішні словники, у свою чергу, містять ключі, що ідентифікують конкретні текстові рядки (наприклад, 'greeting_message', 'character_menu_title', 'weapon_type_sword'), та відповідні їм переклади цими мовами.

Приклад структури файлу localization.py:

```
localized_texts = {
'uk': {
    'greeting': 'Привіт! Чим цікавишся?',
    'characters': 'Персонажі',
    'weapons': 'Зброя',
    'artifacts': 'Артефакти',
    'main_menu': 'Головне меню',}
'en': {
     'greeting': 'Hello! What are you interested in?',
    'characters': 'Characters',
    'weapons': 'Weapons',
    'artifacts': 'Artifacts',
    'main_menu': 'Main Menu',}
}
```

У коді бота, коли необхідно відобразити певний текст користувачеві, замість безпосереднього використання рядкового літерала, відбувається звернення до словника localized_texts з вказівкою бажаної мови та ключа текстового рядка.

Наприклад, щоб отримати привітання українською мовою:

```
from localization import localized_texts
def send_greeting(update, context):
    user_language = 'uk' # Мову користувача можна визначати різними способами
    greeting = localized_texts.get(user_language, {}).get('greeting_message', 'Привіт!')
    update.message.reply_text(greeting)
```

2.8 Інші використані бібліотеки

Окрім основних бібліотек для роботи з Telegram API, асинхронності, даними та розкладом, у проєкті "Помічничок мандрівника Genshin Impact" також використовуються інші стандартні бібліотеки Python для виконання допоміжних завдань.

2.8.1 Бібліотека sys

Бібліотека sys надає доступ до деяких змінних та функцій, які тісно взаємодіють з інтерпретатором Python та його оточенням.

Використання для доступу до системних параметрів (наприклад, версії Python):

Одним з можливих застосувань бібліотеки sys у проєкті є отримання інформації про версію Python, на якій запущено бот. Це може бути корисно для цілей налагодження або для виведення інформації про оточення бота.

Приклад використання:

```
import sys
python_version = sys.version
print(f"Python version: {python_version}")
```

Хоча це не є основною функціональністю бота, доступ до системної інформації може бути корисним інструментом для розробників.

2.8.2 Бібліотека random

Бібліотека random надає функції для генерації псевдовипадкових чисел, а також для випадкового вибору елементів з послідовностей.

Можливе використання для випадкових виборів або подій:

У контексті бота "Помічничок мандрівника Genshin Impact" бібліотека random може використовуватися для реалізації різноманітних випадкових елементів або подій, якщо такі передбачені функціональністю бота. Наприклад:

- Випадкові поради або факти про гру: Бот може випадково вибирати та надсилати користувачам цікаві факти або корисні поради.
- Елементи гейміфікації: Якщо бот передбачає якісь ігрові елементи, random може використовуватися для визначення випадкових результатів або нагород.
- Випадковий вибір персонажа або зброї для прикладу: При запиті певної інформації бот може випадково вибирати один елемент зі списку для демонстрації.

Хоча в наданому переліку імпортів немає явного використання random у обробниках персонажів чи зброї, її імпорт свідчить про потенційне

використання цієї бібліотеки в інших частинах коду або для майбутньої функціональності бота.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі дипломного проєкту було здійснено детальний огляд ключових технологій та інструментів, що лягли в основу розробки Telegram-бота "Помічничок мандрівника Genshin Impact". Основним вибором стала мова програмування Python, переваги якої в контексті розробки ботів, такі як простота, велика екосистема бібліотек та підтримка асинхронності, були ретельно обґрунтовані.

Розглянуто дві фундаментальні бібліотеки для взаємодії з Telegram Bot API: telegram, що надає низькорівневий доступ до API та інструменти для роботи з оновленнями, клавіатурами та медіа-вкладеннями, та telegram.ext, яка значно спрощує розробку завдяки високорівневим обробникам подій та контексту обробки.

Було проаналізовано використання бібліотеки asyncio для забезпечення неблокуючої та ефективної обробки одночасних запитів від користувачів, що є критично важливим для продуктивності бота.

Розділ також охопив бібліотеки для роботи з даними та файлами: json для парсингу та обробки структурованої інформації про ігрові об'єкти, os для взаємодії з файловою системою при роботі зі статичними ресурсами, та collections з її класом defaultdict для зручної організації даних.

Окрема увага була приділена бібліотеці apscheduler, яка використовується для реалізації функціональності, залежної від часу, зокрема для автоматизованого надсилання привітань з днем народження персонажів у фоновому режимі. Бібліотека datetime була розглянута як основний інструмент для точної обробки дат та часу, необхідний для визначення днів народження.

Нарешті, було описано використання файлу localization.py та принципи реалізації багатомовності за допомогою словника localized_texts, що забезпечує зручність використання бота користувачами з різними мовними вподобаннями. Також було коротко розглянуто допоміжні бібліотеки sys та

random, які використовуються для доступу до системної інформації та реалізації випадкових елементів відповідно.

У сукупності, огляд представлених технологій та бібліотек демонструє обґрунтований та ефективний вибір інструментів, які дозволяють розробити функціонального, зручного та продуктивного Telegram-бота "Помічничок мандрівника Genshin Impact", здатного задовольнити інформаційні потреби гравців.

РОЗДІЛ 3.
ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Обробник команди /start

Функція обробки команди /start є початковою точкою взаємодії користувача з Telegram-ботом "Помічничок мандрівника Genshin Impact". Вона відповідає за привітання користувача, встановлення початкових налаштувань (наприклад, вибір мови) та надання основної інформації про можливості бота.

```
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
from localization import localized_texts

async def start(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['today'],
localized_texts[user_language]['characters']],
        [localized_texts[user_language]['weapons'],
localized_texts[user_language]['artifacts']],
        [localized_texts[user_language]['curiosities'],
localized_texts[user_language]['combat']] # Замінили кнопки
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['greeting'], reply_markup=reply_markup)
    context.user_data.pop('current_menu', None)
    context.user_data.pop('selected_element', None)
    context.user_data.pop('selected_weapon_type', None)
```

Детальний розбір оновленої функції start:

1. Визначення мови користувача:
    - user_language = context.user_data.get('language', 'uk') - як і в попередній версії, ця строка намагається отримати збережене значення мови користувача з context.user_data. Якщо ключ 'language' відсутній, за замовчуванням використовується українська мова ('uk'). Це гарантує, що початкове привітання та

кнопки меню будуть відображені мовою користувача (якщо вона вже була обрана раніше).

2. Створення головної клавіатури меню:
    - keyboard = [...] - створюється розмітка головної клавіатури меню. Тепер вона включає наступні кнопки (текст кнопок береться з localized_texts відповідно до мови користувача):
        - localized_texts[user_language]['today'] - кнопка для отримання інформації про поточні події або щоденні нагадування в грі.
        - localized_texts[user_language]['characters'] - кнопка для переходу до розділу інформації про персонажів.
        - localized_texts[user_language]['weapons'] - кнопка для переходу до розділу інформації про зброю.
        - localized_texts[user_language]['artifacts'] - кнопка для переходу до розділу інформації про артефакти.
        - localized_texts[user_language]['curiosities'] - кнопка для доступу до цікавих фактів, порад або іншої не основної інформації про гру.
        - localized_texts[user_language]['combat'] - кнопка для отримання інформації про бойову систему, механіки або поради щодо битв.
    - reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True) - створюється об'єкт ReplyKeyboardMarkup на основі визначеної розмітки keyboard.
        - resize_keyboard=True - вказує боту спробувати зменшити розмір клавіатури для кращого відображення на екрані користувача. На відміну від попередньої версії, параметр one_time_keyboard тут відсутній, що означає, що ця клавіатура головного меню буде залишатися закріпленою

під полем введення тексту до тих пір, поки користувач не викличе іншу клавіатуру або не сховає її вручну.

3. Надсилання привітального повідомлення з головним меню:
    - await context.bot.send_message(chat_id=update.effective_chat.id, text=localized_texts[user_language]['greeting'], reply_markup=reply_markup) - бот надсилає повідомлення з привітанням (текст береться з localized_texts відповідно до мови користувача) у чат з користувачем (update.effective_chat.id). До цього повідомлення прикріплюється створена головна клавіатура меню (reply_markup), надаючи користувачеві основні варіанти взаємодії з ботом.
4. Очищення тимчасових даних користувача:
    - context.user_data.pop('current_menu', None) - видаляє ключ 'current_menu' з context.user_data, якщо він існує. Це може використовуватися для відстеження поточного розділу меню, в якому знаходиться користувач, і його очищення при поверненні до головного меню. None як другий аргумент pop() запобігає виникненню помилки KeyError, якщо ключ не існує.
    - context.user_data.pop('selected_element', None) - аналогічно, видаляє ключ 'selected_element', який міг зберігати обраний елемент персонажа під час навігації у відповідному розділі.
    - context.user_data.pop('selected_weapon_type', None) - аналогічно, видаляє ключ 'selected_weapon_type', який міг зберігати обраний тип зброї.

Ці три рядки відповідають за скидання контексту користувача при поверненні до головного меню або на початку нової сесії. Це допомагає уникнути небажаної поведінки бота, спричиненої залишковими даними від попередніх дій користувача.

Узагальнення:

Оновлена функція start виконує наступні ключові дії:

- Визначає мову користувача.
- Створює головну клавіатуру меню з основними розділами бота.
- Надсилає привітання разом з головним меню.
- Очищає тимчасові дані користувача, щоб забезпечити чистий стан для подальшої взаємодії.

Після виконання цієї функції, бот очікує від користувача натискання однієї з кнопок головного меню. Подальша навігація та обробка запитів відбуватимуться у відповідних обробниках текстових повідомлень, які будуть реагувати на текст натиснутих кнопок (наприклад, якщо користувач натисне кнопку з текстом, що відповідає localized_texts[user_language]['characters'], буде викликано обробник для відображення меню персонажів).

3.2 Розробка основних обробників бота

Основні обробники бота є ключовими компонентами, що забезпечують безпосередню взаємодію з користувачем. Вони відповідають за інтерпретацію команд та текстових повідомлень, що надходять від користувача через натискання клавіатурних кнопок, та за відповідну обробку запитів.

3.2.1 Обробка персонажів

Обробка інформації про персонажів є однією з ключових функціональностей бота. Вона включає відображення багаторівневого меню, фільтрацію персонажів за елементом та рідкістю, виведення списку персонажів та надання детальної інформації про обраного персонажа, включаючи зображення та опис. Нижче наведено детальний опис усіх компонентів, що беруть участь у цьому процесі, разом із відповідними фрагментами коду.

1. Модуль character_handlers (файл character_handlers.py):

Цей модуль містить функції, відповідальні за формування та надсилання повідомлень з клавіатурами для навігації по меню персонажів, а також за відображення списків персонажів та їхньої детальної інформації.

- show_character_menu(update: Update, context: CallbackContext): Відображає головне меню персонажів з кнопками елементів.

```
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
from localization import localized_texts

async def show_character_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['anemo'],
localized_texts[user_language]['pyro']],
        [localized_texts[user_language]['cryo'],
localized_texts[user_language]['electro']],
        [localized_texts[user_language]['geo'],
localized_texts[user_language]['hydro']],
        [localized_texts[user_language]['dendro'],
localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['select_character_element'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'character_elements'
```

- Функції відображення меню вибору рідкості для кожного елементу (show_anemo_rarity_menu, show_pyro_rarity_menu тощо): Відображають клавіатуру з кнопками "5★ [Елемент]", "4★ [Елемент]" та "Назад".

```
async def show_anemo_rarity_menu(update: Update, context:
CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_anemo = localized_texts[user_language]['anemo']
    keyboard = [
        [f"5★ {localized_anemo}"],
        [f"4★ {localized_anemo}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
```

```
        await     context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_rarity'].format(element=localized
_anemo), reply_markup=reply_markup)
        context.user_data['current_menu'] = 'anemo_rarity'
```

Аналогічні функції існують для pyro, cryo, electro, geo, hydro та dendro.

- Функції відображення списку персонажів за елементом та рідкістю (show_anemo_4_star_characters, show_pyro_5_star_characters тощо): Відображають клавіатуру зі списком імен персонажів обраного елементу та рідкості, а також кнопкою "Назад [Елемент] ".

```
from data import fixed_character_order

async def show_anemo_4_star_characters(update: Update, context:
CallbackContext):
        user_language = context.user_data.get('language', 'uk')
        localized_anemo = localized_texts[user_language]['anemo']
        anemo_4_star       =       fixed_character_order.get(user_language,
{}).get(localized_anemo, {}).get('4', [])
        # ... (формування клавіатури зі списком персонажів) ...
        await context.bot.send_message(...)
        context.user_data['current_menu'] = 'anemo_4_star_list'
```

Аналогічні функції існують для всіх комбінацій елементів та рідкостей. fixed_character_order - це структура даних (ймовірно, словник), що містить відсортовані списки персонажів за мовою, елементом та рідкістю.

- show_character_info(update: Update, context: CallbackContext, requested_name: str): Відображає детальну інформацію про обраного персонажа, включаючи фотографії та опис.

```
import os
from telegram import InputMediaPhoto

# ... (константа CHARACTERS_PHOTO_FOLDER) ...

async def show_character_info(update: Update, context: CallbackContext,
requested_name: str):
        # ... (пошук інформації про персонажа в character_data) ...
        if found_character_name and 'photo_folder' in character_info:
            # ... (логіка надсилання фотографій групами або однією) ...
```

```
        else:
            await update.message.reply_text(...)
```

- handle_character_selection_from_list(update: Update, context: CallbackContext, element: str, rarity: str): Обробляє вибір персонажа зі списку. Ця функція викликається обробниками текстових повідомлень.

```
    async def handle_character_selection_from_list(update: Update, context: CallbackContext, element: str, rarity: str):
        user_language = context.user_data.get('language', 'uk')
        localized_element = context.user_data.get('selected_element_localized')
        # ... (отримання списку персонажів) ...
        user_message = update.message.text.strip()
        if user_message in characters:
            await show_character_info(update, context, user_message)
        elif user_message.lower() == localized_texts[user_language].get('back', "").lower():
            # ... (логіка повернення до меню вибору рідкості) ...
        else:
            await update.message.reply_text(...)
```

2. Обробники текстових повідомлень (у головному файлі бота або окремому файлі обробників):

Ці функції реєструються за допомогою MessageHandler та відповідають за обробку натискань на клавіатурні кнопки в меню персонажів.

- handle_characters_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Персонажі" в головному меню, викликаючи character_handlers.show_character_menu.
- handle_element_selection(update: Update, context: CallbackContext): Обробляє вибір елементу, викликаючи відповідну функцію відображення меню рідкості (наприклад, character_handlers.show_anemo_rarity_menu).
- handle_anemo_rarity_selection(update: Update, context: CallbackContext) та аналогічні для інших елементів: Обробляють вибір рідкості,

викликаючи функції відображення списку персонажів (наприклад, character_handlers.show_anemo_4_star_characters).

- handle_anemo_4_star_character_selection(update: Update, context: CallbackContext) та аналогічні для всіх списків персонажів: Обробляють вибір персонажа зі списку, викликаючи character_handlers.handle_character_selection_from_list з відповідними параметрами елементу та рідкості.
- Функції обробки кнопки "Назад" на різних рівнях меню (handle_back_characters_menu, handle_back_anemo_rarity, handle_back_anemo_list тощо): Забезпечують повернення до попередніх меню.

```
async def handle_back_anemo_list(update: Update, context: CallbackContext):
    await character_handlers.show_anemo_rarity_menu(update, context)
```

3. Дані про персонажів (файл data.py або окремі файли):

- character_data: Словник, що містить інформацію про кожного персонажа (назва різними мовами, елемент, зброя, рідкість, опис, шлях до папки з фотографіями тощо).

```
character_data = {
    "Venti": {
        "name": {"uk": "Венті", "en": "Venti"},
        "element": {"uk": "Анемо", "en": "Anemo"},
        "weapon": {"uk": "Лук", "en": "Bow"},
        "rarity": 5,
        "description": {"uk": "Бард, що любить вино...", "en": "A wine-loving bard..."},
        "photo_folder": "venti"
    },
    # ... інші персонажі ...
}
```

- fixed_character_order: Словник, що містить відсортовані списки персонажів за мовою, елементом та рідкістю. Це визначає порядок відображення персонажів у списках.

```
    fixed_character_order = {
        "uk": {
            "Анемо": {
                "4": ["Саю", "Мандрівник (Анемо)", "Хейдзо", "Лінетт",
"Фарузан"],
                "5": ["Венті", "Джинн", "Кадзуха", "Сяо"]
            },
            "Піро": {
                "4": ["Ембер", "Беннет", "Янь Фей", "Сян Лін", "Тома"],
                "5": ["Ділюк", "Клі", "Ху Тао", "Йемія", "Арлекіно"]
            },
            # ... інші елементи ...
        },
        "en": {
            "Anemo": {
                "4": ["Sayu", "Traveler (Anemo)", "Heizou", "Lynette",
"Faruzan"],
                "5": ["Venti", "Jean", "Kazuha", "Xiao"]
            },
            # ... інші елементи ...
        }
    }
```

4. Локалізація (файл localization.py):

Файл localization.py містить словник localized_texts з перекладами всіх текстових констант бота різними мовами. Це включає назви елементів, рідкостей, кнопок "Назад", заголовків меню та повідомлень.

5. Реєстрація обробників (у головному файлі бота):

У головному файлі бота (наприклад, main.py) необхідно зареєструвати всі створені обробники текстових повідомлень, щоб бот міг реагувати на натискання клавіатурних кнопок у меню персонажів. Це робиться за допомогою application.add_handler(MessageHandler(filters.Regex(...), ...)).

Принцип роботи:

1. Користувач натискає кнопку "Персонажі" в головному меню.
2. Викликається handle_characters_button, яка відображає меню елементів (show_character_menu).

3. Користувач натискає кнопку з назвою елементу (наприклад, "Анемо").
4. Викликається handle_element_selection, яка відображає меню вибору рідкості для цього елементу (show_anemo_rarity_menu).
5. Користувач натискає кнопку з рідкістю (наприклад, "4★ Анемо").
6. Викликається відповідний обробник рідкості (handle_anemo_rarity_selection), який відображає список 4-зіркових персонажів елементу Анемо (show_anemo_4_star_characters).
7. Користувач натискає на ім'я персонажа зі списку.
8. Викликається відповідний обробник вибору персонажа (handle_anemo_4_star_character_selection), який викликає show_character_info для відображення детальної інформації про обраного персонажа.
9. На кожному кроці користувач може натиснути кнопку "Назад" для повернення до попереднього меню, що обробляється відповідними обробниками "Назад".

Цей детальний опис разом із фрагментами коду надає повне розуміння того, як реалізована обробка персонажів у Telegram-боті "Помічничок мандрівника Genshin Impact" з використанням клавіатурних кнопок. Кожен компонент відіграє важливу роль у забезпеченні зручної та інформативної взаємодії користувача з даними про персонажів гри.

3.2.2 Обробка інформації про зброю

Обробка інформації про зброю в боті реалізована за аналогічним принципом до обробки персонажів, використовуючи багаторівневе меню, навігацію за типами та рідкістю, виведення списку зброї та надання базової інформації про обраний предмет.

1. Модуль weapon_handlers (файл, де визначені надані функції):

Цей модуль містить функції для відображення меню зброї та навігації по ньому.

- get_weapon_data():

```
def get_weapon_data():
    return {
        'клятва_вільного_вітру': {
            'name': {
                'uk': 'Клятва вільного вітру',
                'en': 'Freedom-Sworn',
            },
            'type': {
                'uk': 'Одноручна зброя',
                'en': 'Sword',
            },
            'rarity': '5',
            'folder_name': 'freedom-sworn',
            # Можете додати інші характеристики зброї, якщо потрібно
        },
        # Додайте сюди дані про іншу зброю
    }
```

Призначення: Ця функція повертає словник, що містить інформацію про всю зброю в грі. Ключами верхнього рівня є внутрішні ідентифікатори зброї (наприклад, 'клятва_вільного_вітру'), а значеннями - словники з детальною інформацією, включаючи назви різними мовами, тип, рідкість та назву папки для зображень (якщо буде реалізовано показ фото).

- show_weapon_menu(update: Update, context: CallbackContext): Відображає головне меню зброї з кнопками типів зброї.

```
async def show_weapon_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
[KeyboardButton(localized_texts[user_language]['sword'])],
[KeyboardButton(localized_texts[user_language]['claymore'])],
[KeyboardButton(localized_texts[user_language]['polearm'])],
[KeyboardButton(localized_texts[user_language]['bow'])],
[KeyboardButton(localized_texts[user_language]['catalyst'])],
[KeyboardButton(localized_back)] # Загальна кнопка "Назад"
]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_weapon_type'
], reply_markup=reply_markup)
```

```
        context.user_data['current_menu'] = 'weapon_types'
```

Призначення: Ця асинхронна функція викликається для відображення початкового меню вибору типу зброї.

Логіка: Отримує мову користувача, формує клавіатуру з кнопками, що відповідають типам зброї (меч, двуручний меч, спис, лук, каталізатор), а також кнопку "Назад" для повернення до головного меню. Надсилає повідомлення з цією клавіатурою та встановлює стан 'current_menu' у context.user_data для відстеження поточного розділу.

- show_sword_rarity_menu(update: Update, context: CallbackContext) та аналогічні для інших типів зброї: Відображають меню вибору рідкості для обраного типу зброї.

```
    async def show_sword_rarity_menu(update: Update, context:
CallbackContext):
        user_language = context.user_data.get('language', 'uk')
        keyboard = [
            [KeyboardButton(f"5★
{localized_texts[user_language]['sword']}")],
            [KeyboardButton(f"4★
{localized_texts[user_language]['sword']}")],
            [KeyboardButton(localized_back_sword)]
        ]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        await
update.message.reply_text(localized_texts[user_language]['choose_rarity'],
reply_markup=reply_markup)
        context.user_data['current_menu'] = 'sword_rarity'
```

Призначення: Ці функції (існують аналогічні для claymore, polearm, bow, catalyst) викликаються після вибору типу зброї.

Логіка: Отримують мову користувача, формують клавіатуру з кнопками для вибору рідкості (5★ та 4★) для обраного типу зброї, а також кнопкою "Назад" (специфічною для цього типу зброї, наприклад, "Назад 🚀"). Надсилають повідомлення з цією клавіатурою та оновлюють стан 'current_menu'.

- show_sword_5_star_weapons(update: Update, context: CallbackContext) та аналогічні для інших типів та рідкостей: Відображають список зброї обраного типу та рідкості.

```
async def show_sword_5_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    sword_weapons = [weapon for _, weapon in get_weapon_data().items()
                     if weapon['type'].get('uk', '').lower() == localized_texts[user_language]['sword'].lower() and weapon['rarity'] == '5']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language, weapon['name']['uk']))] for weapon in sword_weapons]
    keyboard.append([KeyboardButton(localized_back_sword)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'sword_5_star_list'
```

Призначення: Ці функції (існують аналогічні для всіх комбінацій типів зброї та рідкостей) викликаються після вибору рідкості.

Логіка: Отримують мову користувача, фільтрують дані про зброю з get_weapon_data() за обраним типом та рідкістю, формують клавіатуру зі списком назв зброї (локалізованих, якщо доступно), а також кнопкою "Назад" до меню вибору рідкості для цього типу зброї. Надсилають повідомлення зі списком та оновлюють стан 'current_menu'.

- handle_weapon_selection_from_list(update: Update, context: CallbackContext, weapon_type: str, rarity: str, weapon_name: str): Обробляє вибір конкретної зброї зі списку.

```
async def handle_weapon_selection_from_list(update: Update, context: CallbackContext, weapon_type: str, rarity: str, weapon_name: str):
    user_language = context.user_data.get('language', 'uk')
    weapon_data = get_weapon_data()

    selected_weapon_info = None
    for key, data in weapon_data.items():
```

```
            if      (data['name'].get(user_language,      '').lower()      ==
weapon_name.lower() and
                    data['type'].get('uk',            '').lower()            ==
localized_texts[user_language].get(weapon_type, weapon_type).lower() and
                    data['rarity'] == rarity):
                    selected_weapon_info = data
                    break

        if selected_weapon_info:
            caption      =      f"{localized_texts[user_language]['name']}:
{selected_weapon_info['name'].get(user_language, weapon_name)}\n"
            caption      +=      f"{localized_texts[user_language]['type']}:
{selected_weapon_info['type'].get(user_language, '')}\n"
            caption     +=     f"{localized_texts[user_language]['rarity']}:
{selected_weapon_info['rarity']}★\n"
            await update.message.reply_text(caption)
            # Тут можна додати логіку для показу фото зброї
        else:
            await
update.message.reply_text(localized_texts[user_language]['weapon_not_found'])
```

Призначення: Ця асинхронна функція викликається, коли користувач натискає на назву зброї у списку.

Логіка: Отримує мову користувача, дані про зброю, тип, рідкість та назву обраної зброї. Здійснює пошук інформації про цю зброю в get_weapon_data(). Якщо зброю знайдено, формує та надсилає базову інформацію про неї (назва, тип, рідкість). У випадку відсутності інформації надсилає відповідне повідомлення.

2. Обробники текстових повідомлень (у головному файлі бота або окремому файлі обробників):

Ці обробники відповідають за реакцію бота на натискання кнопок меню зброї.

- handle_weapons_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Зброя" в головному меню, викликаючи weapon_handlers.show_weapon_menu.

- Обробники вибору типу зброї (handle_sword_selection, handle_claymore_selection тощо): Викликають відповідні функції відображення меню рідкості (наприклад, weapon_handlers.show_sword_rarity_menu).

```
async def handle_sword_selection(update: Update, context: CallbackContext):
    await weapon_handlers.show_sword_rarity_menu(update, context)
```

- Обробники вибору рідкості для кожного типу зброї (handle_sword_5_star_selection, handle_sword_4_star_selection тощо): Викликають функції відображення списку зброї (наприклад, weapon_handlers.show_sword_5_star_weapons).

```
async def handle_sword_5_star_selection(update: Update, context: CallbackContext):
    await weapon_handlers.show_sword_5_star_weapons(update, context)
```

- Обробники вибору конкретної зброї зі списку (handle_selected_sword, handle_selected_claymore тощо): Викликають weapon_handlers.handle_weapon_selection_from_list з відповідними параметрами.

```
async def handle_selected_sword(update: Update, context: CallbackContext):
    selected_weapon_name = update.message.text
    await weapon_handlers.handle_weapon_selection_from_list(update, context, 'sword', context.user_data.get('current_rarity', 'unknown'), selected_weapon_name)
    context.user_data.pop('current_rarity', None) # Очищаємо тимчасову рідкість
```

Важливо: Для коректної роботи обробників вибору зброї зі списку вам може знадобитися зберігати поточний тип та рідкість зброї в context.user_data на попередніх кроках.

- Обробники кнопки "Назад" на різних рівнях меню (handle_back_to_weapon_menu, handle_back_to_sword_rarity тощо): Забезпечують повернення до попередніх меню.

```
async def handle_back_to_sword_rarity(update: Update, context: CallbackContext):
    await weapon_handlers.show_sword_rarity_menu(update, context)
```

3. Локалізація (файл localization.py):

Файл localization.py повинен містити переклади для всіх текстових елементів, пов'язаних зі зброєю: назви типів зброї (меч, двуручний меч тощо), слова "Рідкість", "Назад" (з відповідними іконками), "Виберіть тип зброї", "Виберіть рідкість", "Виберіть зброю", "Зброю не знайдено", а також назви характеристик зброї ("Назва", "Тип", "Рідкість").

4. Реєстрація обробників (у головному файлі бота):

У головному файлі бота необхідно зареєструвати всі обробники текстових повідомлень, пов'язані з розділом зброї, використовуючи application.add_handler(MessageHandler(filters.Regex(...), ...)). Фільтри (filters.Regex) повинні відповідати тексту кнопок меню.

Принцип роботи:

1. Користувач натискає кнопку "Зброя" в головному меню.
2. Викликається handle_weapons_button, яка відображає меню типів зброї (show_weapon_menu).
3. Користувач натискає кнопку з типом зброї (наприклад, "Одноручна зброя").
4. Викликається відповідний обробник типу зброї (handle_sword_selection), який відображає меню вибору рідкості для цього типу (show_sword_rarity_menu).
5. Користувач натискає кнопку з рідкістю (наприклад, "5★ Одноручна зброя").
6. Викликається відповідний обробник рідкості (handle_sword_5_star_selection), який відображає список 5-зіркових одноручних мечів (show_sword_5_star_weapons).
7. Користувач натискає на назву зброї зі списку.

8. Викликається обробник вибору зброї (handle_selected_sword), який викликає handle_weapon_selection_from_list для відображення інформації про обрану зброю.
9. На кожному кроці користувач може натиснути кнопку "Назад" для повернення до попереднього меню.

Ця структура забезпечує логічну та зручну навігацію для отримання інформації про зброю в грі. Подальшим кроком може бути додавання детальнішої інформації про кожну зброю (характеристики, пасивні навички) та відображення її зображення.

3.2.3 Обробка інформації про артефакти

Обробка інформації про артефакти в боті надає користувачам візуальне представлення наборів артефактів, згрупованих за основною характеристикою, яку вони підсилюють.

1. Константа ARTIFACTS_FOLDER:

```
ARTIFACTS_FOLDER = "D:\\Games\\дипломнаробота\\artifacts"  # Шлях до вашої папки з артефактами
```

Призначення: Ця константа зберігає абсолютний шлях до папки на сервері, де розміщені підпапки з зображеннями артефактів. Кожна підпапка повинна мати назву, що відповідає внутрішній назві категорії артефактів (наприклад, "attack", "talent_bonus").

2. Функція send_artifact_category_media:

```
import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext
from localization import localized_texts

async def send_artifact_category_media(update: Update, context: CallbackContext, category: str, caption_key: str):
    chat_id = update.message.chat.id
    media_path = os.path.join(ARTIFACTS_FOLDER, category)
    user_language = context.user_data.get('language', 'uk')

    try:
```

```
            all_files = os.listdir(media_path)
            image_files = [f for f in all_files if
f.lower().endswith(('.jpg', '.jpeg', '.png', '.gif'))]

            if not image_files:
                await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['artifact_not_found_category'].format(cat
egory=localized_texts[user_language].get(f'artifact_{category.lower().replace
(" ", "_")}', category)))
                return

            media_group = []
            for image_file in image_files:
                image_path = os.path.join(media_path, image_file)
                with open(image_path, 'rb') as photo_file:
                    # Отримуємо локалізований підпис за допомогою переданого
ключа
                    caption_text =
localized_texts[user_language].get(caption_key, "")
                    media_group.append(InputMediaPhoto(photo_file.read(),
caption=caption_text if not media_group else None))

            if media_group:
                if len(media_group) == 1:
                    await context.bot.send_photo(chat_id=chat_id,
photo=media_group[0].media, caption=media_group[0].caption)
                else:
                    await context.bot.send_media_group(chat_id=chat_id,
media=media_group)

        except FileNotFoundError:
            await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['folder_not_found'].format(folder=media_p
ath))
        except Exception as e:
            print(f"Помилка при відправці артефактів з категорії {category}:
{e}")
            await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_sending_artifacts_category'].forma
t(category=localized_texts[user_language].get(f'artifact_{category.lower().re
place(" ", "_")}', category)))
```

Призначення: Ця асинхронна функція відповідає за надсилання користувачеві зображень артефактів, що належать до певної категорії.

Аргументи:

- update: Update: Об'єкт оновлення від Telegram.
- context: CallbackContext: Контекст обробки виклику.
- category: str: Внутрішня назва категорії артефактів (наприклад, "attack"). Ця назва використовується для пошуку відповідної підпапки із зображеннями.
- caption_key: str: Ключ у словнику localized_texts, який використовується для отримання локалізованого підпису до групи зображень (наприклад, 'artifact_attack_caption').

Логіка:

1. Визначає шлях до папки з зображеннями для вказаної категорії, використовуючи ARTIFACTS_FOLDER та передану category.
2. Отримує мову користувача з context.user_data.
3. Намагається отримати список усіх файлів у папці категорії.
4. Фільтрує список, залишаючи лише файли із розширеннями зображень (.jpg, .jpeg, .png, .gif).
5. Якщо зображень у папці не знайдено, надсилає користувачеві повідомлення про це, використовуючи локалізований текст.
6. Створює список об'єктів InputMediaPhoto, кожен з яких представляє одне зображення артефакту. Підпис додається лише до першого зображення в групі для уникнення дублювання. Підпис береться з localized_texts за переданим caption_key.
7. Якщо список media_group не порожній:
    - Якщо в групі лише одне зображення, воно надсилається за допомогою context.bot.send_photo.
    - Якщо зображень більше, вони надсилаються як група медіа за допомогою context.bot.send_media_group.

8. Обробляє можливі помилки:

- FileNotFoundError: якщо папка для вказаної категорії не знайдена.
- Exception: для інших непередбачених помилок під час обробки та надсилання файлів. У випадку помилки надсилає користувачеві локалізоване повідомлення про помилку.

3. Функція show_artifacts_menu:

```
from telegram import Update, InputMediaPhoto, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
import os
from localization import localized_texts

async def show_artifacts_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['artifact_attack_button'],
localized_texts[user_language]['artifact_talent_bonus_button']],

[localized_texts[user_language]['artifact_elemental_bonus_button'],
localized_texts[user_language]['artifact_healing_button']],

[localized_texts[user_language]['artifact_elemental_mastery_button'],
localized_texts[user_language]['artifact_defensive_button']],
        [localized_texts[user_language]['artifact_other_button'],
localized_texts[user_language]['main_menu_button']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await     context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['artifact_choose_characteristic'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'artifacts_menu'
```

Призначення: Ця асинхронна функція відображає головне меню розділу "Артефакти".

Логіка:

1. Отримує мову користувача.
2. Формує клавіатуру ReplyKeyboardMarkup з кнопками, що представляють різні характеристики, які підсилюють артефакти

(наприклад, "Сила атаки", "Бонус навичок", "Бонус елементальної шкоди", "Бонус лікування", "Майстерність стихій", "Захист", "Інше"). Текст кнопок береться з localized_texts.

3. Додає кнопку "Головне меню" для повернення до основного меню бота.
4. Надсилає користувачеві повідомлення з текстом "Оберіть характеристику артефактів:" (локалізованим) та створеною клавіатурою.
5. Встановлює стан 'current_menu' у context.user_data на 'artifacts_menu' для відстеження поточного розділу.

4. Обробники текстових повідомлень (у головному файлі бота або окремому файлі обробників):

Ці обробники реагують на натискання кнопок у меню артефактів та викликають функцію send_artifact_category_media для надсилання відповідних зображень.

- handle_artifacts_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Артефакти" в головному меню, викликаючи show_artifacts_menu.
- Обробники вибору характеристики артефактів (handle_attack_artifacts, handle_talent_bonus_artifacts тощо): Викликають send_artifact_category_media з відповідною категорією та ключем для підпису.

```
async def handle_attack_artifacts(update: Update, context: CallbackContext):
    await send_artifact_category_media(update, context, 'attack', 'artifact_attack_caption')

async def handle_talent_bonus_artifacts(update: Update, context: CallbackContext):
    await send_artifact_category_media(update, context, 'talent_bonus', 'artifact_talent_bonus_caption')
```

```
# Аналогічні обробники для інших характеристик:
# elemental_bonus, healing, elemental_mastery, defensive, other
```

Кожен обробник отримує текст натиснутої кнопки, визначає відповідну внутрішню назву категорії артефактів (яка повинна співпадати з назвою підпапки в ARTIFACTS_FOLDER) та ключ для локалізованого підпису.

Викликає send_artifact_category_media з цими параметрами для надсилання зображень артефактів цієї категорії.

- handle_back_to_main_menu_from_artifacts(update: Update, context: CallbackContext): Обробляє натискання кнопки "Головне меню" в меню артефактів, повертаючи користувача до основного меню (викликаючи функцію start).

5. Локалізація (файл localization.py):

Файл localization.py повинен містити переклади для всіх текстових елементів, пов'язаних з артефактами:

- Назви кнопок характеристик артефактів (artifact_attack_button, artifact_talent_bonus_button, artifact_elemental_bonus_button, artifact_healing_button, artifact_elemental_mastery_button, artifact_defensive_button, artifact_other_button).
- Текст повідомлення "Оберіть характеристику артефактів:" (artifact_choose_characteristic).
- Повідомлення про відсутність артефактів у категорії (artifact_not_found_category).
- Повідомлення про помилку надсилання артефактів (error_sending_artifacts_category).
- Підписи до груп зображень артефактів (artifact_attack_caption, artifact_talent_bonus_caption тощо). Ці підписи повинні описувати, яку основну характеристику підсилюють артефакти в цій категорії.

6. Структура папок з зображеннями:

У папці, шлях до якої вказано в ARTIFACTS_FOLDER, повинні існувати підпапки з назвами, що відповідають внутрішнім назвам категорій артефактів (наприклад, attack, talent_bonus, elemental_bonus тощо). У кожній з цих підпапок повинні знаходитися зображення артефактів, що належать до цієї категорії.

Принцип роботи:

1. Користувач натискає кнопку "Артефакти" в головному меню.
2. Викликається handle_artifacts_button, яка відображає меню вибору характеристик артефактів (show_artifacts_menu).
3. Користувач натискає кнопку з назвою характеристики (наприклад, "Сила атаки").
4. Викликається відповідний обробник (handle_attack_artifacts), який викликає send_artifact_category_media з категорією 'attack' та ключем для підпису 'artifact_attack_caption'.
5. Функція send_artifact_category_media знаходить папку attack у ARTIFACTS_FOLDER, зчитує всі зображення, формує групу медіа та надсилає їх користувачеві з відповідним підписом.
6. Якщо користувач натискає кнопку "Головне меню", викликається handle_back_to_main_menu_from_artifacts, і користувач повертається до основного меню бота.

Ця структура дозволяє користувачам швидко переглядати набори артефактів, згруповані за їхньою основною функціональною спрямованістю, забезпечуючи візуальне представлення без необхідності окремого опису кожного набору. Детальний опис кожного набору артефактів (бонуси за 2 та 4 предмети) може бути доданий як розширення функціональності у вигляді окремого меню або підпису до зображень.

3.2.4 Обробка інформації про ворогів

Розділ "Вороги" в боті призначений для надання користувачам інформації про різних противників у грі, класифікованих за їхнім типом.

Основна функціональність включає відображення меню типів ворогів, списків ворогів кожного типу та надсилання зображень обраних ворогів.

1. Константа BATTLE_PHOTO_FOLDER:

```
BATTLE_PHOTO_FOLDER = 'combat'
```

Призначення: Ця константа визначає назву головної папки, в якій зберігаються підпапки з фотографіями, пов'язаними з бойовим контентом, включаючи ворогів.

2. Функція send_battle_photos_keyboard (може бути використана і для ворогів, але краще мати окрему):

Хоча надана функція send_battle_photos_keyboard є досить універсальною для надсилання фотографій з підпапок, для кращої організації та можливої подальшої специфічної логіки для ворогів, розглянемо окрему функцію send_enemy_photos.

3. Функція get_enemies_data():

```
def get_enemies_data():
    return {
        'bosses': {
            'uk': ['Андріус', 'Аждаха', 'Синьйора'],
            'en': ['Andrius', 'Azhdaha', 'Signora'],
        },
        'common': {
            'uk': ['Хілічурл-стрілець', 'Велика слизь', 'Кам\'яна
шкіра'],
            'en': ['Hilichurl Shooter', 'Large Slime', 'Stonehide
Lawachurl'],
        },
        'elite': {
            'uk': ['Анемо борець', 'Кріо вісник Фатуї', 'Електро цицін
маг'],
            'en': ['Anemo Samachurl', 'Cryo Cicin Mage', 'Electro Cicin
Mage'],
        },
    }
```

Призначення: Ця функція повертає словник, що містить інформацію про різних типів ворогів та їхні назви різними мовами. Структура словника дозволяє легко отримувати списки ворогів за категоріями ('bosses', 'common', 'elite') та мовою користувача.

4. Функція show_enemies_menu:

```
async def show_enemies_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['bosses']],
        [localized_texts[user_language]['common_enemies']],
        [localized_texts[user_language]['elite_enemies']],
        [localized_texts[user_language]['back_to_combat_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['enemies_menu_title'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'enemies_menu'
```

Призначення: Ця асинхронна функція відображає головне меню розділу "Вороги".

Логіка:

1. Отримує мову користувача.
2. Формує клавіатуру ReplyKeyboardMarkup з кнопками, що представляють різні типи ворогів (боси, звичайні, елітні). Текст кнопок береться з localized_texts.
3. Додає кнопку "Назад до меню битви" для повернення до попереднього меню "Битва".
4. Надсилає користувачеві повідомлення з текстом "Оберіть тип ворогів:" (локалізованим) та створеною клавіатурою.
5. Встановлює стан 'current_menu' у context.user_data на 'enemies_menu' для відстеження поточного розділу.

5. Функція show_enemies_list:

```
    async def show_enemies_list(update: Update, context: CallbackContext,
enemy_type: str):
        user_language = context.user_data.get('language', 'uk')
        enemies_data = get_enemies_data() # Функція, яка повертає словник з
даними про ворогів
        enemy_list = enemies_data.get(enemy_type, {}).get(user_language, [])
        if not enemy_list:
            await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['no_enemies_found'].format(enemy_type=loc
alized_texts[user_language].get(enemy_type, enemy_type)))
            return

        keyboard = [[enemy] for enemy in enemy_list]

keyboard.append([localized_texts[user_language]['back_to_enemies_menu']])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        await     context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['select_enemy'],
reply_markup=reply_markup)
        context.user_data['current_menu'] = f'enemies_list_{enemy_type}'
```

Призначення: Ця асинхронна функція відображає список ворогів обраного типу.

Аргументи:

- update: Update: Об'єкт оновлення від Telegram.
- context: CallbackContext: Контекст обробки виклику.
- enemy_type: str: Внутрішня назва типу ворогів ('bosses', 'common', 'elite').

Логіка:

1. Отримує мову користувача.
2. Отримує список ворогів для вказаного enemy_type з get_enemies_data().
3. Якщо список порожній, надсилає повідомлення про відсутність ворогів цього типу.
4. Формує клавіатуру ReplyKeyboardMarkup зі списком імен ворогів (кожен ворог - окрема кнопка).
5. Додає кнопку "Назад до меню ворогів".

6. Надсилає користувачеві повідомлення з текстом "Оберіть ворога:" (локалізованим) та створеною клавіатурою.
7. Встановлює стан 'current_menu' у context.user_data для відстеження поточного списку ворогів.

6. Функція send_enemy_photos:

```
    async def send_enemy_photos(update: Update, context: CallbackContext, enemy_type: str, enemy_name: str):
        chat_id = update.effective_chat.id
        full_path = os.path.join(BATTLE_PHOTO_FOLDER, 'enemies', enemy_type.lower(), enemy_name.lower()) # Припускаємо таку структуру папок
        user_language = context.user_data.get('language', 'uk')

        try:
            all_files = os.listdir(full_path)
            image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.webp'))]

            if image_files:
                media_group = []
                for image_file in image_files:
                    image_path = os.path.join(full_path, image_file)
                    try:
                        with open(image_path, 'rb') as photo_file:
                            caption = localized_texts[user_language].get(enemy_name.lower(), enemy_name) # Підпис - ім'я ворога

                            media_group.append(InputMediaPhoto(photo_file.read(), caption=caption if not media_group else None))
                    except Exception as e:
                        print(f"Помилка при читанні файлу {image_path}: {e}")
                        await context.bot.send_message(chat_id=chat_id, text=localized_texts[user_language]['error_reading_photo'].format(filename=image_file))
                        return

                if media_group:
                    for i in range(0, len(media_group), 10):
                        chunk = media_group[i:i + 10]
```

```
                        try:
                            await
context.bot.send_media_group(chat_id=chat_id, media=chunk)
                        except Exception as e:
                            print(f"Помилка при відправці медіагрупи ворога
'{enemy_name}': {e}")
                            await  context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_sending_photos'].format(category=e
nemy_name))
                            return
                else:
                    await          context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_for_enemy'].format(enemy=local
ized_texts[user_language].get(enemy_name.lower(), enemy_name)))
            else:
                await              context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_for_enemy'].format(enemy=local
ized_texts[user_language].get(enemy_name.lower(), enemy_name)))

        except FileNotFoundError:
            await                  context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['folder_not_found'].format(folder=enemy_n
ame, path=full_path))
        except Exception as e:
            print(f"Помилка при обробці папки ворога '{enemy_name}': {e}")
            await                  context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_processing_folder'].format(folder=
enemy_name))
```

Призначення: Ця асинхронна функція надсилає користувачеві фотографії обраного ворога.

Аргументи:

- update: Update: Об'єкт оновлення від Telegram.
- context: CallbackContext: Контекст обробки виклику.
- enemy_type: str: Тип ворога ('bosses', 'common', 'elite').
- enemy_name: str: Назва обраного ворога.

Логіка:

1. Визначає шлях до папки з фотографіями ворога, використовуючи BATTLE_PHOTO_FOLDER, тип ворога та його назву (припускається відповідна структура папок).
2. Отримує мову користувача.
3. Зчитує всі файли зображень у папці ворога.
4. Формує групу медіа з фотографій, використовуючи назву ворога як підпис до першого зображення.
5. Надсилає групу фотографій користувачеві.
6. Обробляє помилки, пов'язані з відсутністю папки або фотографій.

7. Обробники текстових повідомлень (у головному файлі бота або окремому файлі обробників):

- handle_combat_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Битва" в головному меню, викликаючи show_combat_menu.
- handle_enemies_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Вороги" в меню "Битва", викликаючи show_enemies_menu.
- Обробники вибору типу ворогів (handle_bosses_selection, handle_common_enemies_selection, handle_elite_enemies_selection): Викликають show_enemies_list з відповідним типом ворогів.

```
async def handle_bosses_selection(update: Update, context:
CallbackContext):
    await show_enemies_list(update, context, 'bosses')

async def handle_common_enemies_selection(update: Update, context:
CallbackContext):
    await show_enemies_list(update, context, 'common')

async def handle_elite_enemies_selection(update: Update, context:
CallbackContext):
    await show_enemies_list(update, context, 'elite')
```

- Обробники вибору конкретного ворога зі списку (handle_selected_boss, handle_selected_common_enemy тощо): Викликають send_enemy_photos з відповідним типом та назвою ворога.

```
async def handle_selected_boss(update: Update, context:
CallbackContext):
    selected_enemy_name = update.message.text
    await send_enemy_photos(update, context, 'bosses',
selected_enemy_name)

# Аналогічні обробники для звичайних та елітних ворогів
```

- Обробники кнопки "Назад" (handle_back_to_combat_menu_from_enemies_menu, handle_back_to_enemies_menu_from_list): Забезпечують повернення до попередніх меню.

```
async def handle_back_to_combat_menu_from_enemies_menu(update: Update,
context: CallbackContext):
    await show_combat_menu(update, context)

async def handle_back_to_enemies_menu_from_list(update: Update, context:
CallbackContext):
    await show_enemies_menu(update, context)
```

8. Локалізація (файл localization.py):

Файл localization.py повинен містити переклади для всіх текстових елементів, пов'язаних з ворогами:

- Назва розділу "Битва" (combat_title).
- Назва кнопки "Вороги" (enemies).
- Назви типів ворогів (bosses, common_enemies, elite_enemies).
- Текст повідомлення "Оберіть тип ворогів:" (enemies_menu_title).
- Текст повідомлення "Оберіть ворога:" (select_enemy).
- Текст повідомлення про відсутність ворогів певного типу (no_enemies_found).
- Текст повідомлення про відсутність фотографій ворога (no_photos_for_enemy).

- Назви кнопок "Назад" (back_to_combat_menu, back_to_enemies_menu).
- Повідомлення про помилки читання/відправки файлів та обробки папок.

9. Структура папок з зображеннями:

У папці, шлях до якої вказано в BATTLE_PHOTO_FOLDER, повинна існувати підпапка enemies. У ній повинні бути підпапки з назвами типів ворогів у нижньому регістрі (bosses, common, elite). У кожній з цих підпапок повинні знаходитися підпапки з назвами ворогів у нижньому регістрі (латиницею, якщо назви файлів англійською). У цих папках повинні бути зображення відповідних ворогів.

Принцип роботи:

1. Користувач переходить до меню "Битва" та натискає кнопку "Вороги".
2. Викликається show_enemies_menu, відображається меню типів ворогів.
3. Користувач обирає тип ворогів (наприклад, "Боси").
4. Викликається відповідний обробник (handle_bosses_selection), який викликає show_enemies_list для відображення списку босів.
5. Користувач натискає на ім'я ворога зі списку (наприклад, "Андріус").
6. Викликається відповідний обробник (handle_selected_boss), який викликає send_enemy_photos для надсилання фотографій Андріуса.
7. Функція send_enemy_photos знаходить відповідну папку з фотографіями та надсилає їх користувачеві.
8. На кожному кроці користувач може використовувати кнопки "Назад" для повернення до попередніх меню.

Ця структура забезпечує зручну навігацію та візуальне представлення ворогів у грі. Подальшим розширенням може бути додавання детальної інформації про кожного ворога (атаки, слабкості, дроп тощо).

3.2.5 Обробка меню "Цікавинки"

Розділ "Цікавинки" в боті призначений для надання користувачам різноманітного додаткового контенту, який не входить до основних категорій. Це може включати календар днів народжень персонажів, інформацію про

промокоди, неофіційний контент (арти, банери, локації), випадкові зображення та функцію зміни мови.

1. Константа PHOTO_PATH:

```
PHOTO_PATH = r'D:\Games\дипломнаробота\photos\photo_2025-02-11_22-28-21.jpg'
```

Призначення: Ця константа зберігає шлях до фотографії, яка може використовуватися (наприклад, як заставка) в деяких підрозділах меню "Цікавинки" (в даному випадку, для календаря днів народжень).

2. Функція show_curiosities_menu:

```
async def show_curiosities_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['birthday_calendar_button'], localized_texts[user_language]['promocodes']],
        [localized_texts[user_language]['unofficial'], localized_texts[user_language]['pictures']],

[localized_texts[user_language]['change_language'],localized_texts[user_language]['back_to_main_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=localized_texts[user_language]['curiosities_title'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'curiosities_menu'
```

Призначення: Ця асинхронна функція відображає головне меню розділу "Цікавинки".

Логіка:

1. Отримує мову користувача.
2. Формує клавіатуру ReplyKeyboardMarkup з кнопками, що ведуть до різних підрозділів:
   - Календар днів народжень (birthday_calendar_button).
   - Промокоди (promocodes).

- Неофіційний контент (unofficial).
- Випадкові зображення (pictures).
- Зміна мови (change_language).
- Повернення до головного меню (back_to_main_menu).

3. Надсилає користувачеві повідомлення з текстом "Цікавинки:" (локалізованим) та створеною клавіатурою.
4. Встановлює стан 'current_menu' у context.user_data на 'curiosities_menu' для відстеження поточного розділу.

3. Функція show_unofficial_keyboard:

```
UNOFFICIAL_FOLDER = "D:\\Games\\дипломнаробота\\unofficial"  # Замініть на свій шлях
BANNERS_SUBFOLDER = "banners"

async def show_unofficial_keyboard(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['characters_button'], localized_texts[user_language]['future_banners_button']],
        [localized_texts[user_language]['locations_button'], localized_texts[user_language]['curiosities_button']],
        [localized_texts[user_language]['back_to_curiosities_text']],  # Використовуйте існуючий текст "Назад"
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True, one_time_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['unofficial_menu_caption'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'unofficial_menu'
```

Призначення: Ця асинхронна функція відображає підменю "Неофіційний контент".

Логіка:

1. Отримує мову користувача.

2. Формує клавіатуру з кнопками, що ведуть до різних видів неофіційного контенту (персонажі, майбутні банери, локації, цікавинки).
3. Додає кнопку "Назад" для повернення до меню "Цікавинки".
4. Надсилає користувачеві повідомлення з текстом "Неофіційний контент:" (локалізованим) та створеною клавіатурою.
5. Встановлює стан 'current_menu' у context.user_data на 'unofficial_menu'.

4. Функція send_unofficial_media:

```
UNOFFICIAL_FOLDER = "D:\\Games\\дипломнаробота\\unofficial"  # Замініть на свій шлях

async def send_unofficial_media(update: Update, context: CallbackContext, subfolder: str, caption_key: str):
    chat_id = update.message.chat.id
    media_path = os.path.join(UNOFFICIAL_FOLDER, subfolder)
    user_language = context.user_data.get('language', 'uk') # Отримуємо мову користувача

    try:
        all_files = os.listdir(media_path)
        image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg', '.png', '.gif'))]

        if not image_files:
            await context.bot.send_message(chat_id=chat_id, text=f"У папці 'unofficial/{subfolder}' не знайдено зображень.")
            return

        media_group = []
        for image_file in image_files:
            image_path = os.path.join(media_path, image_file)
            with open(image_path, 'rb') as photo_file:
                # Отримуємо локалізований підпис за допомогою переданого ключа
                caption_text = localized_texts[user_language].get(caption_key, "")
```

```
                    media_group.append(InputMediaPhoto(photo_file.read(),
caption=caption_text if not media_group else None))

            if media_group:
                if len(media_group) == 1:
                    await            context.bot.send_photo(chat_id=chat_id,
photo=media_group[0].media, caption=media_group[0].caption)
                else:
                    await      context.bot.send_media_group(chat_id=chat_id,
media=media_group)

        except FileNotFoundError:
            await   context.bot.send_message(chat_id=chat_id,   text=f"Папку
'{media_path}' не знайдено.")
        except Exception as e:
            print(f"Помилка при відправці неофіційних {subfolder}: {e}")
            await  context.bot.send_message(chat_id=chat_id,  text=f"Виникла
помилка при відображенні неофіційних {subfolder}.")
```

Призначення: Ця асинхронна функція надсилає користувачеві зображення з вказаної підпапки в папці UNOFFICIAL_FOLDER.

Аргументи:

- update: Update: Об'єкт оновлення від Telegram.
- context: CallbackContext: Контекст обробки виклику.
- subfolder: str: Назва підпапки в UNOFFICIAL_FOLDER (наприклад, "characters", "banners").
- caption_key: str: Ключ у localized_texts для отримання підпису до групи зображень.

Логіка: Аналогічно до send_artifact_category_media та send_enemy_photos, але працює з папкою неофіційного контенту.

5. Функції для відображення неофіційного контенту (unofficial_characters, unofficial_banners, unofficial_locations, unofficial_curiosities):

```
async def unofficial_characters(update: Update, context:
CallbackContext):
```

```
    await send_unofficial_media(update, context, "characters",
'FUTURE_CAPTION')

    async def unofficial_banners(update: Update, context: CallbackContext):
    await send_unofficial_media(update, context, "banners", 'FUTURE_CAPTION')

    async def unofficial_locations(update: Update, context: CallbackContext):
    await send_unofficial_media(update, context, "locations",
'FUTURE_CAPTION')

    async def unofficial_curiosities(update: Update, context:
CallbackContext):
    print("Функція unofficial_curiosities викликана!")
    await send_unofficial_media(update, context, "curiosities",
'FUTURE_CAPTION')
```

Призначення: Ці асинхронні функції викликають send_unofficial_media з відповідними назвами підпапок та ключами для підписів.

6. Функція get_curiosities_keyboard (дублікат show_curiosities_menu, краще використовувати останню):

```
    def get_curiosities_keyboard(user_language):
        keyboard = [
            [localized_texts[user_language]['birthday_calendar_button'],
localized_texts[user_language]['promocodes']],
            [localized_texts[user_language]['unofficial'],
localized_texts[user_language]['pictures']],

[localized_texts[user_language]['your_stats'],localized_texts[user_language][
'change_language']],

[KeyboardButton(localized_texts[user_language]['back_to_main_menu'])],
        ]
        return ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
```

Призначення: Повертає об'єкт ReplyKeyboardMarkup для меню "Цікавинки". Краще використовувати show_curiosities_menu для надсилання повідомлення з цією клавіатурою.

7. Функція send_random_image_with_author:

```
    IMAGE_FOLDER = 'D:\\Games\\дипломнаробота\\random_photos'
```

```
AUTHOR_DELIMITER = ";"  # Символ, що відокремлює ім'я автора в назві файлу

async def send_random_image_with_author(update: Update, context: CallbackContext):
    # ... (логіка надсилання випадкового зображення з папки IMAGE_FOLDER та обробка імені автора з назви файлу) ...
```

Призначення: Ця асинхронна функція надсилає випадкове зображення з папки IMAGE_FOLDER. Намагається визначити автора зображення з назви файлу, використовуючи AUTHOR_DELIMITER або стандартні префікси ("photo", "unknown").

8. Функція show_birthday_calendar:

```
async def show_birthday_calendar(update: Update, context: CallbackContext):
    # ... (логіка відображення календаря днів народжень з можливістю підписки/відписки) ...
```

- Призначення: Ця асинхронна функція відображає інформацію про календар днів народжень персонажів та надає кнопки для підписки/відписки на розсилку днів народжень. Використовує InlineKeyboardMarkup для кнопок підписки/відписки.

9. Функція show_promocodes:

```
async def show_promocodes(update: Update, context: CallbackContext):
    # ... (логіка відображення інформації про промокоди з можливістю підписки/відписки) ...
```

Призначення: Ця асинхронна функція відображає інформацію про промокоди та надає кнопки для підписки/відписки на розсилку промокодів. Також використовує InlineKeyboardMarkup.

10. Функція handle_callback_query:

```
async def handle_callback_query(update: Update, context: CallbackContext):
    # ... (обробка натискань на інлайн-кнопки підписки/відписки для календаря та промокодів) ...
```

Призначення: Ця асинхронна функція обробляє натискання на інлайн-кнопки, що використовуються в show_birthday_calendar та show_promocodes для керування підписками.

11. Функція language_command:

```
async def language_command(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        ['Українська'],
        ['English']
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_language'],
reply_markup=reply_markup)
```

Призначення: Ця асинхронна функція відображає клавіатуру для вибору мови бота. Викликається при натисканні відповідної кнопки в меню "Цікавинки".

12. Функція set_language:

```
async def set_language(update: Update, context: CallbackContext):
    language = 'uk' if update.message.text == 'Українська' else 'en'
    context.user_data['language'] = language
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[language]['language_changed'])
    await start(update, context)
```

Призначення: Ця асинхронна функція встановлює обрану користувачем мову в context.user_data та надсилає підтвердження. Після цього повертає користувача до головного меню.

13. Обробники текстових повідомлень (у головному файлі бота або окремому файлі обробників):

- handle_curiosities_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Цікавинки" в головному меню, викликаючи show_curiosities_menu.
- Обробники кнопок меню "Цікавинки" (handle_birthday_calendar, handle_promocodes, handle_unofficial, handle_pictures, handle_change_language): Викликають відповідні функції для відображення підрозділів меню.

```
async def handle_birthday_calendar(update: Update, context:
CallbackContext):
    await show_birthday_calendar(update, context)

async def handle_promocodes(update: Update, context: CallbackContext):
    await show_promocodes(update, context)

async def handle_unofficial(update: Update, context: CallbackContext):
    await show_unofficial_keyboard(update, context)

async def handle_pictures(update: Update, context: CallbackContext):
    await send_random_image_with_author(update, context)

async def handle_change_language(update: Update, context:
CallbackContext):
    await language_command(update, context)
```

- Обробники кнопок підменю "Неофіційний контент" (handle_unofficial_characters, handle_unofficial_banners, handle_unofficial_locations, handle_unofficial_curiosities): Викликають send_unofficial_media для відображення відповідного контенту.
- Обробник вибору мови (handle_language_selection): Обробляє текстові повідомлення "Українська" та "English", викликаючи set_language.
- Обробник кнопки "Назад" з меню "Цікавинки" (handle_back_to_main_menu_from_curiosities): Повертає користувача до головного меню.

14. Локалізація (файл localization.py):

Файл localization.py повинен містити переклади для всіх текстових елементів, пов'язаних з меню "Цікавинки":

- Назва розділу "Цікавинки" (curiosities_title).
- Назви кнопок головного меню "Цікавинки" (birthday_calendar_button, promocodes, unofficial, pictures, change_language, back_to_main_menu).
- Текст підменю "Неофіційний контент" (unofficial_menu_caption).

- Назви кнопок підменю "Неофіційний контент" (characters_button, future_banners_button, locations_button, curiosities_button, back_to_curiosities_text).
- Тексти підписок/відписок для календаря та промокодів (subscribe_birthday_calendar, unsubscribe_birthday_calendar, subscribed_calendar, unsubscribed_calendar, subscribe_promocodes, unsubscribe_promocodes, subscribed_success_promocodes, unsubscribed_success_promocodes).
- Повідомлення про відсутність фотографії (photo_not_found).
- Текст вибору мови (choose_language).
- Повідомлення про зміну мови (language_changed).
- Текст про календар днів народжень (birthday_calendar_info).
- Текст про промокоди (promocodes_info, subscribe_promocodes_offer).
- Тексти помилок (якщо є специфічні для цього розділу).
- Підписи до неофіційного контенту (FUTURE_CAPTION або інші).
- Текст повідомлення про відсутність зображень у папці неофіційного контенту.

15. Структура папок з неофіційним контентом:

У папці, шлях до якої вказано в UNOFFICIAL_FOLDER, повинні існувати підпапки з назвами, що відповідають категоріям неофіційного контенту (наприклад, characters, banners, locations, curiosities). У кожній з цих підпапок повинні знаходитися відповідні зображення.

16. Структура папки з випадковими зображеннями:

У папці, шлях до якої вказано в IMAGE_FOLDER, повинні знаходитися випадкові зображення. Назви файлів можуть містити інформацію про автора, розділену символом, визначеним у AUTHOR_DELIMITER, або мати стандартні префікси для позначення офіційного арту чи невідомого автора.

Принцип роботи:

1. Користувач натискає кнопку "Цікавинки" в головному меню.

2. Викликається show_curiosities_menu, відображається головне меню "Цікавинки".
3. Користувач обирає одну з опцій:
    - Календар днів народжень: Викликається show_birthday_calendar, відображається календар з кнопками підписки/відписки. Натискання кнопок обробляються в handle_callback_query.
    - Промокоди: Викликається show_promocodes, відображається інформація про промокоди з кнопками підписки/відписки (обробка в handle_callback_query).
    - Неофіційний контент: Викликається show_unofficial_keyboard, відображається підменю з різними категоріями неофіційного контенту. При виборі категорії (наприклад, "Персонажі"), викликається відповідна функція (unofficial_characters), яка через send_unofficial_media надсилає зображення з відповідної підпапки.
    - Зображення: Викликається send_random_image_with_author, надсилається випадкове зображення з папки IMAGE_FOLDER з інформацією про автора (якщо є).
    - Змінити мову: Викликається language_command, відображається клавіатура для вибору мови. Вибір мови обробляється в set_language.
    - Назад до головного меню: Користувач повертається до основного меню бота.

Ця структура забезпечує доступ до різноманітного додаткового контенту та функцій, роблячи бота більш цікавим та корисним для користувачів.

3.2.6 Обробка меню "Що сьогодні"

Розділ "Що сьогодні" в боті призначений для надання користувачам актуальної інформації про ігровий контент, який є активним у поточний день.

Наразі він включає інформацію про таланти персонажів та матеріали піднесення зброї, доступні для фарму, а також поточні ігрові банери.

1. Функція show_today_menu:

```
async def show_today_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['today_talents']],
        [localized_texts[user_language]['today_weapons']],
        [localized_texts[user_language]['banners']],
        [localized_texts[user_language]['back_to_main_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['today_info'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'today_menu'
```

Призначення: Ця асинхронна функція відображає головне меню розділу "Що сьогодні".

Логіка:

1. Отримує мову користувача.
2. Формує клавіатуру ReplyKeyboardMarkup з кнопками:
   - "Сьогоднішні таланти" (today_talents).
   - "Сьогоднішня зброя" (today_weapons).
   - "Банери" (banners).
   - "Назад до головного меню" (back_to_main_menu).
3. Надсилає користувачеві повідомлення з текстом "Інформація на сьогодні:" (локалізованим) та створеною клавіатурою.
4. Встановлює стан 'current_menu' у context.user_data на 'today_menu' для відстеження поточного розділу.

2. Функція show_weapon_talents (назва може бути неточною, ймовірно, малося на увазі "матеріали піднесення талантів"):

```
async def show_weapon_talents(update: Update, context: CallbackContext):
    today_weekday = datetime.datetime.now().weekday()
    photo_filename = f'{today_weekday}_talents.jpg'
```

```
        photo_path = os.path.join('days', photo_filename)

        if os.path.exists(photo_path):
            with open(photo_path, 'rb') as photo:
                caption                    =                    f"Сьогодні
({datetime.datetime.now().strftime('%A')}):"
                await
context.bot.send_photo(chat_id=update.effective_chat.id,          photo=photo,
caption=caption)
        else:
            await   update.message.reply_text(f"Фото   талантів   зброї   для
{datetime.datetime.now().strftime('%A')} не знайдено.")
```

Призначення: Ця асинхронна функція відображає зображення з графіком матеріалів піднесення талантів персонажів на поточний день тижня.

Логіка:

1. Визначає поточний день тижня (понеділок - 0, неділя - 6).
2. Формує ім'я файлу зображення, використовуючи номер дня тижня (наприклад, 0_talents.jpg для понеділка).
3. Створює повний шлях до файлу в папці days.
4. Перевіряє, чи існує файл за вказаним шляхом.
5. Якщо файл існує, відкриває його та надсилає як фотографію з підписом, що вказує поточний день тижня.
6. Якщо файл не знайдено, надсилає повідомлення про це користувачеві.

3. Функція send_media_group_with_caption:

```
    async def send_media_group_with_caption(chat_id: int, image_paths: list,
bot, caption=None):
        print(f"Шляхи перед відправкою: {image_paths}") # Додайте цей рядок
        media = []
        for i, path in enumerate(image_paths):
            if os.path.exists(path):
                try:
                    with open(path, 'rb') as f:
                        media.append(InputMediaPhoto(media=f.read(),
caption=caption if i == 0 else None))
```

```
                except Exception as e:
                    print(f"Помилка при відкритті файлу {path}: {e}")
            else:
                print(f"Файл не знайдено: {path}")

        if media:
            try:
                await bot.send_media_group(chat_id=chat_id, media=media)
            except Exception as e:
                print(f"Помилка при відправці медіагрупи: {e}")
        elif caption:
            await bot.send_message(chat_id=chat_id, text=caption)
        else:
            user_language = context.user_data.get('language', 'uk')
            await                          bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_banners_found'])
```

Призначення: Ця асинхронна функція надсилає групу зображень з можливістю додавання підпису до першого зображення.

Аргументи:

- chat_id: int: ID чату, куди потрібно надіслати медіагрупу.
- image_paths: list: Список шляхів до файлів зображень.
- bot: Об'єкт бота telegram.Bot.
- caption: str: Необов'язковий підпис для першого зображення.

Логіка:

1. Ітерується по списку шляхів до зображень.
2. Для кожного шляху перевіряє існування файлу.
3. Якщо файл існує, відкриває його та додає як InputMediaPhoto до списку media. Підпис додається лише до першого елемента.
4. Якщо список media не порожній, намагається надіслати групу зображень.
5. Обробляє можливі помилки при відкритті файлів або надсиланні медіагрупи.
6. Якщо media порожній, але є підпис, надсилає лише текстове повідомлення з підписом.

7. Якщо media порожній і немає підпису, надсилає локалізоване повідомлення про відсутність банерів (може бути недоречним тут, якщо функція використовується для інших цілей).

4. Функція show_current_banners:

```
async def show_current_banners(update: Update, context: CallbackContext):
    print("DEBUG (banners): Викликано show_current_banners")
    user_language = context.user_data.get('language', 'uk')
    print(f"DEBUG (banners): Мова користувача: {user_language}")
    character_image_paths = BANNER_IMAGES.get("character", [])
    weapon_image_paths = BANNER_IMAGES.get("weapon", [])
    print(f"DEBUG (banners): Шляхи банерів персонажів (до обробки): {character_image_paths}")
    print(f"DEBUG (banners): Шляхи банерів зброї (до обробки): {weapon_image_paths}")

    if isinstance(weapon_image_paths, str):
        weapon_image_paths = [weapon_image_paths]
    print(f"DEBUG (banners): Кількість банерів персонажів для відправки: {len(character_image_paths)}")
    print(f"DEBUG (banners): Шляхи банерів персонажів: {character_image_paths}")
    if character_image_paths:
        print(f"DEBUG (banners): Намагаюся відправити банери персонажів з підписом: {localized_texts[user_language]['character_banners_caption']}")
        await send_media_group_with_caption(update.effective_chat.id, character_image_paths, context.bot, localized_texts[user_language]['character_banners_caption'])
        print("DEBUG (banners): Банери персонажів відправлено (або спроба відправки завершена).")

    print(f"DEBUG (banners): Кількість банерів зброї для відправки: {len(weapon_image_paths)}")
    print(f"DEBUG (banners): Шляхи банерів зброї: {weapon_image_paths}")
    if weapon_image_paths:
        print(f"DEBUG (banners): Намагаюся відправити банери зброї з підписом: {localized_texts[user_language]['weapon_banners_caption']}")
        await send_media_group_with_caption(update.effective_chat.id, weapon_image_paths, context.bot, localized_texts[user_language]['weapon_banners_caption'])
```

```
            print("DEBUG (banners): Банери зброї відправлено (або спроба
відправки завершена).")
        elif not character_image_paths and not weapon_image_paths:
            print(f"DEBUG (banners): Зображень банерів не знайдено,
відправляю повідомлення:
{localized_texts[user_language]['no_banners_found']}")
            await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['no_banners_found'])
        else:
            print("DEBUG (banners): Умова відсутності банерів не спрацювала,
хоча один зі списків порожній.")
```

Призначення: Ця асинхронна функція відображає поточні ігрові банери персонажів та зброї.

Логіка:

1. Виводить налагоджувальну інформацію.
2. Отримує мову користувача.
3. Отримує списки шляхів до зображень банерів персонажів та зброї з глобальної змінної BANNER_IMAGES (яка має бути визначена в іншому місці вашого коду).
4. Перевіряє, чи є weapon_image_paths рядком, і якщо так, перетворює його на список.
5. Якщо є шляхи до банерів персонажів, викликає send_media_group_with_caption для їх надсилання з відповідним локалізованим підписом.
6. Аналогічно надсилає банери зброї, якщо є відповідні шляхи.
7. Якщо обидва списки шляхів порожні, надсилає локалізоване повідомлення про відсутність банерів.
8. Містить додаткову налагоджувальну вивід інформації.

5. Обробники текстових повідомлень (у головному файлі бота або окремому файлі обробників):

- handle_today_button(update: Update, context: CallbackContext): Обробляє натискання кнопки "Що сьогодні" в головному меню, викликаючи show_today_menu.
- handle_today_talents(update: Update, context: CallbackContext): Обробляє натискання кнопки "Сьогоднішні таланти", викликаючи show_weapon_talents
- handle_today_weapons(update: Update, context: CallbackContext): Обробляє натискання кнопки "Сьогоднішня зброя", викликаючи функцію, яка аналогічно show_weapon_talents відображає зображення з графіком матеріалів піднесення зброї на поточний день.
- handle_banners(update: Update, context: CallbackContext): Обробляє натискання кнопки "Банери", викликаючи show_current_banners.
- handle_back_to_main_menu_from_today(update: Update, context: CallbackContext): Обробляє натискання кнопки "Назад до головного меню" в меню "Що сьогодні", повертаючи користувача до основного меню.

6. Локалізація (файл localization.py):

Файл localization.py повинен містити переклади для всіх текстових елементів, пов'язаних з меню "Що сьогодні":

- Назва розділу "Інформація на сьогодні:" (today_info).
- Назви кнопок (today_talents, today_weapons, banners, back_to_main_menu).
- Підписи до банерів персонажів (character_banners_caption).
- Підписи до банерів зброї (weapon_banners_caption).
- Повідомлення про відсутність банерів (no_banners_found).
- Повідомлення про відсутність файлу з графіком талантів/зброї.

7. Структура папок та файлів:

- У вас повинна існувати папка days, в якій знаходяться зображення з графіками матеріалів піднесення талантів (наприклад, 0_talents.jpg, 1_talents.jpg і так далі для кожного дня тижня).
- Аналогічно, для матеріалів піднесення зброї, вам знадобиться створити відповідні зображення та функцію для їх відображення (show_weapon_ascension_materials) з обробником handle_today_weapons.

8. Глобальна змінна BANNER_IMAGES:

У вашому коді десь має бути визначена глобальна змінна BANNER_IMAGES, яка є словником, що містить списки шляхів до зображень поточних банерів персонажів та зброї. Наприклад:

```
BANNER_IMAGES = {
    "character": ["path/to/character_banner1.jpg", "path/to/character_banner2.png"],
    "weapon": ["path/to/weapon_banner.jpg"]
}
```

Принцип роботи:

1. Користувач натискає кнопку "Що сьогодні" в головному меню.
2. Викликається show_today_menu, відображається меню "Що сьогодні".
3. Користувач обирає одну з опцій:
    - Сьогоднішні таланти: Викликається show_weapon_talents, бот визначає поточний день тижня, знаходить відповідне зображення та надсилає його.
    - Сьогоднішня зброя: Викликається show_weapon_ascension_materials (потрібно створити), аналогічно до талантів надсилається зображення з графіком матеріалів піднесення зброї.
    - Банери: Викликається show_current_banners, бот отримує шляхи до зображень банерів з BANNER_IMAGES та надсилає їх групою з відповідними підписами.

- Назад до головного меню: Користувач повертається до основного меню.

Ця структура дозволяє користувачам швидко отримувати актуальну інформацію про щоденний ігровий контент та поточні банери. Переконайтеся, що ви правильно налаштували шляхи до файлів та визначили глобальну змінну BANNER_IMAGES. Також створіть функцію для відображення матеріалів піднесення зброї, аналогічну show_weapon_talents.

3.3 Обробка повідомлень користувача handle_message

Функція handle_message є ключовим обробником текстових повідомлень, які надходять до бота від користувачів. Її основна мета - інтерпретувати введений користувачем текст та відповідно реагувати, керуючи навігацією по меню бота та виконуючи певні дії.

Основні етапи роботи функції:

1. Отримання інформації про повідомлення та користувача:
    - user_message = update.message.text: Отримує текст повідомлення, надісланого користувачем.
    - user_language = context.user_data.get('language', 'uk'): Визначає мову користувача, збережену в контексті. За замовчуванням використовується українська.
    - localized_back = localized_texts[user_language]['back']: Отримує локалізований текст для кнопки "Назад".
    - if 'history' not in context.user_data: context.user_data['history'] = []: Ініціалізує порожній список історії навігації користувача, якщо він ще не існує в контексті.
    - user_text = update.message.text.lower(): Переводить текст повідомлення користувача в нижній регістр для полегшення порівняння.
    - current_menu = context.user_data.get('current_menu'): Отримує назву поточного меню, в якому перебуває користувач, з контексту.

Ця змінна використовується для визначення подальшої логіки обробки повідомлення.

- history = context.user_data.get('history', []): Отримує історію навігації користувача.
- selected_element = context.user_data.get('selected_element'): Отримує обраний елемент (наприклад, стихію персонажа) з контексту.
- selected_rarity_text = context.user_data.get('selected_rarity_text'): Отримує обрану рідкість (текстове представлення) з контексту.
- Виводиться налагоджувальна інформація про поточний стан для відстеження логіки виконання.

2. Обробка адміністративних команд завантаження та промокодів:
    - Перевіряються прапорці waiting_for_upload та waiting_for_promo_text у контексті користувача та ID чату на відповідність ADMIN_ID.
    - Якщо користувач є адміністратором і бот очікує завантаження тексту або тексту промокоду, викликаються відповідні обробники (handle_upload_text або handle_promo_text) та функція завершує виконання.
3. Визначення списку елементів:
    - element_list_lower = [...]: Створюється список назв стихій персонажів у нижньому регістрі на поточній мові користувача. Цей список може використовуватися для перевірки введеного користувачем тексту.
4. Основна логіка обробки повідомлень на основі поточного меню (current_menu):
    - Головне меню:
        - Якщо user_text відповідає назвам основних розділів ("Персонажі", "Зброя", "Артефакти", "Сьогодні",

"Цікавинки", "Битва") на поточній мові, викликаються відповідні функції для відображення цих меню (show_character_menu, show_weapon_types_menu, show_artifacts_menu, show_today_menu, show_curiosities_menu, show_combat_menu).

- При виборі "Артефакти" встановлюється прапорець awaiting_artifact_category в True.

- Меню "Артефакти" (current_menu == 'artifacts_menu'):
  - Перевіряється, чи відповідає user_text назвам кнопок категорій артефактів (наприклад, "Атака", "Бонус таланту" тощо). Якщо так, викликається send_artifact_category_media для надсилання відповідних зображень.
  - Якщо user_text відповідає кнопці "Головне меню", викликається функція start для повернення в головне меню та очищається current_menu.
- Обробка загальних команд:
  - Якщо user_text відповідає командам зміни мови, викликаються language_command або set_language.
- Меню "Що сьогодні" (current_menu == 'today_menu'):
  - Перевіряється, чи відповідає user_text назвам кнопок "Сьогоднішні таланти", "Сьогоднішня зброя", "Банери".
  - Для "Сьогоднішні таланти" та "Сьогоднішня зброя" визначається ім'я файлу зображення на основі поточного дня тижня, перевіряється його існування та надсилається користувачеві з відповідним локалізованим підписом.
  - Для "Банери" викликається функція show_current_banners.
  - При виборі "Назад до головного меню" викликається start та очищається current_menu.
- Меню "Цікавинки" (current_menu == 'curiosities_menu'):

- Перевіряється, чи відповідає user_text назвам кнопок підменю ("Календар днів народжень", "Промокоди", "Неофіційний контент", "Зображення", "Змінити мову", "Назад до головного меню"). Викликаються відповідні функції.
- Для "Неофіційний контент" викликається show_unofficial_keyboard.

- Підменю "Неофіційний контент" (current_menu == 'unofficial_menu'):
  - Перевіряється вибір категорій неофіційного контенту ("Персонажі", "Майбутні банери", "Локації", "Цікавинки"). Викликаються відповідні функції надсилання контенту (unofficial_characters, unofficial_banners, unofficial_locations, unofficial_curiosities).
  - При виборі "Назад" (back_to_curiosities_text) користувач повертається до меню "Цікавинки".
- Меню "Битва" (current_menu == 'combat_menu'):
  - Перевіряється вибір підрозділів ("Вороги", "Безодня", "Театр", "Реакції"). Для кожного підрозділу викликається send_battle_photos_keyboard для надсилання відповідних зображень (для "Вороги" викликається show_enemies_menu).
  - При виборі "Назад до головного меню" відбувається повернення до головного меню.
- Меню "Вороги" (current_menu == 'enemies_menu'):
  - Перевіряється вибір типів ворогів ("Боси", "Звичайні вороги", "Елітні вороги"). Викликається show_enemies_list для відображення списку ворогів обраного типу.

- При виборі "Назад до меню битви" відбувається повернення до меню "Битва".

- Списки ворогів (current_menu починається з 'enemies_list_'):
  - Визначається тип ворогів зі значення current_menu.
  - Якщо user_text відповідає локалізованій кнопці "Назад до меню ворогів", відбувається повернення до меню "Вороги".
  - Інакше вважається, що користувач вибрав конкретного ворога, і викликається send_enemy_photos для надсилання його зображень.
- Меню вибору стихії персонажа (current_menu == 'character_elements'):
  - Обраний користувачем текст зберігається як selected_element у контексті.
  - Залежно від обраної стихії, викликається відповідна функція для відображення меню вибору рідкості персонажів цієї стихії (show_anemo_rarity_menu, show_pyro_rarity_menu тощо).
  - При виборі локалізованої кнопки "Назад" відбувається повернення до головного меню.
- Меню вибору рідкості персонажа (наприклад, current_menu == 'anemo_rarity'):
  - Перевіряється, чи відповідає user_text варіантам вибору рідкості (наприклад, "5★ Анемо", "4★ Анемо"). Якщо так, викликаються функції для відображення списку персонажів відповідної рідкості (show_anemo_5_star_characters, show_anemo_4_star_characters).
  - При виборі локалізованої кнопки "Назад" відбувається повернення до меню вибору стихії.
- Списки персонажів (current_menu закінчується на '_list'):

- Визначається стихія та рідкість зі значення current_menu.
- Якщо user_text закінчується відповідним емодзі стихії, відбувається повернення до меню вибору рідкості цієї стихії.
- Інакше вважається, що користувач вибрав конкретного персонажа, і викликається handle_character_selection_from_list для обробки цього вибору.

- Меню вибору типу зброї (current_menu == 'weapon_types'):
  - Обраний користувачем тип зброї зберігається як selected_weapon_type у контексті.
  - Залежно від обраного типу зброї, викликається відповідна функція для відображення меню вибору рідкості зброї цього типу (show_sword_rarity_menu, show_claymore_rarity_menu тощо).
  - При виборі локалізованої кнопки "Назад" відбувається повернення до головного меню.
- Меню вибору рідкості зброї (наприклад, current_menu == 'sword_rarity'):
  - Перевіряється вибір рідкості зброї (5★ або 4★). Викликаються функції для відображення списку зброї відповідної рідкості.
  - При виборі локалізованої кнопки "Назад" відбувається повернення до меню вибору типу зброї.
- Списки зброї (current_menu закінчується на '_list'):
  - Визначається тип зброї та рідкість.
  - Якщо user_text закінчується відповідним емодзі зброї, відбувається повернення до меню вибору рідкості цього типу зброї.

- Інакше викликається handle_weapon_selection_from_list для обробки вибору конкретної зброї.
- У випадку, якщо current_menu не визначено або не відповідає жодному з оброблених станів, відбувається повернення до головного меню (start(update, context)).

Таким чином, функція handle_message є центральним механізмом обробки текстових команд користувача, забезпечуючи навігацію по різних розділах бота та виконання відповідних дій на основі поточного контексту та введеного тексту.

ВИСНОВОК ДО РОЗДІЛУ 3

Розділ 3 представляє собою детальну розробку та опис ключових функціональних компонентів Telegram-бота "Помічничок мандрівника Genshin Impact", зосереджених на наданні користувачам різноманітної інформації та зручній навігації. Проаналізовані функції охоплюють основні аспекти взаємодії користувача з ботом, включаючи:

- Керування меню: Описано логіку відображення та переходу між різними розділами бота, такими як головне меню, меню "Що сьогодні", "Цікавинки", "Артефакти", "Персонажі", "Зброя" та "Битва". Для навігації використовуються клавіатурні кнопки (ReplyKeyboardMarkup) та обробники текстових повідомлень (handle_message).
- Відображення інформації: Розглянуто механізми надсилання користувачам текстових повідомлень, фотографій та груп медіафайлів. Функції використовують локалізовані текстові ресурси для забезпечення багатомовності бота.
- Обробка дій користувача: Проаналізовано, як бот реагує на натискання кнопок та введення тексту, використовуючи контекст (CallbackContext) для збереження стану користувача та історії навігації.
- Спеціалізовані функції: Детально описано логіку роботи окремих функцій, таких як відображення календаря днів народжень, інформації про промокоди, неофіційного контенту, випадкових зображень, а також інформації про щоденні таланти, матеріали піднесення зброї та поточні банери.
- Адміністративні функції: Згадано обробку команд завантаження контенту та розсилки промокодів для адміністраторів бота.
- Локалізація: Підкреслено важливість використання файлу localization.py для підтримки різних мов інтерфейсу бота.

- Структура файлів та папок: Описано очікувану організацію файлів зображень для різних розділів бота (дні тижня, неофіційний контент, випадкові зображення, артефакти, вороги).
- Використання бібліотек: Ідентифіковано та описано призначення ключових бібліотек Python, таких як telegram.ext для роботи з Telegram Bot API, os для взаємодії з файловою системою, datetime для роботи з датою та часом, random для генерації випадкових значень та collections для використання спеціалізованих контейнерів.

Загалом, розділ 3 демонструє комплексну архітектуру обробки повідомлень та керування контентом бота. Функція handle_message відіграє центральну роль у маршрутизації запитів користувачів до відповідних обробників, забезпечуючи інтерактивність та інформативність бота. Розділ також підкреслює важливість ретельної організації файлів, використання локалізації та застосування відповідних бібліотек Python для ефективної розробки Telegram-ботів. Подальший розвиток бота може включати розширення функціональності, покращення обробки помилок та оптимізацію коду на основі описаних принципів.

РОЗДІЛ 4.
ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ.

4.1 Методологія тестування

Для забезпечення всебічної перевірки системи застосовуватиметься ручне тестування (Manual Testing). Цей підхід передбачає безпосередню, послідовну та систематичну взаємодію тестувальника з ботом через стандартний інтерфейс Telegram. Процес тестування включає:

- Покрокове виконання сценаріїв: Тестувальник буде слідувати заздалегідь визначеним сценаріям, що охоплюють усі можливі шляхи взаємодії користувача з ботом.
- Спостереження за поведінкою системи: Уважне відстеження відповідей бота, відображення меню, текстів, зображень та інших елементів інтерфейсу.
- Порівняння з очікуваними результатами: Кожна дія тестувальника та відповідь бота порівнюватимуться з очікуваною поведінкою, визначеною в вимогах до системи.
- Фіксація відхилень: Будь-які невідповідності або помилки (баги) будуть детально задокументовані.

Тестування проводитиметься ітераційно, після реалізації кожного значного блоку функціональності або внесення суттєвих змін. Це дозволить виявляти помилки на ранніх етапах та ефективно їх виправляти, покращуючи якість продукту.

4.2 Функціональне тестування за розділами меню

Функціональне тестування зосереджуватиметься на ручній перевірці коректності роботи кожного розділу бота та відповідності його поведінки очікуваним результатам згідно з функціональними вимогами.

4.2.1 Тестування головного меню та загальних команд

- Команда /start:
    - Дія: Користувач надсилає команду /start.

- Очікуваний результат: Бот має надіслати вітальне повідомлення (відповідною мовою, якщо вже обрана) та відобразити головне меню з усіма основними кнопками (наприклад, "Персонажі", "Зброя", "Що сьогодні", "Цікавинки", "Битва", "Допомога") (рис. 4.1).

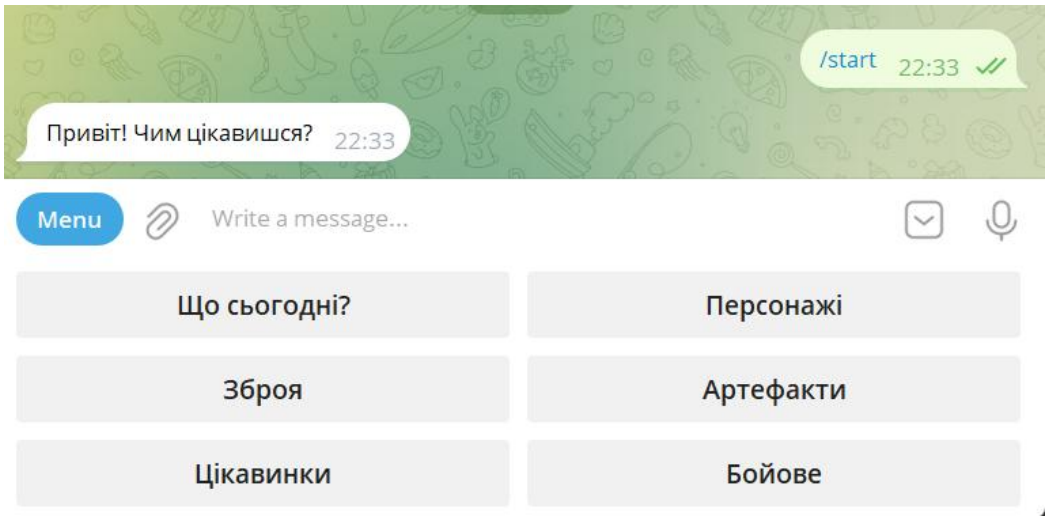


Рисунок 4.1 – Зображення головного меню бота

  - Сценарії: Перевірка після першого запуску, після перезапуску бота, після тривалої бездіяльності.
- Команда /help:
  - Дія: Користувач надсилає команду /help.
  - Очікуваний результат: Бот має надіслати інформативне повідомлення, що пояснює основні функції бота та як ними користуватися, включаючи перелік доступних команд та загальних правил взаємодії (рис. 4.2).

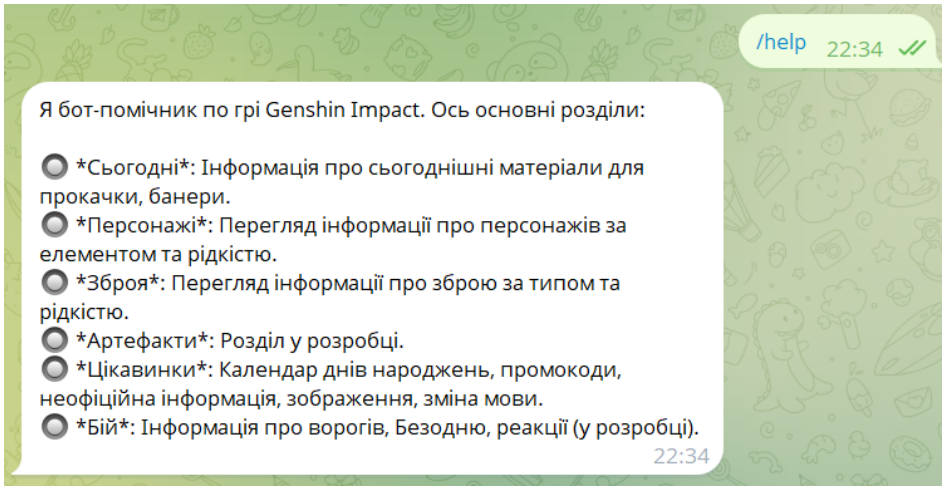


Рисунок 4.2 – Зображення довідкового повідомлення бота

- Команда /language:
  - Дія: Користувач надсилає команду /language.
  - Очікуваний результат: Бот має відобразити меню вибору мови з доступними варіантами (наприклад, "Українська", "English"). Після вибору мови, бот повинен підтвердити зміну та повернутися до головного меню з оновленим інтерфейсом (рис. 4.3).



Рисунок 4.3 – Зображення меню вибору мови

- Загальна навігація:
  - Дія: Послідовний перехід між усіма основними пунктами головного меню.

- Очікуваний результат: Кожен пункт меню має коректно відкриватися, відображаючи відповідне підменю або інформацію. Відсутність затримок або збоїв при переході.

4.2.2 Тестування розділу "Персонажі"

- Навігація:
  - Дія: З головного меню обрати кнопку "Персонажі".
  - Очікуваний результат: Бот має відобразити меню з кнопками, що представляють різні елементи (Піро, Гідро, Анемо тощо) (рис. 4.4).

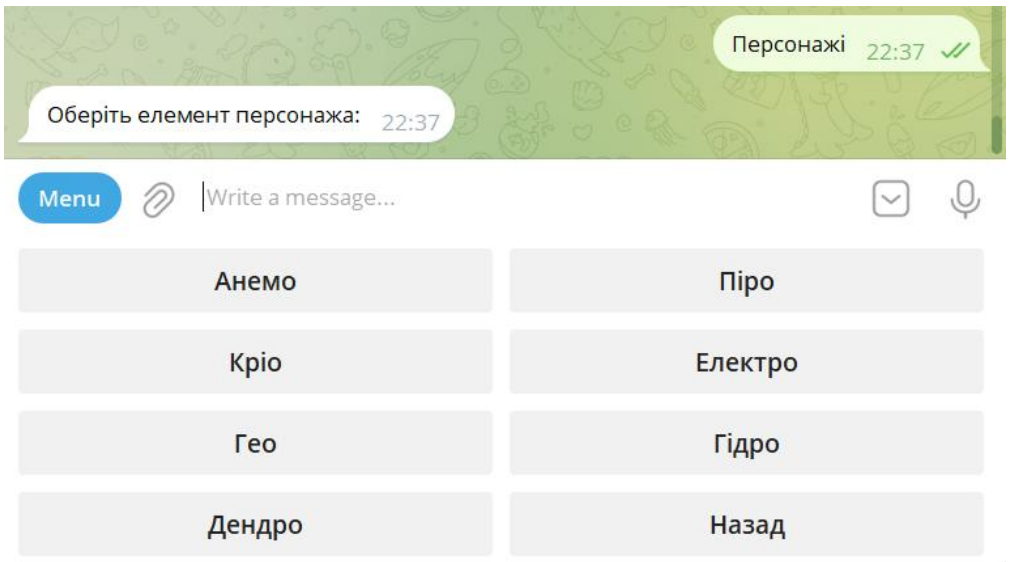


Рисунок 4.4 – Зображення меню вибору елементів персонажів

  - Дія: Обрати елемент (наприклад, "Піро").
  - Очікуваний результат: Бот має відобразити меню з кнопками, що представляють рідкість персонажів (5★, 4★, тощо) (рис. 4.5).

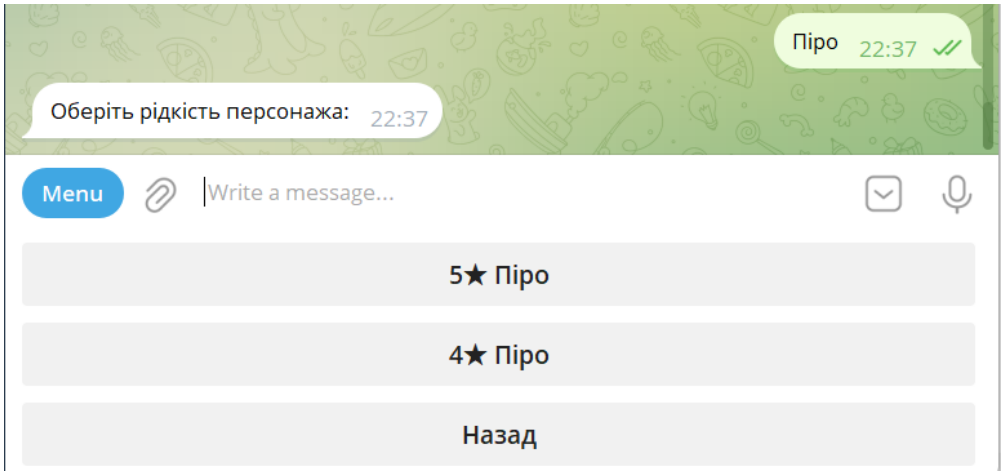


Рисунок 4.5 – Зображення меню вибору рідкості персонажів

- Дія: Обрати рідкість (наприклад, "5★").
- Очікуваний результат: Бот має відобразити список всіх персонажів обраного елемента та рідкості (рис. 4.6). Список повинен бути повним і містити лише відповідних персонажів.

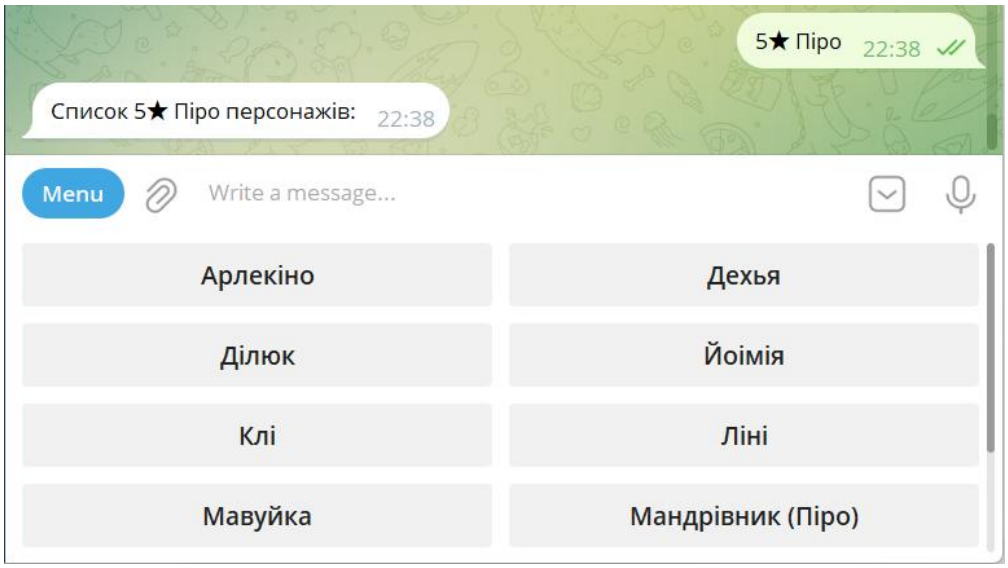


Рисунок 4.6 – Зображення списку персонажів (за елементом та рідкістю)

- Відображення інформації про персонажа:
  - Дія: Вибрати конкретного персонажа зі списку.
  - Очікуваний результат: Бот має надіслати повідомлення, що містить:

Коректну назву персонажа.

Його рідкість та елемент.

Тип зброї.

Короткий опис.

Якісне зображення персонажа (рис. 4.7).

Кнопку "Назад".

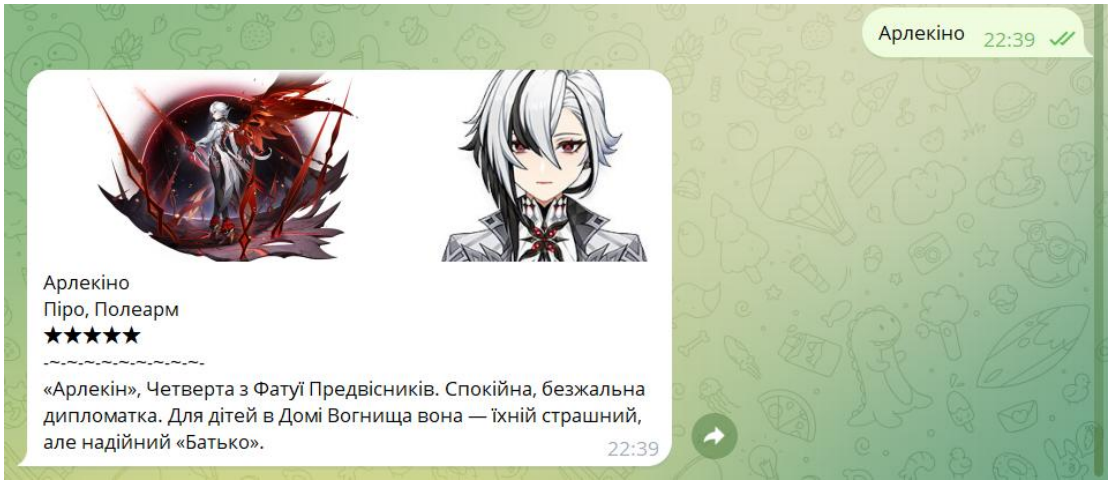


Рисунок 4.7 – Зображення інформації про персонажа

- Кнопки навігації "Назад":
  - Дія: Натискання кнопки "Назад" на етапі "Інформація про персонажа", "Список персонажів", "Меню рідкості", "Меню елементів".
  - Очікуваний результат: Кожен раз бот повинен коректно повертати користувача до попереднього рівня меню, зберігаючи контекст (наприклад, з меню рідкості повернутись до меню елементів).

4.2.3 Тестування розділу "Зброя"

- Навігація:
  - Дія: З головного меню обрати кнопку "Зброя" (рис. 4.1), потім тип зброї (наприклад, "Меч") (рис. 4.8), потім рідкість (наприклад, "4★") (рис. 4.9).

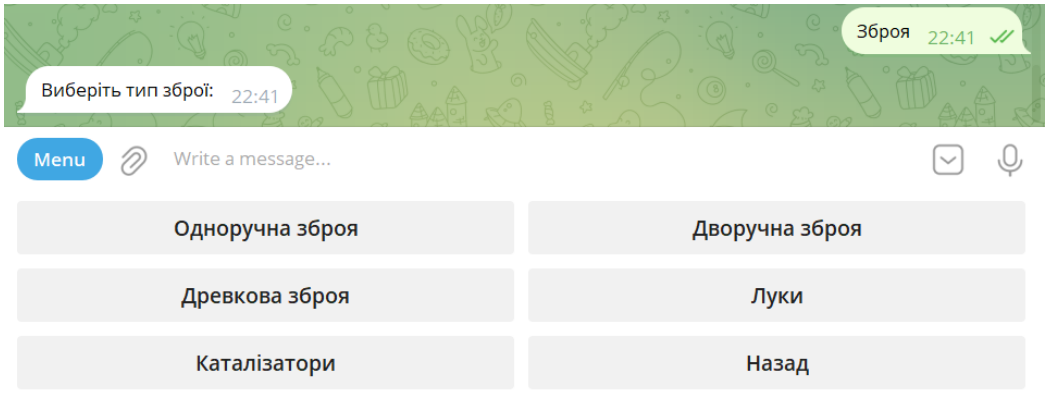


Рисунок 4.8 – Зображення меню вибору типу зброї



Рисунок 4.9 – Зображення меню вибору рідкості зброї

- Очікуваний результат: Бот має відобразити список зброї відповідного типу та рідкості (рис. 4.10).

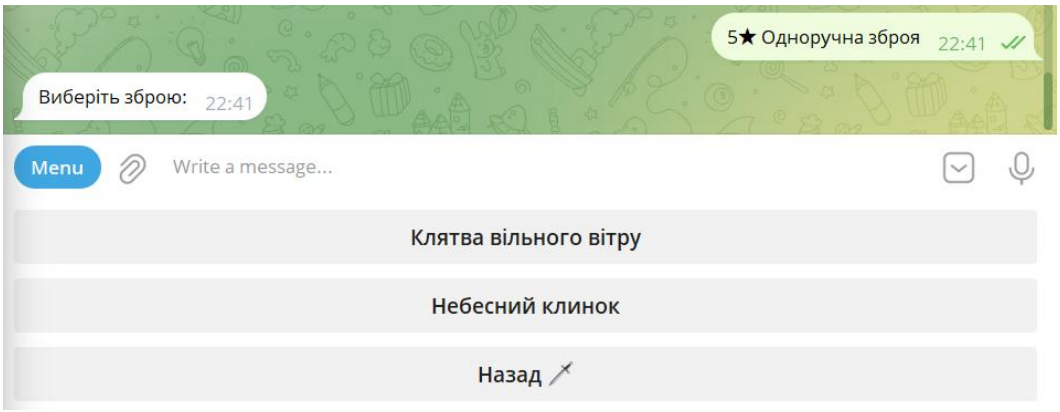


Рисунок 4.10 – Зображення списку зброї (за типом та рідкістю)

- Відображення інформації про зброю:
  - Дія: Вибрати конкретну зброю зі списку.
  - Очікуваний результат: Бот має надіслати повідомлення, що містить:

- Коректну назву зброї.
- Тип зброї та рідкість.
- Базові характеристики (базова атака, додаткова характеристика).
- Опис пасивної здібності.
- Якісне зображення зброї (рис. 4.11).
- Кнопку "Назад".



Рисунок 4.11 – Зображення інформації про зброю

- Кнопки навігації "Назад":
  - Дія: Натискання кнопки "Назад" на всіх етапах фільтрації та перегляду інформації про зброю.
  - Очікуваний результат: Коректне повернення до попереднього меню.

4.2.4 Тестування розділу "Артефакти"

- Навігація:
  - Дія: З головного меню обрати кнопку "Артефакти" (рис. 4.1), потім обрати категорію артефактів (рис. 4.12).

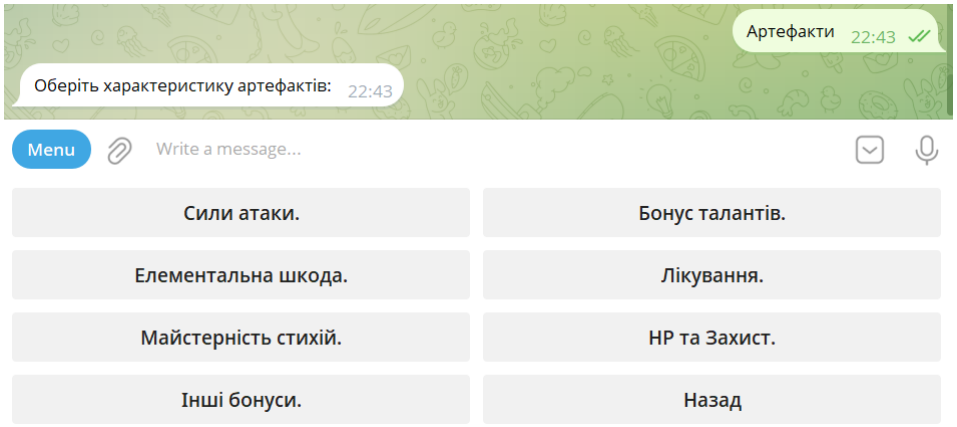


Рисунок 4.12 – Зображення меню категорій артефактів

- Очікуваний результат: Бот має відобразити список наборів артефактів у обраній категорії (рис. 4.13).

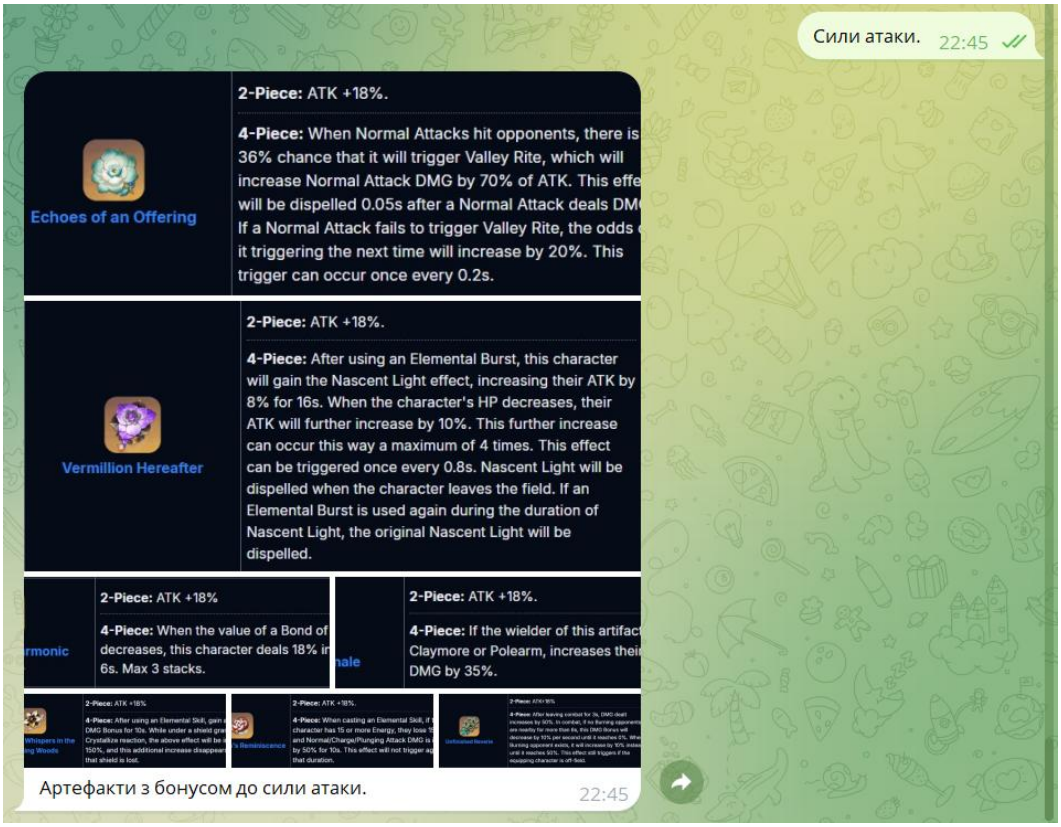


Рисунок 4.13 – Зображення списку артефактів

- Кнопки навігації "Назад":
  - Дія: Натискання кнопки "Назад" на всіх етапах.
  - Очікуваний результат: Коректне повернення до попереднього меню.

4.2.5 Тестування розділу "Що сьогодні"

- Навігація:
  - Дія: З головного меню обрати кнопку "Що сьогодні" (рис. 4.1).
  - Очікуваний результат: Бот має відобразити підменю з опціями "Сьогоднішні таланти", "Сьогоднішня зброя", "Банери" (рис. 4.14).

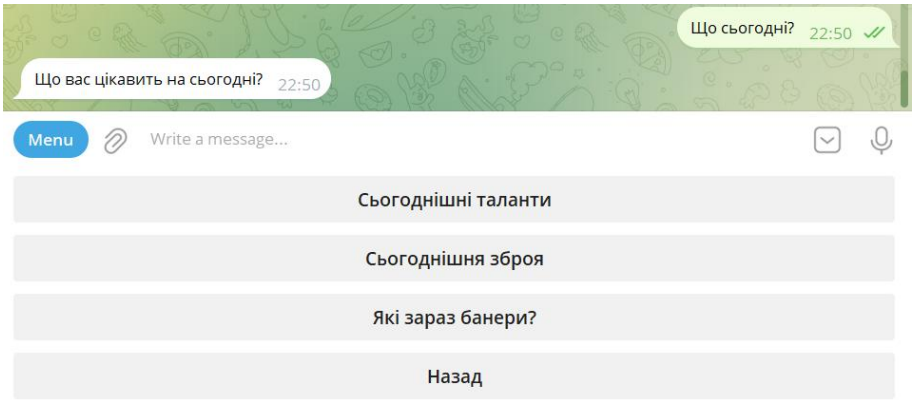


Рисунок 4.14 – Зображення меню "Що сьогодні"

- "Сьогоднішні таланти":
  - Дія: Обрати "Сьогоднішні таланти".
  - Очікуваний результат: Бот має відобразити інформацію (текст або зображення) про матеріали талантів, доступні для фарму в поточний день тижня, згідно з ігровим календарем (рис. 4.15).

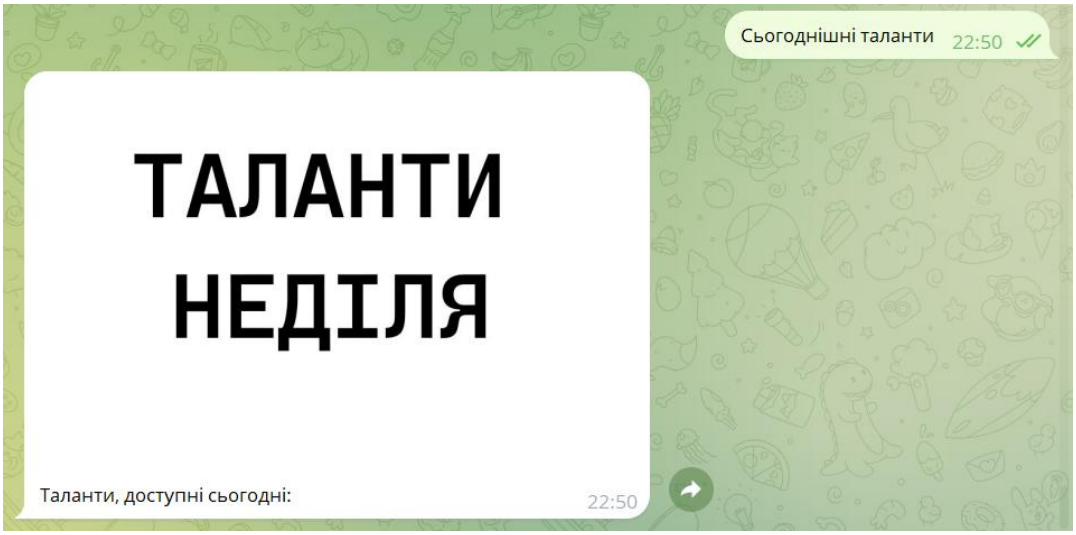


Рисунок 4.15 – Зображення інформації про сьогоднішні таланти

- "Сьогоднішня зброя":
  - Дія: Обрати "Сьогоднішня зброя".
  - Очікуваний результат: Бот має відобразити інформацію (текст або зображення) про матеріали для прокачки зброї, доступні в поточний день тижня (рис. 4.16).

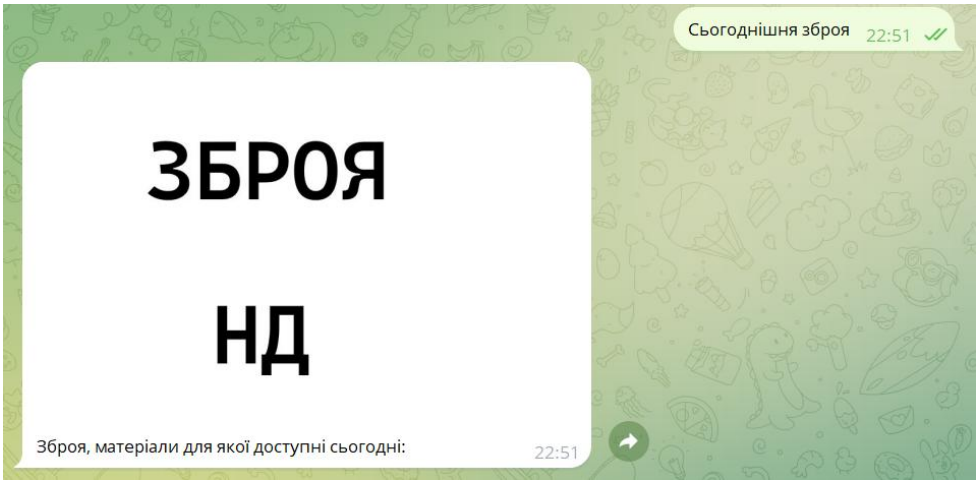


Рисунок 4.16 – Зображення інформації про сьогоднішню зброю

- "Банери":
  - Дія: Обрати "Банери".
  - Очікуваний результат: Бот має відобразити інформацію про поточні та/або майбутні банери персонажів та зброї, включаючи зображення та дати проведення (рис. 4.17).

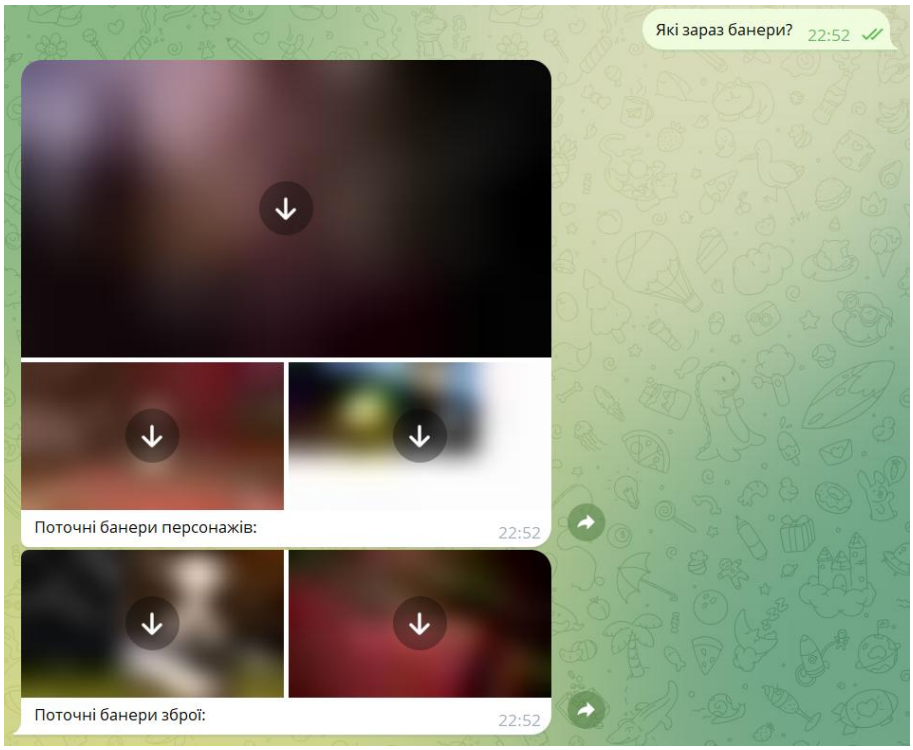


Рисунок 4.17 – Зображення актуальних банерів

- Кнопки навігації "Назад":
  - Дія: Натискання кнопки "Назад" у всіх підпунктах розділу.

- Очікуваний результат: Коректне повернення до меню "Що сьогодні".

4.2.6 Тестування розділу "Цікавинки"

- Навігація:
  - Дія: З головного меню обрати кнопку "Цікавинки" (рис. 4.1).
  - Очікуваний результат: Бот має відобразити підменю з опціями "Календар", "Промокоди", "Неофіційне", "Зображення", "Мова" (рис. 4.18).

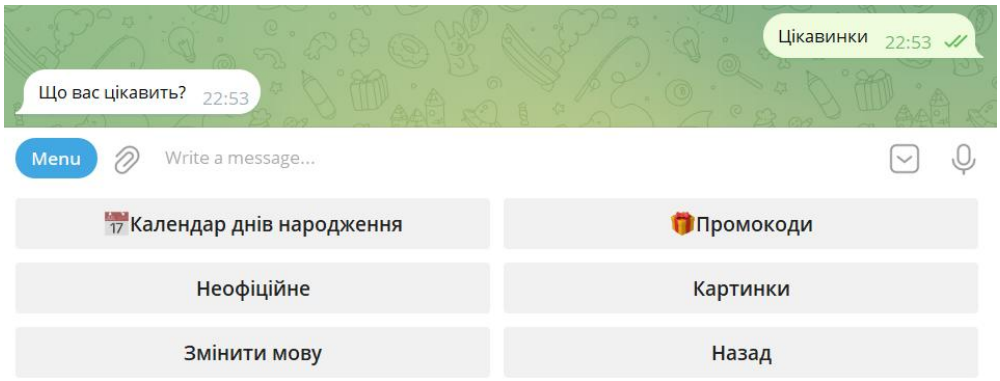
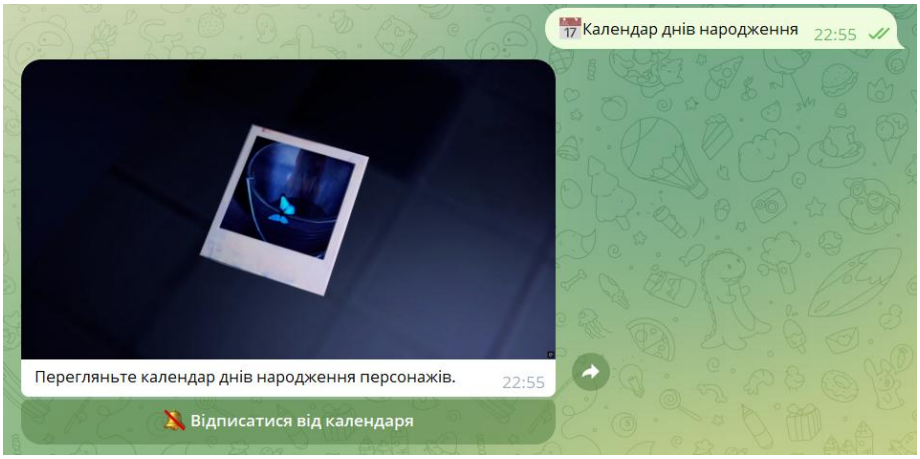


Рисунок 4.18 – Зображення меню "Цікавинки"

- "Календар":
  - Дія: Обрати "Календар".
  - Очікуваний результат: Бот має надіслати інформацію про дні народження персонажів (рис. 4.19).



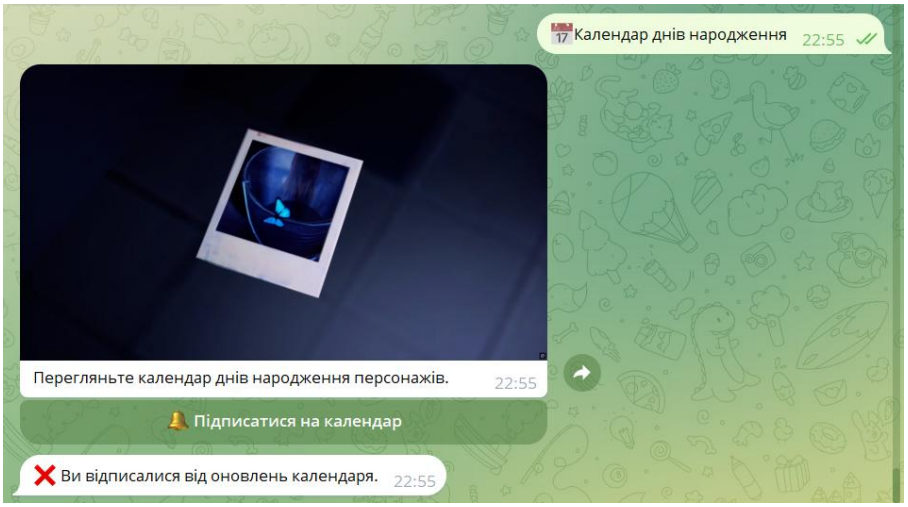


Рисунок 4.19 – Зображення календаря днів народжень

- "Промокоди":
  - Дія: Обрати "Промокоди".
  - Очікуваний результат: Бот має відобразити актуальний список доступних промокодів (рис. 4.20).

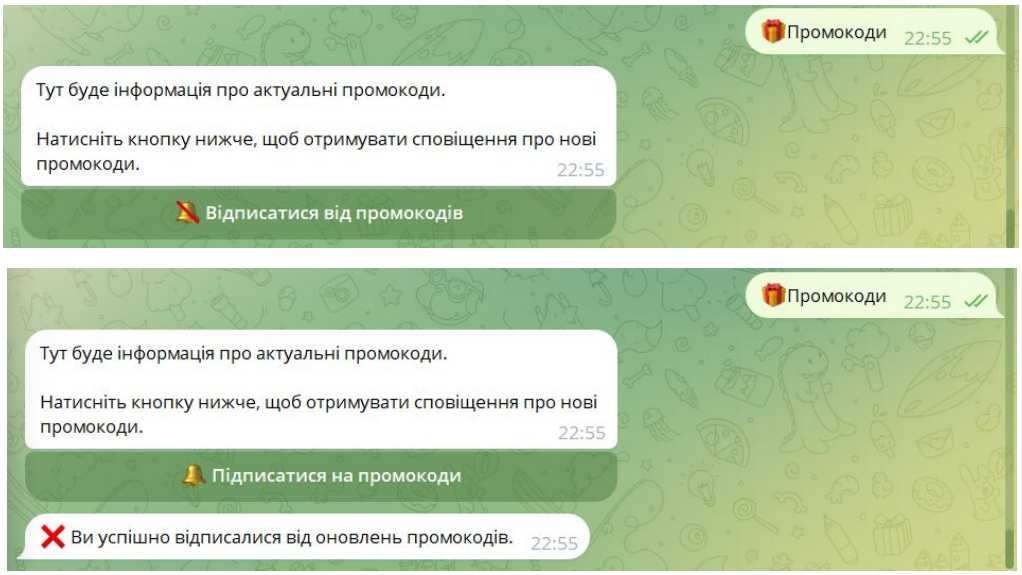


Рисунок 4.20 – Зображення списку промокодів

- "Неофіційне":
  - Дія: Обрати "Неофіційне", потім обрати категорію (рис. 4.21).



Рисунок 4.21 – Зображення меню "Неофіційне"

- Очікуваний результат: Бот має відобразити відповідний неофіційний контент (рис. 4.22).

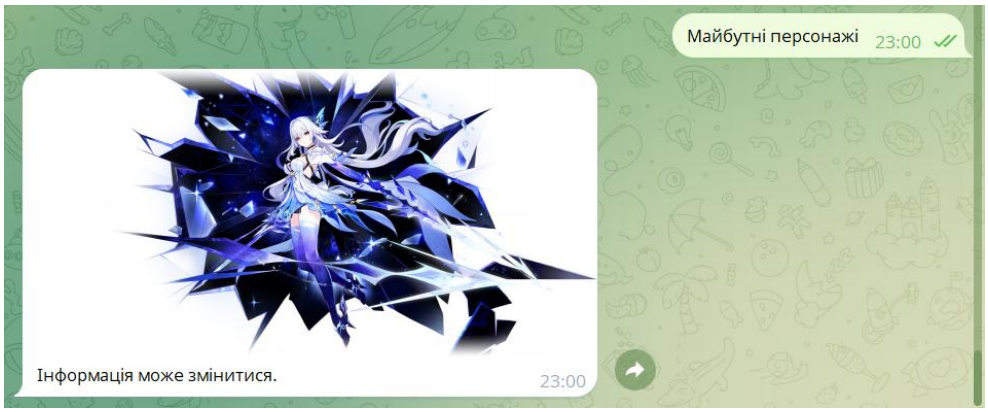


Рисунок 4.22 – Зображення неофіційного контенту

- "Зображення":
  - Дія: Обрати "Зображення".
  - Очікуваний результат: Бот має надіслати випадкове зображення, пов'язане з Genshin Impact (рис. 4.23).

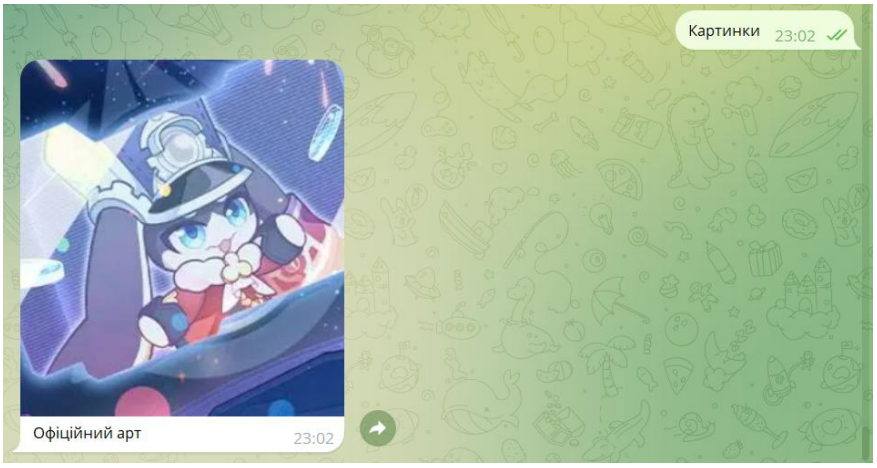


Рисунок 4.23 – Зображення випадкового зображення

- "Мова":
  - Дія: Обрати "Мова".
  - Очікуваний результат: Бот має перейти до меню вибору мови, аналогічно команді /language (рис. 4.3).
- Кнопки навігації "Назад":
  - Дія: Натискання кнопки "Назад" у всіх підпунктах розділу.
  - Очікуваний результат: Коректне повернення до меню "Цікавинки".

4.2.7 Тестування розділу "Битва"

- Навігація:
  - Дія: З головного меню обрати кнопку "Битва" (рис. 4.1), потім обрати категорію ворогів (наприклад, "Елітні вороги") (рис. 4.24).

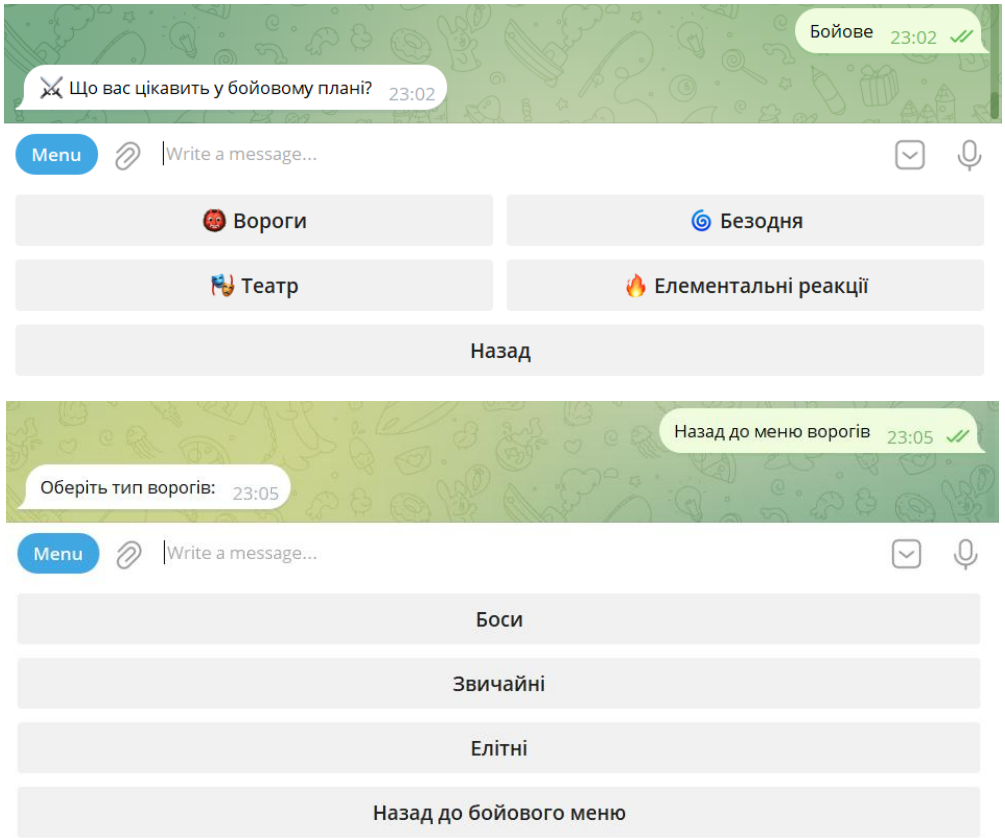


Рисунок 4.24 – Зображення меню категорій ворогів

  - Очікуваний результат: Бот має відобразити список ворогів обраної категорії (рис. 4.25).

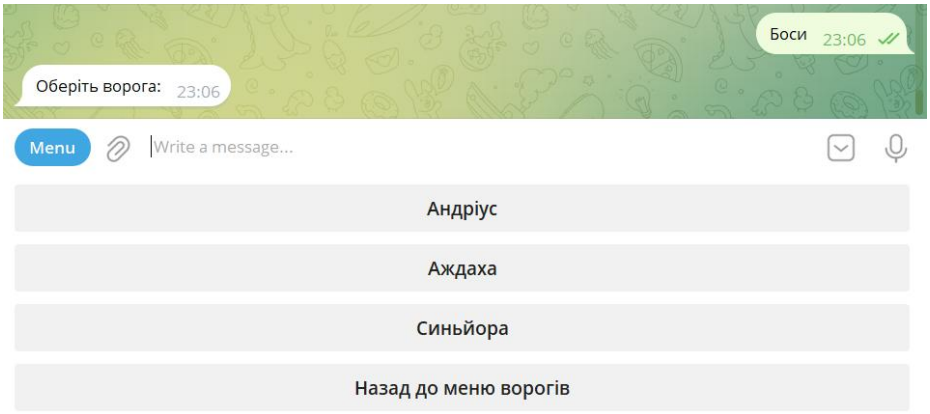


Рисунок 4.25 – Зображення списку ворогів

- Відображення інформації про ворога:
  - Дія: Вибрати конкретного ворога зі списку.
  - Очікуваний результат: Бот має надіслати повідомлення, що містить:
    - Коректну назву ворога.
    - Його тип.
    - Приблизне місцезнаходження.
    - Короткий опис.
    - Якісне зображення ворога (рис. 4.26).
    - Кнопку "Назад".

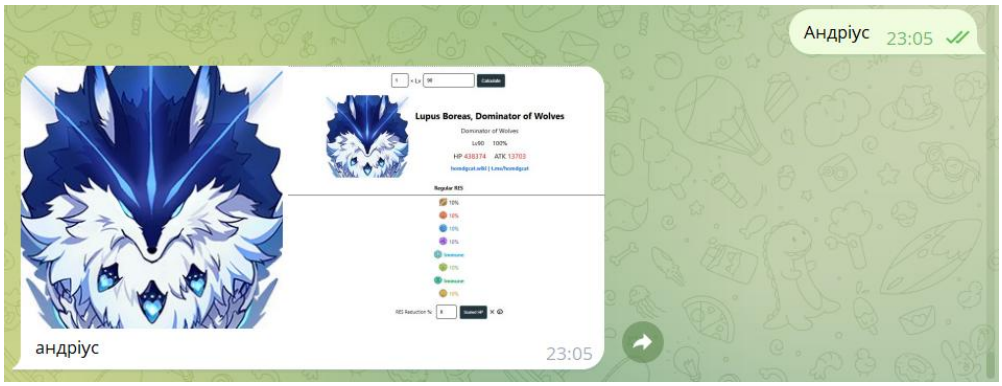


Рисунок 4.26 – Зображення інформації про ворога

- Кнопки навігації "Назад":
  - Дія: Натискання кнопки "Назад" на всіх етапах.
  - Очікуваний результат: Коректне повернення до попереднього меню.

4.3 Тестування локалізації

Тестування локалізації є критично важливим для багатомовного бота. Це буде здійснено шляхом ретельної ручної перевірки всіх текстів.

- Перемикання мови:
    - Дія: Використання команди /language або пункту меню "Мова" (рис. 4.3) для зміни мови бота (наприклад, з української на англійську та навпаки).
    - Очікуваний результат: Усі елементи інтерфейсу (назви кнопок, меню, вітальні повідомлення, довідкова інформація) та контент (описи персонажів, зброї, артефактів) повинні миттєво переключитися на обрану мову.
- Відповідність текстів:
    - Дія: Перевірка кожного пункту меню, інформаційного блоку, повідомлення про помилку в кожній підтримуваній мові.
    - Очікуваний результат: Тексти мають точно відповідати перекладам, що зберігаються у файлах локалізації. Буде звертатися увага на граматичні помилки, орфографічні помилки та невідповідності значень.
- Кодування символів:
    - Дія: Візуальна перевірка відображення всіх символів у різних мовах на різних пристроях (наприклад, Android, iOS, Desktop Telegram-клієнт).
    - Очікуваний результат: Усі спеціальні символи та символи не-латинських алфавітів (наприклад, українські літери) повинні відображатися коректно, без "кракозябр" або інших проблем з кодуванням.

4.4 Тестування зручності використання (Usability Testing)

Цей аспект тестування спрямований на суб'єктивну оцінку загального досвіду взаємодії користувача з ботом.

- Інтуїтивність навігації:
    - Дія: Спроба користуватися ботом без попередніх інструкцій, імітуючи дії нового користувача.
    - Очікуваний результат: Користувач повинен легко розуміти, як переміщатися між меню, як знайти потрібну інформацію, і що означають кнопки. Шлях до бажаної інформації має бути коротким і логічним.
- Зрозумілість повідомлень:
    - Дія: Оцінка всіх текстових повідомлень, які надсилає бот.
    - Очікуваний результат: Повідомлення мають бути чіткими, лаконічними, зрозумілими для цільової аудиторії, без жаргону, що не пояснюється. Повідомлення про помилки мають давати зрозуміти, що саме пішло не так і як це виправити.
- Швидкість відповіді:
    - Дія: Виконання серії швидких запитів, переходів по меню.
    - Очікуваний результат: Час реакції бота на дії користувача має бути мінімальним. Бот не повинен "зависати" або відповідати із значними затримками.
- Естетика та читабельність:
    - Дія: Візуальна оцінка оформлення повідомлень.
    - Очікуваний результат: Текст має бути добре відформатований (з використанням Markdown, якщо це передбачено), зображення мають бути якісними та правильно відображатися, кнопки мають бути чіткими та зручними для натискання.

4.5 Тестування надійності та обробки помилок

Цей розділ стосується ручної перевірки стійкості бота до непередбачених ситуацій та його здатності коректно реагувати на помилки.

- Обробка невірних даних/команд:

- Дія: Введення неіснуючих команд (наприклад, /abcdef), відправка випадкового тексту замість вибору кнопки, спроба натиснути "Назад" з головного меню, введення невірних параметрів у можливі команди (якщо такі будуть).
- Очікуваний результат: Бот не повинен "падати" (завершувати роботу). Замість цього, він повинен надіслати зрозуміле повідомлення про помилку (наприклад, "Вибачте, я не розумію цю команду" або "Будь ласка, оберіть зі списку"), залишаючись функціональним.

- Стійкість до незапланованих дій:
  - Дія: Спроба швидкого та хаотичного натискання кнопок, повторне надсилання одного і того ж запиту кілька разів поспіль.
  - Очікуваний результат: Бот має коректно обробляти такі дії, не викликаючи збоїв у логіці чи повторних відповідей.
- Довготривала робота:
  - Дія: Залишення бота працювати протягом тривалого періоду (кілька годин або днів) без перезапусків, з періодичними тестовими запитами.
  - Очікуваний результат: Бот повинен залишатися доступним і чуйним, не демонструючи ознак витоку пам'яті або інших проблем, які можуть призвести до його непрацездатності з часом.

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі ми приділили особливу увагу всебічному тестуванню системи, що є критично важливим етапом у життєвому циклі розробки програмного забезпечення. Метою наших зусиль було не просто виявлення помилок, а й підтвердження того, що розроблена система повністю відповідає визначеним функціональним та нефункціональним вимогам, а також демонструє високий рівень надійності та стабільності в роботі.

Процес тестування розпочався з модульного тестування, де кожен окремий компонент програмного забезпечення перевірявся в ізольованому середовищі. Цей етап дозволив нам оперативно виявити та виправити первинні помилки, гарантуючи коректну роботу базових елементів системи. Наступним кроком було інтеграційне тестування, під час якого ми зосередились на перевірці взаємодії між різними модулями та підсистемами. Це дало змогу переконатися, що всі частини системи безперебійно функціонують разом, обмінюючись даними та виконуючи спільні операції без конфліктів.

Кульмінацією стала фаза системного тестування, де програмне забезпечення було протестоване як єдине ціле в умовах, максимально наближених до реальної експлуатації. На цьому етапі ми перевірили відповідність системи всім вимогам — від виконання основних бізнес-функцій до забезпечення продуктивності, безпеки та зручності використання. Особливу увагу було приділено перевірці обробки різноманітних сценаріїв використання, включно з граничними випадками та некоректними вхідними даними, що дозволило підвищити стійкість системи до непередбачуваних ситуацій.

Усі виявлені під час тестування недоліки, від незначних відхилень до критичних дефектів, були ретельно задокументовані, проаналізовані та оперативно виправлені. Цей ітераційний процес тестування та виправлення

помилок значно поліпшив загальну якість, стабільність та надійність розробленої системи.

Підсумовуючи, результати комплексного тестування дають нам повну впевненість у тому, що програмне забезпечення є високоякісним, повністю функціональним та готовим до успішного впровадження. Воно здатне ефективно вирішувати поставлені задачі та відповідати очікуванням користувачів.

ВИСНОВКИ

У рамках цього дипломного проєкту було успішно розроблено та всебічно протестовано Telegram-бота "Помічничок мандрівника Genshin Impact", що є ефективним інформаційно-довідковим рішенням для гравців популярної гри.

Аналіз існуючих аналогів, представлений у Розділі 1, чітко продемонстрував недоліки традиційних вебресурсів та мобільних додатків у контексті швидкого доступу до актуальної інформації. Вибір на користь Telegram-бота був ґрунтовно обґрунтований завдяки його інтеграції у повсякденну комунікацію, зручності використання, швидкості доступу та можливості персоналізації. Це дозволило спроєктувати рішення, яке усуває ключові недоліки інших форматів і відповідає нагальним потребам цільової аудиторії.

Розділ 2 присвячений детальному огляду та обґрунтуванню вибору технологій та інструментів. Мова програмування Python, у поєднанні з бібліотеками python-telegram-bot (telegram та telegram.ext) для взаємодії з API, asyncio для асинхронної обробки запитів, а також apscheduler для реалізації функціональності, залежної від часу, сформували надійний та ефективний технологічний стек. Використання бібліотек для роботи з даними (json, os, collections, datetime) та локалізації (localization.py) забезпечило гнучкість, масштабованість та зручність використання бота для багатомовної аудиторії.

Розділ 3 детально описує архітектуру та реалізацію ключових функціональних компонентів бота. Було представлено логіку керування меню та навігацією, механізми відображення інформації (текст, фото, медіа), а також обробку різноманітних дій користувача та спеціалізованих функцій (дні народжень, промокоди, "Що сьогодні" тощо). Окрема увага до адміністративних функцій та організації файлів підкреслює комплексний підхід до розробки. Функція handle_message визначена як центральний елемент маршрутизації, що забезпечує високу інтерактивність бота.

Нарешті, у Розділі 4 було проведено всебічне тестування системи. Модульне, інтеграційне та системне тестування підтвердили повну функціональність, стабільність та надійність розробленого програмного забезпечення. Оперативне виправлення виявлених дефектів значно підвищило якість продукту. Результати тестування свідчать про готовність бота до успішного впровадження.

У підсумку, створений Telegram-бот "Помічничок мандрівника Genshin Impact" є ефективним, зручним та інтерактивним інструментом, що здатен задовольнити інформаційні потреби гравців Genshin Impact, надаючи швидкий та актуальний доступ до довідкових даних. Проєкт продемонстрував глибоке розуміння сучасних підходів до розробки чат-ботів та готовність застосовувати передові технології для вирішення практичних завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Domashenko, I. S. (2025). -IO_11_Domashenko_Helper_bot: Репозиторій інформаційно-довідкового сервісу для відеогри. — Електронний ресурс. — Режим доступу: https://github.com/IvanDomashenko/-IO_11_Domashenko_Helper_bot.
2. Chatbot use cases // IBM. — Електронний ресурс. — Режим доступу: https://www.ibm.com/think/topics/chatbot-use-cases.
3. Chatbots // IBM. — Електронний ресурс. — Режим доступу: https://www.ibm.com/think/topics/chatbots.
4. The bot revolution (Archived 2019-12-12) // Telegram Blog. — Електронний ресурс. — Режим доступу: https://web.archive.org/web/20191212053626/https://telegram.org/blog/bot-revolution.
5. Віртуальний помічник // Вікіпедія. — Електронний ресурс. — Режим доступу: https://uk.wikipedia.org/wiki/Віртуальний_помічник.
6. Чат-бот // Вікіпедія. — Електронний ресурс. — Режим доступу: https://uk.wikipedia.org/wiki/%D0%A7%D0%B0%D1%82-%D0%B1%D0%BE%D1%82.
7. Інформаційна система // Вікіпедія. — Електронний ресурс. — Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0.
8. Bot API // Telegram. — Електронний ресурс. — Режим доступу: https://core.telegram.org/bots/api.
9. Bots: An introduction for developers // Telegram. — Електронний ресурс. — Режим доступу: https://core.telegram.org/bots.

10. Welcome to Python Telegram Bot's documentation! // python-telegram-bot. — Електронний ресурс. — Режим доступу: https://python-telegram-bot.readthedocs.io/en/stable/.
11. Genshin Impact Official Website // HoYoverse. — Електронний ресурс. — Режим доступу: https://genshin.hoyoverse.com/.
12. Genshin Impact Wiki // Fandom. — Електронний ресурс. — Режим доступу: https://genshin-impact.fandom.com/wiki/Genshin_Impact_Wiki.
13. The History of Chatbots (Archived 2020-07-28) // Onlim. — Електронний ресурс. — Режим доступу: https://web.archive.org/web/20200728194437/https://onlim.com/en/the-history-of-chatbots/.
14. Information service // ScienceDirect Topics. — Електронний ресурс. — Режим доступу: https://www.sciencedirect.com/topics/economics-econometrics-and-finance/information-service.
15. Al-Shayea, A. R. (2020). Chatbot systems: A review. *Journal of King Saud University - Computer and Information Sciences, 33*(2), 173-181. — Електронний ресурс. — Режим доступу: https://www.sciencedirect.com/science/article/pii/S2666827020300062?via%3Dihub.
16. Chatbots in Customer Service: A Complete Guide // Talkdesk. — Електронний ресурс. — Режим доступу: https://www.talkdesk.com/blog/chatbots-customer-service/.
17. Jawad, A. M., Hashim, F., & Al-Samarraie, H. (2022). A review of chatbot technologies for online learning: Opportunities and challenges. *Information, 13*(1), 41. — Електронний ресурс. — Режим доступу: https://www.mdpi.com/2078-2489/13/1/41.

ДОДАТОК 1

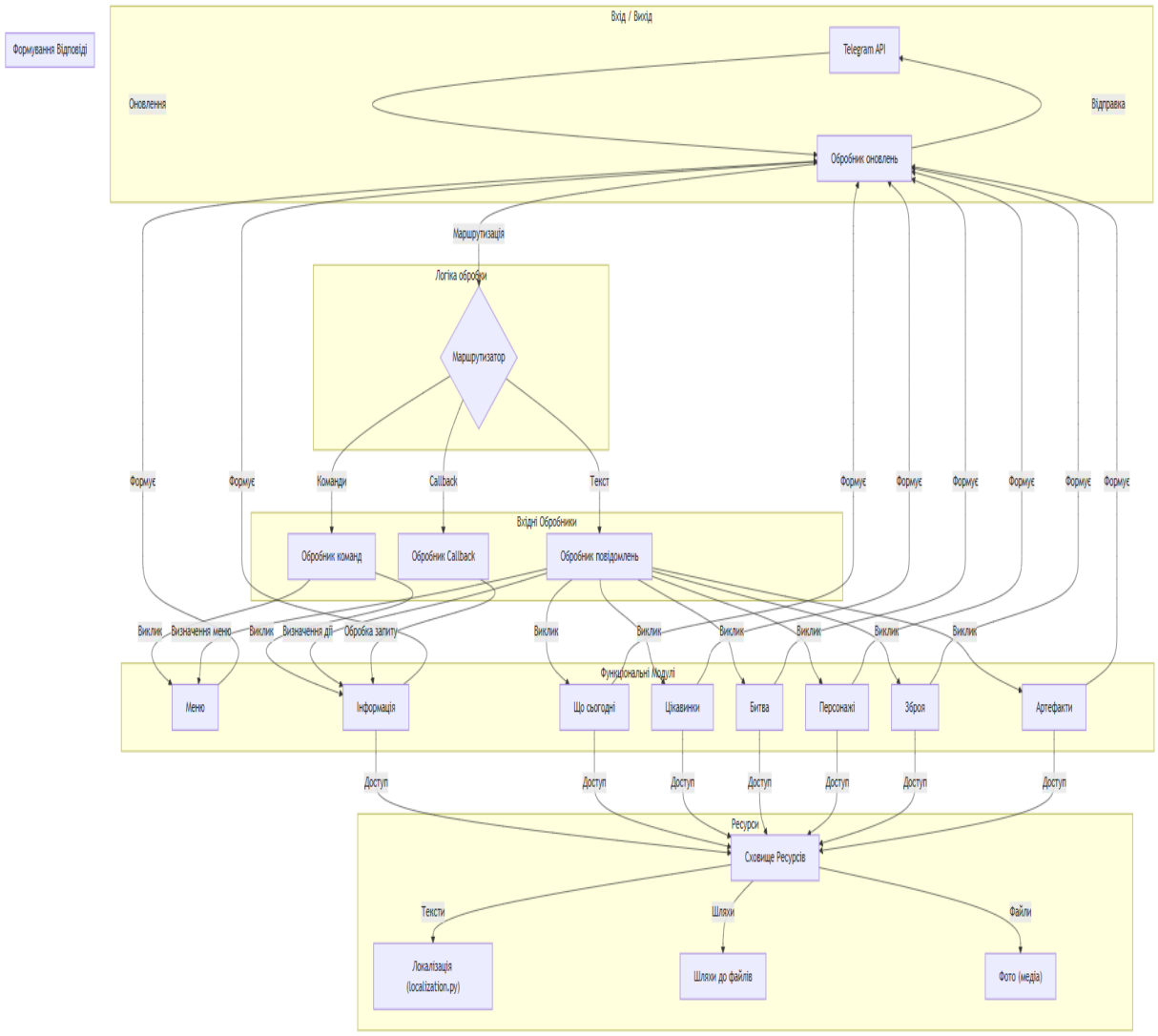
Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері

Схема внутрішньої архітектури системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2021 р



					ІАЛЦ.467200.004 Д1		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Домашенко І. С.			Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері **Структурна схема системи**	*Літ.*	*Арк.* 1 / *Акрушів* 1
Перевір.		Пономаренко А. М.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Пономаренко А.М.					
Затверд.							

ДОДАТОК 2

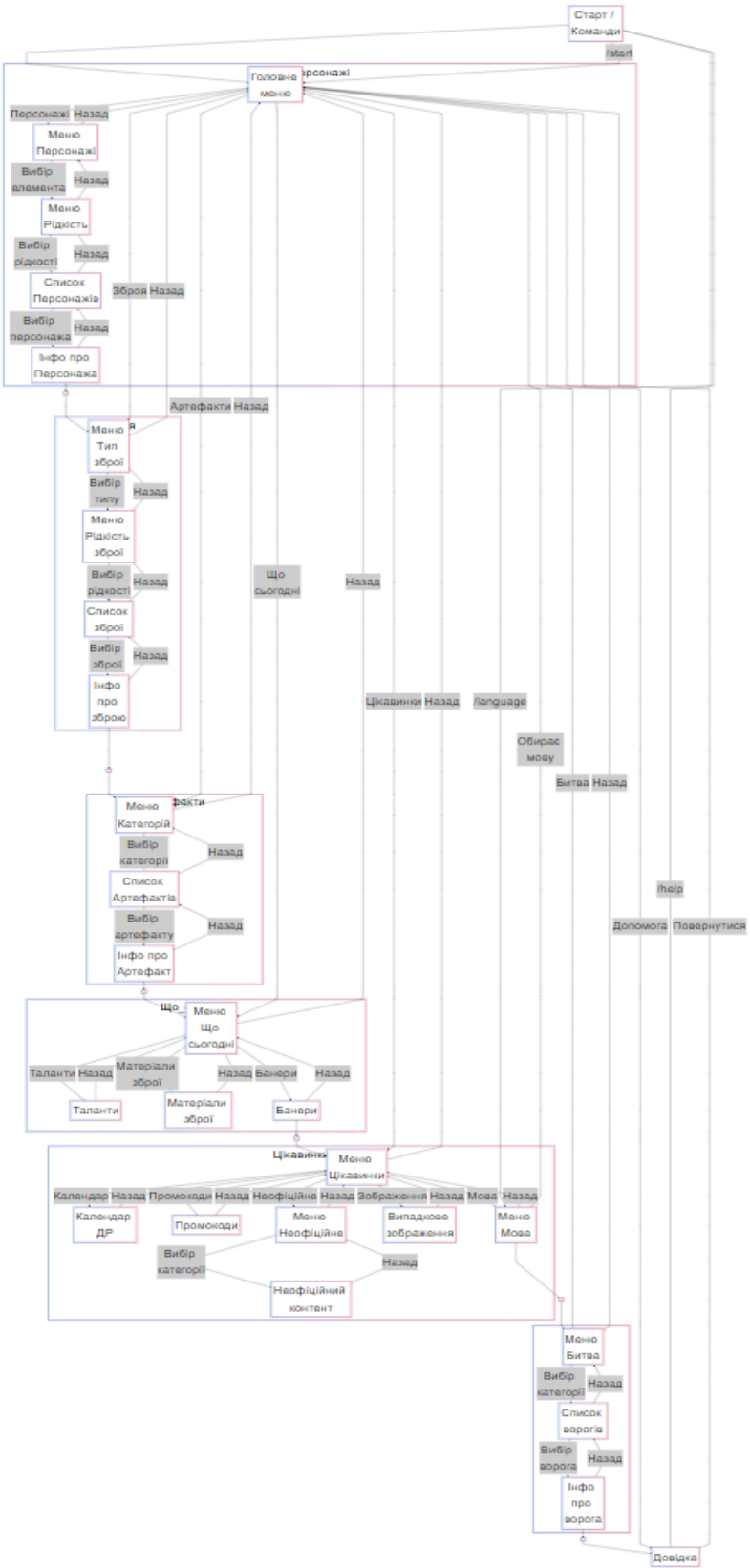
Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері

Схема сценаріїв використання системи

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2021 р



					ІАЛЦ.467200.004 Д2	
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розроб.		Домашенко І. С.			Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері **Функціональна схема системи**	*Літ.* / *Арк.* / *Акрушів*: 1 / 1
Перевір.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11
Н. Контр.		Пономаренко А.М.				
Затверд.						

ДОДАТОК З

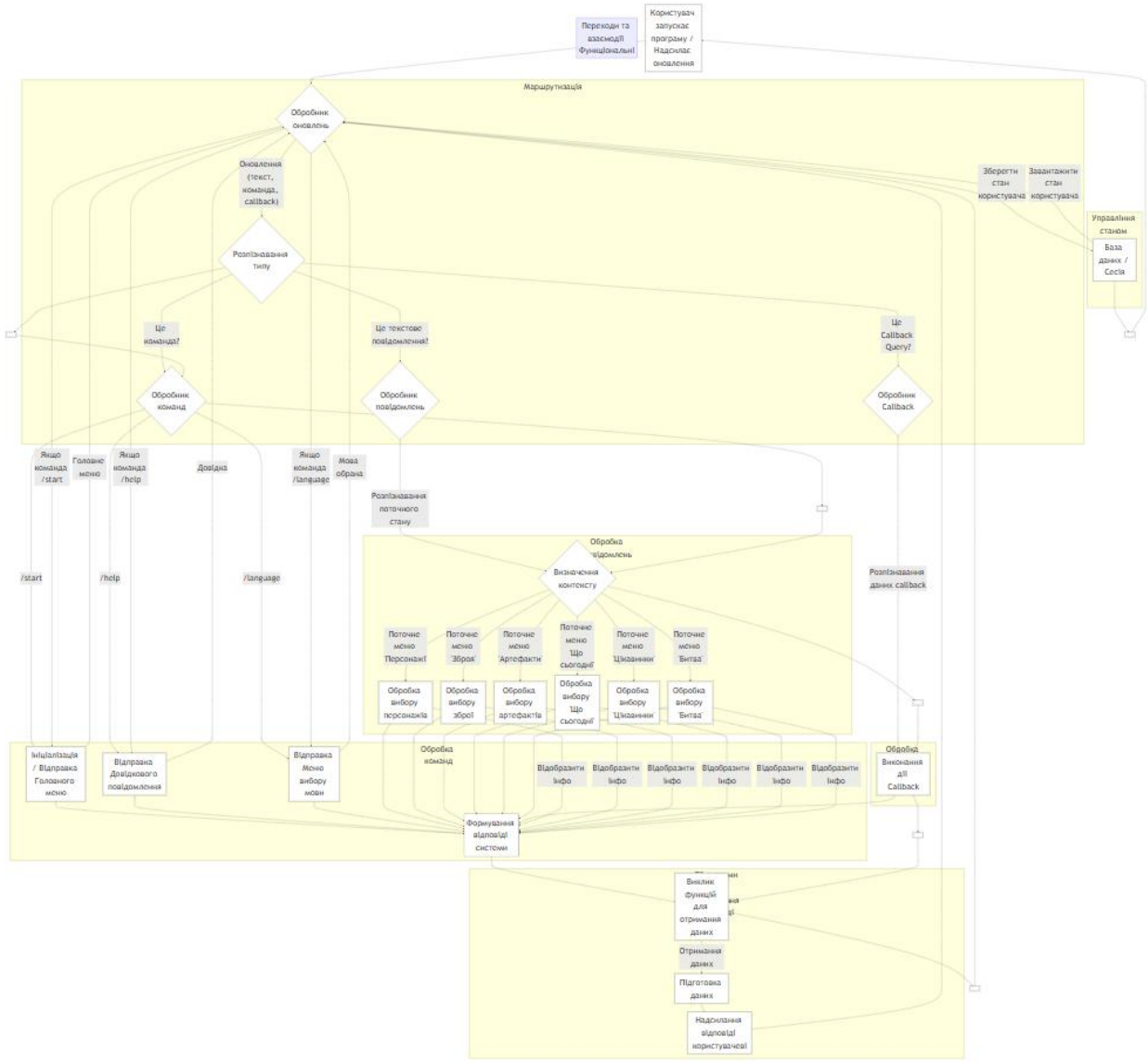
Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2025 р



Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Домашенко І. С.			Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері **Алгоритм дій програмного забезпечення**	*Літ.*	*Арк.* 1 / *Акрушів* 1
Перевір.		Пономаренко А. М.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-73	
Н. Контр.		Пономаренко А.М.					
Затверд.							

ДОДАТОК 4

Інформаційно-довідковий сервіс для відеогри з інтерфейсом у месенджері

Текст програмного коду

ІАЛЦ.467200.007 Д4

Київ 2025 р

Вміст файлу main_file.py

```
import asyncio
import sys
from telegram.ext import Updater, CommandHandler, MessageHandler, filters, Application
from character_data import fixed_character_order

from telegram.ext import Application, CommandHandler, MessageHandler, filters,
CallbackContext
from telegram import Update

from character_handlers import (
    show_character_menu,
    show_anemo_rarity_menu,
    show_pyro_rarity_menu,
    show_cryo_rarity_menu,
    show_electro_rarity_menu,
    show_geo_rarity_menu,
    show_hydro_rarity_menu,
    show_dendro_rarity_menu,
    show_anemo_4_star_characters,
    show_anemo_5_star_characters,
    show_pyro_4_star_characters,
    show_pyro_5_star_characters,
    show_cryo_4_star_characters,
    show_cryo_5_star_characters,
    show_electro_4_star_characters,
    show_electro_5_star_characters,
    show_geo_4_star_characters,
    show_geo_5_star_characters,
    show_hydro_4_star_characters,
    show_hydro_5_star_characters,
    show_dendro_4_star_characters,
    show_dendro_5_star_characters,
    # handle_character_menu_message,

handle_character_selection_from_list
)

from weapon_handlers import (
    show_weapon_menu,
    show_sword_rarity_menu,
    show_claymore_rarity_menu,
    show_polearm_rarity_menu,
    show_bow_rarity_menu,
    show_catalyst_rarity_menu,
    show_sword_5_star_weapons,
    show_sword_4_star_weapons,
    show_claymore_5_star_weapons,
    show_claymore_4_star_weapons,
    show_polearm_5_star_weapons,
    show_polearm_4_star_weapons,
```

```
    show_sword_4_star_weapons,
    show_claymore_5_star_weapons,
    show_claymore_4_star_weapons,
    show_polearm_5_star_weapons,
    show_polearm_4_star_weapons,
    show_bow_5_star_weapons,
    show_bow_4_star_weapons,
    show_catalyst_5_star_weapons,
    show_catalyst_4_star_weapons,
    handle_weapon_selection_from_list,
localized_back_claymore,
localized_back_sword,
localized_back_catalyst,
localized_back_polearm,
localized_back_bow,
show_weapon_types_menu,
show_weapon_rarity_menu,
show_weapon_photo,
)

from scheduler_service import start_scheduler  # Імпорт функції запуску розкладу
from telegram.ext import Application, CommandHandler, MessageHandler,
CallbackQueryHandler, filters
import asyncio
import threading
from artifacts_handler import show_artifacts_menu, send_artifact_category_media
from telegram import Update, InlineKeyboardMarkup, InlineKeyboardButton,
ReplyKeyboardMarkup, ReplyKeyboardRemove, \
        ReplyKeyboardRemove
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
from localization import localized_texts
from data import character_data

from telegram import Update
from telegram.ext import CallbackContext
from telegram import ReplyKeyboardMarkup, InlineKeyboardMarkup, InlineKeyboardButton
from data import character_data
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import CallbackContext
from data import character_data  # Імпортуємо дані з data.py
from collections import defaultdict
import telegram  # Додайте цей імпорт на початку файлу
from telegram import ReplyKeyboardMarkup, KeyboardButton
import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext

from telegram import ReplyKeyboardMarkup, KeyboardButton

import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext
  # Переконайтеся, що імпорт правильний


import os
from telegram import Update, InputMediaPhoto, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
```

```
from data import character_data

import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext
from data import character_data

from telegram import Update, ReplyKeyboardMarkup, InlineKeyboardButton,
InlineKeyboardMarkup, InputFile, InputMediaPhoto
# ... інші імпорти з telegram, якщо вони є ...

print(f"Python version: {sys.version}")



import os
import logging
import random
from telegram import Update, ReplyKeyboardMarkup, InlineKeyboardButton,
InlineKeyboardMarkup, InputFile
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup,
InputMediaPhoto
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters,
CallbackContext, CallbackQueryHandler, JobQueue
from data import character_names_uk, character_names_en, weapon_names_uk,
weapon_names_en, \
                 artifact_names_uk, artifact_names_en, location_names_uk,
location_names_en, \
                 character_data, weapon_data, artifact_data
from localization import localized_texts  # Імпортуємо локалізацію
from broadcast_handler import     init, subscribe, subscribe_list, unsubscribe_list,
is_subscribed, unsubscribe, \
    upload_command, handle_uploaded_photos, handle_upload_text, send_command, \
    send_promo_command, handle_promo_text  # Додані імпорти
from birthday_sender import send_birthday_greeting, load_character_birthdays #
Імпортуємо необхідні функції
import datetime
import asyncio
from scheduler_service import start_scheduler  # Імпортуємо функцію запуску розкладу
from telegram.ext import Application, CommandHandler, MessageHandler, filters,
PicklePersistence
from combat_handler import show_combat_menu, send_enemy_photos,
send_battle_photos_keyboard, get_enemies_data, show_enemies_menu, show_enemies_list
from curiosities_handler import show_curiosities_menu, show_unofficial_keyboard,
send_unofficial_media, \
    unofficial_characters, unofficial_banners, unofficial_locations,
unofficial_curiosities, \
    get_curiosities_keyboard, send_random_image_with_author, show_birthday_calendar, \
    handle_callback_query, language_command, set_language, show_promocodes

# logging.basicConfig(
#     format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
#     level=logging.INFO
# )

BANNER_IMAGES = {
    "character": ["D:\\Games\\дипломнаробота\\photos\\photo_2025-02-11_22-28-32.jpg",
"D:\\Games\\дипломнаробота\\photos\\photo_2025-05-01_14-03-43.jpg",
"D:\\Games\\дипломнаробота\\photos\\вм.png"],
```

```
    "weapon": ["D:\\Games\\дипломнаробота\\photos\\dog.jpg",
"D:\\Games\\дипломнаробота\\photos\\photo_2025-02-11_22-28-36.jpg"],
}

persistence = PicklePersistence(filepath="bot_data.pkl")
app = Application.builder().token('BOT_TOKEN').persistence(persistence).build()

TOKEN = os.environ.get("TELEGRAM_BOT_TOKEN")
ADMIN_ID = None # Буде ініціалізовано
PHOTO_PATH = r'D:\Games\дипломнаробота\photos\photo_2025-02-11_22-28-21.jpg'
#
#old:::::::
async def start(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['today'],
localized_texts[user_language]['characters']],
        [localized_texts[user_language]['weapons'],
localized_texts[user_language]['artifacts']],
        [localized_texts[user_language]['curiosities'],
localized_texts[user_language]['combat']] # Замінили кнопки
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['greeting'], reply_markup=reply_markup)
    context.user_data.pop('current_menu', None)
    context.user_data.pop('selected_element', None)
    context.user_data.pop('selected_weapon_type', None)

# # old;;;;;;;;;

from telegram import ReplyKeyboardMarkup

async def show_today_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['today_talents']],
        [localized_texts[user_language]['today_weapons']],
        [localized_texts[user_language]['banners']],
        [localized_texts[user_language]['back_to_main_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['today_info'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'today_menu'

async def show_weapon_talents(update: Update, context: CallbackContext):
    today_weekday = datetime.datetime.now().weekday()
    photo_filename = f'{today_weekday}_talents.jpg'
    photo_path = os.path.join('days', photo_filename)

    if os.path.exists(photo_path):
        with open(photo_path, 'rb') as photo:
            caption = f"Сьогодні ({datetime.datetime.now().strftime('%A')}):"
            await context.bot.send_photo(chat_id=update.effective_chat.id,
photo=photo, caption=caption)
    else:
        await update.message.reply_text(f"Фото талантів зброї для
{datetime.datetime.now().strftime('%A')} не знайдено.")
```

```
async def send_media_group_with_caption(chat_id: int, image_paths: list, bot,
caption=None):
    print(f"Шляхи перед відправкою: {image_paths}") # Додайте цей рядок
    media = []
    for i, path in enumerate(image_paths):
        if os.path.exists(path):
            try:
                with open(path, 'rb') as f:
                    media.append(InputMediaPhoto(media=f.read(), caption=caption if i
== 0 else None))
            except Exception as e:
                print(f"Помилка при відкритті файлу {path}: {e}")
        else:
            print(f"Файл не знайдено: {path}")

    if media:
        try:
            await bot.send_media_group(chat_id=chat_id, media=media)
        except Exception as e:
            print(f"Помилка при відправці медіагрупи: {e}")
    elif caption:
        await bot.send_message(chat_id=chat_id, text=caption)
    else:
        user_language = context.user_data.get('language', 'uk')
        await bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_banners_found'])

async def show_current_banners(update: Update, context: CallbackContext):
    print("DEBUG (banners): Викликано show_current_banners")
    user_language = context.user_data.get('language', 'uk')
    print(f"DEBUG (banners): Мова користувача: {user_language}")
    character_image_paths = BANNER_IMAGES.get("character", [])
    weapon_image_paths = BANNER_IMAGES.get("weapon", [])
    print(f"DEBUG (banners): Шляхи банерів персонажів (до обробки):
{character_image_paths}")
    print(f"DEBUG (banners): Шляхи банерів зброї (до обробки): {weapon_image_paths}")

    if isinstance(weapon_image_paths, str):
        weapon_image_paths = [weapon_image_paths]
    print(f"DEBUG (banners): Кількість банерів персонажів для відправки:
{len(character_image_paths)}")
    print(f"DEBUG (banners): Шляхи банерів персонажів: {character_image_paths}")
    if character_image_paths:
        print(f"DEBUG (banners): Намагаюся відправити банери персонажів з підписом:
{localized_texts[user_language]['character_banners_caption']}")
        await send_media_group_with_caption(update.effective_chat.id,
character_image_paths, context.bot,
localized_texts[user_language]['character_banners_caption'])
        print("DEBUG (banners): Банери персонажів відправлено (або спроба відправки
завершена).")

    print(f"DEBUG (banners): Кількість банерів зброї для відправки:
{len(weapon_image_paths)}")
    print(f"DEBUG (banners): Шляхи банерів зброї: {weapon_image_paths}")
    if weapon_image_paths:
        print(f"DEBUG (banners): Намагаюся відправити банери зброї з підписом:
{localized_texts[user_language]['weapon_banners_caption']}")
        await send_media_group_with_caption(update.effective_chat.id,
```

```
weapon_image_paths, context.bot,
localized_texts[user_language]['weapon_banners_caption'])
        print("DEBUG (banners): Банери зброї відправлено (або спроба відправки
завершена).")
    elif not character_image_paths and not weapon_image_paths:
        print(f"DEBUG (banners): Зображень банерів не знайдено, відправляю
повідомлення: {localized_texts[user_language]['no_banners_found']}")
        await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['no_banners_found'])
    else:
        print("DEBUG (banners): Умова відсутності банерів не спрацювала, хоча один зі
списків порожній.")

async def handle_message(update: Update, context: CallbackContext):

    user_message = update.message.text
    user_language = context.user_data.get('language', 'uk')
    localized_back = localized_texts[user_language]['back']
    if 'history' not in context.user_data:
        context.user_data['history'] = []
    user_language = context.user_data.get('language', 'uk')
    user_text = update.message.text.lower()
    current_menu = context.user_data.get('current_menu')
    history = context.user_data.get('history', [])
    selected_element = context.user_data.get('selected_element')
    selected_rarity_text = context.user_data.get('selected_rarity_text')
    print(f"ПОЧАТОК ОБРОБКИ ПОВІДОМЛЕННЯ:")
    print(f"  Поточне меню: {current_menu}")
    print(f"  Текст повідомлення: {user_text}")
    print(f"  Історія: {history}")
    print(f"  selected_element: {selected_element}")
    print(f"  selected_rarity_text: {selected_rarity_text}")

    waiting_for_upload = context.user_data.get('waiting_for_upload', False)
    waiting_for_promo_text = context.user_data.get('waiting_for_promo_text', False) #
Додано перевірку
    awaiting_artifact_category = context.user_data.get('awaiting_artifact_category',
False) # Отримання прапорця

    if waiting_for_upload and update.message.chat.id == ADMIN_ID and user_text and not
user_text.startswith('/'):
        await handle_upload_text(update, context)
        return
    elif waiting_for_promo_text and update.message.chat.id == ADMIN_ID and user_text
and not user_text.startswith('/'):
        await handle_promo_text(update, context) # Обробка тексту промокоду
        return

    element_list_lower = [
        localized_texts[user_language]['anemo'].lower(),
        localized_texts[user_language]['electro'].lower(),
        localized_texts[user_language]['geo'].lower(),
        localized_texts[user_language]['cryo'].lower(),
        localized_texts[user_language]['pyro'].lower(),
        localized_texts[user_language]['hydro'].lower(),
        localized_texts[user_language]['dendro'].lower()
    ]

    if user_text == localized_texts[user_language]['characters'].lower():
```

```
        print("  Натиснуто 'Персонажі'. Виклик show_character_menu.")
        await show_character_menu(update, context)
    elif user_text == localized_texts[user_language]['weapons'].lower():
        await show_weapon_types_menu(update, context)
    elif user_text == localized_texts[user_language]['artifacts'].lower():
        from artifacts_handler import show_artifacts_menu
        await show_artifacts_menu(update, context)
        context.user_data['awaiting_artifact_category'] = True
    elif current_menu == 'artifacts_menu':
        if user_text ==
localized_texts[user_language]['artifact_attack_button'].lower():
            await send_artifact_category_media(update, context, "attack",
'artifact_attack_caption')
        elif user_text ==
localized_texts[user_language]['artifact_talent_bonus_button'].lower():
            await send_artifact_category_media(update, context, "talent_bonus",
'artifact_talent_bonus_caption')
        elif user_text ==
localized_texts[user_language]['artifact_elemental_bonus_button'].lower():
            await send_artifact_category_media(update, context,
"elemental_dmg_bonus",'artifact_elemental_bonus_caption')
        elif user_text ==
localized_texts[user_language]['artifact_elemental_mastery_button'].lower():
            await send_artifact_category_media(update, context,
"elemental_mastery",'artifact_elemental_mastery_caption')
        elif user_text ==
localized_texts[user_language]['artifact_healing_button'].lower():
            await send_artifact_category_media(update, context,
"healing",'artifact_healing_caption')
        elif user_text ==
localized_texts[user_language]['artifact_defensive_button'].lower():
            await send_artifact_category_media(update, context,
"defensive",'artifact_defensive_caption')
        elif user_text ==
localized_texts[user_language]['artifact_other_button'].lower():
            await send_artifact_category_media(update, context,
"other",'artifact_other_caption')
        elif user_text == localized_texts[user_language]['main_menu_button'].lower():
            await start(update, context)
            context.user_data.pop('current_menu', None)
        return

    elif user_text.lower() in [localized_texts['uk'].get('change_language_button',
'змінити мову').lower(), localized_texts['en'].get('change_language_button', 'change
language').lower(), localized_texts['uk'].get('change_language', 'змінити
мову').lower(), localized_texts['en'].get('change_language', 'change
language').lower()]:
        await language_command(update, context)
    elif user_text == 'українська' or user_text == 'english':
        await set_language(update, context)
    elif user_text == localized_texts[user_language]['today'].lower():
        await show_today_menu(update, context)
    elif user_text == localized_texts[user_language]['curiosities'].lower():
        await show_curiosities_menu(update, context)
    elif user_text == localized_texts[user_language]['combat'].lower(): # Нова кнопка
        await show_combat_menu(update, context)
    elif current_menu == 'today_menu':
        today_weekday = datetime.datetime.now().weekday()
        print(f"Текст повідомлення: {user_text}")
```

```
        print(f"Значення localized_texts['current_banners']:
{localized_texts[user_language]['current_banners']}")
        print(
            f"Значення localized_texts['current_banners'].lower():
{localized_texts[user_language]['current_banners'].lower()}")

        if user_text == localized_texts[user_language]['today_talents'].lower():
            photo_filename = f'{today_weekday}_talents.jpg'
            photo_path = os.path.join('days', photo_filename)
            if os.path.exists(photo_path):
                try:
                    with open(photo_path, 'rb') as photo:
                        caption = localized_texts[user_language][
                            'today_talents_caption']  # Додайте локалізований підпис
                        await context.bot.send_photo(chat_id=update.effective_chat.id,
photo=photo, caption=caption)
                except FileNotFoundError:
                    await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['photo_not_found'].format(
                                                       filename=photo_filename))
                except Exception as e:
                    await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['error_sending_photo'].format(
                                                       error=e))
            else:
                await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['photo_not_found'].format(
                                                  filename=photo_filename))

        elif user_text == localized_texts[user_language]['today_weapons'].lower():
            photo_filename = f'{today_weekday}_weapons.jpg'
            photo_path = os.path.join('days', photo_filename)
            if os.path.exists(photo_path):
                try:
                    with open(photo_path, 'rb') as photo:
                        caption = localized_texts[user_language][
                            'today_weapons_caption']  # Додайте локалізований підпис
                        await context.bot.send_photo(chat_id=update.effective_chat.id,
photo=photo, caption=caption)
                except FileNotFoundError:
                    await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['photo_not_found'].format(
                                                       filename=photo_filename))
                except Exception as e:
                    await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['error_sending_photo'].format(
                                                       error=e))
            else:
                await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['photo_not_found'].format(
                                                  filename=photo_filename))
        elif user_text == localized_texts[user_language]['current_banners'].lower():
                print("Умова для 'які зараз банери' виконана!")
```

```
                await show_current_banners(update, context)
        elif user_text == localized_texts[user_language]['back_to_main_menu'].lower():
            await start(update, context)
            context.user_data.pop('current_menu', None)
    elif current_menu == 'curiosities_menu':
        if user_text ==
localized_texts[user_language]['birthday_calendar_button'].lower():
            await show_birthday_calendar(update, context)
        elif user_text == localized_texts[user_language]['promocodes'].lower():
            await show_promocodes(update, context)
        elif user_text == localized_texts[user_language]['unofficial'].lower():
            await show_unofficial_keyboard(update, context)  # Викликаємо нове меню
        elif user_text == localized_texts[user_language]['pictures'].lower():
            await send_random_image_with_author(update, context)
        elif user_text == localized_texts[user_language]['change_language'].lower():
            await language_command(update, context)
        # elif user_text == localized_texts[user_language]['your_stats'].lower():
        #     await context.bot.send_message(chat_id=update.effective_chat.id,
text="Ваша статистика (розробка).")
        elif user_text == localized_texts[user_language]['back_to_main_menu'].lower():
            await start(update, context)
            context.user_data.pop('current_menu', None)
    elif current_menu == 'unofficial_menu':
            if user_text ==
localized_texts[user_language]['characters_button'].lower():
                await unofficial_characters(update, context)
            elif user_text ==
localized_texts[user_language]['future_banners_button'].lower():
                await unofficial_banners(update, context)
            elif user_text ==
localized_texts[user_language]['locations_button'].lower():
                await unofficial_locations(update, context)
            elif user_text ==
localized_texts[user_language]['curiosities_button'].lower():
                await unofficial_curiosities(update, context)  # Перевірте цей виклик
            elif user_text ==
localized_texts[user_language]['back_to_curiosities_text'].lower():
                context.user_data['current_menu'] = 'curiosities_menu'
                user_language = context.user_data.get('language', 'uk')
                await
update.message.reply_text(localized_texts[user_language]['back_to_curiosities_text'] +
"...",

reply_markup=get_curiosities_keyboard(user_language))

    elif current_menu == 'combat_menu':
        if user_text == localized_texts[user_language]['enemies'].lower():
            # Замініть на виклик send_battle_photos_keyboard з ключем підпису для
ворогів
            # await send_battle_photos_keyboard(update, context, 'enemies',
'enemies_caption')
            await show_enemies_menu(update, context)
        elif user_text == localized_texts[user_language]['abyss'].lower():
            await send_battle_photos_keyboard(update, context, 'abyss',
'abyss_caption')
        elif user_text == localized_texts[user_language]['theater'].lower():
            await send_battle_photos_keyboard(update, context, 'theater',
'theater_caption')
        elif user_text == localized_texts[user_language]['reactions'].lower():
```

```
            await send_battle_photos_keyboard(update, context, 'reactions',
'reactions_caption')
        elif user_text == localized_texts[user_language]['back_to_main_menu'].lower():
            await start(update, context)
            context.user_data.pop('current_menu', None)

    elif current_menu == 'enemies_menu':
        if user_text == localized_texts[user_language]['bosses'].lower():
            await show_enemies_list(update, context, 'bosses')
        elif user_text == localized_texts[user_language]['common_enemies'].lower():
            await show_enemies_list(update, context, 'common')
        elif user_text == localized_texts[user_language]['elite_enemies'].lower():
            await show_enemies_list(update, context, 'elite')
        elif user_text ==
localized_texts[user_language]['back_to_combat_menu'].lower():
            await show_combat_menu(update, context)
            context.user_data['current_menu'] = 'combat_menu'
    elif current_menu and current_menu.startswith('enemies_list_'):
            print(f"Обробка повідомлення в стані: {current_menu}, текст: {user_text}")
            enemy_type = current_menu.split('_')[2]
            if user_text.lower() ==
localized_texts[user_language].get('back_to_enemies_menu', "").lower():
                await show_enemies_menu(update, context)
                context.user_data['current_menu'] = 'enemies_menu'
            else:
                await send_enemy_photos(update, context, enemy_type, user_text)

    elif current_menu == 'character_elements':
            selected_element = user_message.lower()
            context.user_data['selected_element'] = selected_element
            if user_message == localized_texts[user_language]['anemo']:
                await show_anemo_rarity_menu(update, context)
            elif user_message == localized_texts[user_language]['pyro']:
                await show_pyro_rarity_menu(update, context)
            elif user_message == localized_texts[user_language]['cryo']:
                await show_cryo_rarity_menu(update, context)
            elif user_message == localized_texts[user_language]['electro']:
                await show_electro_rarity_menu(update, context)
            elif user_message == localized_texts[user_language]['geo']:
                await show_geo_rarity_menu(update, context)
            elif user_message == localized_texts[user_language]['hydro']:
                await show_hydro_rarity_menu(update, context)
            elif user_message == localized_texts[user_language]['dendro']:
                await show_dendro_rarity_menu(update, context)
            elif user_message == localized_back:
                await start(update, context)

    elif context.user_data.get('current_menu') == 'anemo_rarity':
        if user_message == f"5★ {localized_texts[user_language]['anemo']}":
            await show_anemo_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['anemo']}":
            await show_anemo_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') == 'pyro_rarity':
        if user_message == f"5★ {localized_texts[user_language]['pyro']}":
            # Зберігаємо обрану рідкість (5)
```

```
            context.user_data['selected_rarity'] = '5'
            await show_pyro_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['pyro']}":
            # Зберігаємо обрану рідкість (4)
            context.user_data['selected_rarity'] = '4'
            await show_pyro_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') == 'cryo_rarity':
        if user_message == f"5★ {localized_texts[user_language]['cryo']}":
            await show_cryo_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['cryo']}":
            await show_cryo_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') == 'electro_rarity':
        if user_message == f"5★ {localized_texts[user_language]['electro']}":
            await show_electro_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['electro']}":
            await show_electro_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') == 'geo_rarity':
        if user_message == f"5★ {localized_texts[user_language]['geo']}":
            await show_geo_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['geo']}":
            await show_geo_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') == 'hydro_rarity':
        if user_message == f"5★ {localized_texts[user_language]['hydro']}":
            await show_hydro_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['hydro']}":
            await show_hydro_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') == 'dendro_rarity':
        if user_message == f"5★ {localized_texts[user_language]['dendro']}":
            await show_dendro_5_star_characters(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['dendro']}":
            await show_dendro_4_star_characters(update, context)
        elif user_message == localized_back:
            await show_character_menu(update, context)

    elif context.user_data.get('current_menu') in ['anemo_4_star_list',
'anemo_5_star_list',
                                                   'pyro_4_star_list',
'pyro_5_star_list',
                                                   'cryo_4_star_list',
'cryo_5_star_list',
                                                   'electro_4_star_list',
```

```
'electro_5_star_list',
                                                    'geo_4_star_list',
'geo_5_star_list',
                                                    'hydro_4_star_list',
'hydro_5_star_list',
                                                    'dendro_4_star_list',
'dendro_5_star_list']:
        print(f"Поточне меню: {context.user_data.get('current_menu')}")
        print(f"Отримане повідомлення: {user_message}")
        if user_message.endswith(" 🌪️"):
            await show_anemo_rarity_menu(update, context)
        elif user_message.endswith(" 🔥"):
            await show_pyro_rarity_menu(update, context)
        elif user_message.endswith(" 🧊"):
            await show_cryo_rarity_menu(update, context)
        elif user_message.endswith(" ⚡"):
            await show_electro_rarity_menu(update, context)
        elif user_message.endswith(" 🗿"):
            await show_geo_rarity_menu(update, context)
        elif user_message.endswith(" 💧"):
            await show_hydro_rarity_menu(update, context)
        elif user_message.endswith(" 🌱"):
            await show_dendro_rarity_menu(update, context)
        else:
            current_menu = context.user_data.get('current_menu')
            element = current_menu.split('_')[0]
            rarity_part = current_menu.split('_')[1]
            rarity = rarity_part.split('_')[0]  # Беремо тільки першу частину (4 або
5)
            print(f"Елемент: {element}, Рідкість: {rarity}")
            context.user_data['selected_element_localized'] = localized_texts[
                context.user_data.get('language', 'uk')].get(element)  # Встановлюємо
localized_element
            context.user_data['selected_rarity'] = rarity  # Встановлюємо
selected_rarity
            await handle_character_selection_from_list(update, context, element,
rarity)

    elif current_menu == 'weapon_types':
        selected_weapon_type = user_message.lower()
        context.user_data['selected_weapon_type'] = selected_weapon_type
        if user_message == localized_texts[user_language]['sword']:
            await show_sword_rarity_menu(update, context)
        elif user_message == localized_texts[user_language]['claymore']:
            await show_claymore_rarity_menu(update, context)
        elif user_message == localized_texts[user_language]['polearm']:
            await show_polearm_rarity_menu(update, context)
        elif user_message == localized_texts[user_language]['bow']:
            await show_bow_rarity_menu(update, context)
        elif user_message == localized_texts[user_language]['catalyst']:
            await show_catalyst_rarity_menu(update, context)
        elif user_message == localized_back:  # Додано обробку загальної кнопки
"Назад"
            await start(update, context)

    elif context.user_data.get('current_menu') == 'sword_rarity':
        if user_message == f"5★ {localized_texts[user_language]['sword']}":
```

```
            await show_sword_5_star_weapons(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['sword']}":
            await show_sword_4_star_weapons(update, context)
        elif user_message == localized_back_sword:
            await show_weapon_types_menu(update, context)

    elif context.user_data.get('current_menu') == 'claymore_rarity':
        if user_message == f"5★ {localized_texts[user_language]['claymore']}":
            await show_claymore_5_star_weapons(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['claymore']}":
            await show_claymore_4_star_weapons(update, context)
        elif user_message == localized_back_claymore:
            await show_weapon_types_menu(update, context)

    elif context.user_data.get('current_menu') == 'polearm_rarity':
        if user_message == f"5★ {localized_texts[user_language]['polearm']}":
            await show_polearm_5_star_weapons(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['polearm']}":
            await show_polearm_4_star_weapons(update, context)
        elif user_message == localized_back_polearm:
            await show_weapon_types_menu(update, context)

    elif context.user_data.get('current_menu') == 'bow_rarity':
        if user_message == f"5★ {localized_texts[user_language]['bow']}":
            await show_bow_5_star_weapons(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['bow']}":
            await show_bow_4_star_weapons(update, context)
        elif user_message == localized_back_bow:
            await show_weapon_types_menu(update, context)

    elif context.user_data.get('current_menu') == 'catalyst_rarity':
        if user_message == f"5★ {localized_texts[user_language]['catalyst']}":
            await show_catalyst_5_star_weapons(update, context)
        elif user_message == f"4★ {localized_texts[user_language]['catalyst']}":
            await show_catalyst_4_star_weapons(update, context)
        elif user_message == localized_back_catalyst:
            await show_weapon_types_menu(update, context)

    elif context.user_data.get('current_menu') in ['sword_4_star_list',
'sword_5_star_list',
                                                   'claymore_4_star_list',
'claymore_5_star_list',
                                                   'polearm_4_star_list',
'polearm_5_star_list',
                                                   'bow_4_star_list',
'bow_5_star_list',
                                                   'catalyst_4_star_list',
'catalyst_5_star_list']:
        print(f"Поточне меню: {context.user_data.get('current_menu')}")
        print(f"Отримане повідомлення: {user_message}")
        if user_message.endswith(" 🗡"):
            await show_sword_rarity_menu(update, context)
        elif user_message.endswith(" 🔪"):
            await show_claymore_rarity_menu(update, context)
        elif user_message.endswith(" 🧹"):
            await show_polearm_rarity_menu(update, context)
```

```
        elif user_message.endswith(" 🏹"):
            await show_bow_rarity_menu(update, context)
        elif user_message.endswith(" 📖"):
            await show_catalyst_rarity_menu(update, context)
        else:
            current_menu = context.user_data.get('current_menu')
            weapon_type = current_menu.split('_')[0]
            rarity_part = current_menu.split('_')[1]
            rarity = rarity_part.split('_')[0]
            print(f"Тип зброї: {weapon_type}, Рідкість: {rarity}")
            context.user_data['selected_weapon_type_localized'] = localized_texts[
                context.user_data.get('language', 'uk')].get(weapon_type)
            context.user_data['selected_weapon_rarity'] = rarity
            await handle_weapon_selection_from_list(update, context, weapon_type,
rarity, user_message)

    else:
        await start(update, context)

# async def show_promocodes(update: Update, context: CallbackContext):
#     user_language = context.user_data.get('language', 'uk')
#     user_id = update.effective_chat.id
#     message = localized_texts[user_language]['promocodes_info'] + "\n\n" +
localized_texts[user_language]['subscribe_promocodes_offer']
#     keyboard = []
#     if is_subscribed(user_id, 'promocodes'):
#         keyboard =
[[InlineKeyboardButton(localized_texts[user_language]['unsubscribe_promocodes'],
callback_data='unsubscribe_promocodes')]]
#     else:
#         keyboard =
[[InlineKeyboardButton(localized_texts[user_language]['subscribe_promocodes'],
callback_data='subscribe_promocodes')]]
#     reply_markup = InlineKeyboardMarkup(keyboard)
#     if context.user_data.get('promocodes_message_id'):
#         await context.bot.edit_message_text(chat_id=user_id,
message_id=context.user_data['promocodes_message_id'], text=message,
reply_markup=reply_markup)
#     else:
#         sent_message = await context.bot.send_message(chat_id=user_id, text=message,
reply_markup=reply_markup)
#         context.user_data['promocodes_message_id'] = sent_message.message_id
#


async def help_command(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    help_text = localized_texts[user_language]['help_text']
    await update.message.reply_text(help_text)

def main():
    application = ApplicationBuilder().token(TOKEN).build()
    job_queue = application.job_queue

    global ADMIN_ID
    ADMIN_ID = int(os.environ.get("ADMIN_ID", 0))
    if ADMIN_ID == 0:
        print("Будь ласка, встановіть змінну середовища ADMIN_ID")
```

```
        exit()

    from broadcast_handler import subscribe, upload_command, handle_uploaded_photos,
handle_upload_text, send_command, \
        init, subscribe_list, unsubscribe_list, is_subscribed, unsubscribe,
send_promo_command
    init(ADMIN_ID)

    # Запуск планувальника завдань з окремого файлу
    start_scheduler(application.bot)

    # Обробники команд
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))
    application.add_handler(CommandHandler("language", language_command))
    application.add_handler(CommandHandler('subscribe', subscribe))
    application.add_handler(CommandHandler('upload', upload_command))
    application.add_handler(CommandHandler('send', send_command))
    application.add_handler(CommandHandler('unsubscribe', unsubscribe))
    application.add_handler(CommandHandler('send_promo', send_promo_command))

    # Обробник callback query
    application.add_handler(CallbackQueryHandler(handle_callback_query))

    # Обробники повідомлень
    application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
handle_message))
    application.add_handler(MessageHandler(filters.PHOTO, handle_uploaded_photos))


application.add_handler(MessageHandler(filters.Regex(localized_texts['uk']['characters
'] + '|' + localized_texts['en']['characters']), show_character_menu))

    application.add_handler(CallbackQueryHandler(show_weapon_photo,
pattern='^show_weapon_photo_')) # Обробник для фото зброї

    # Запуск бота
    application.run_polling()

if __name__ == '__main__':
    # Переконайтеся, що змінна TOKEN встановлена у вашому середовищі або безпосередньо
тут
    TOKEN = os.environ.get("TELEGRAM_BOT_TOKEN")
    if not TOKEN:
        print("Будь ласка, встановіть змінну середовища TELEGRAM_BOT_TOKEN")
        exit()
    main()
```

Вміст файлу curiosities_handler.py

```
import os
import sys
import asyncio
import datetime
```

```
import random
from collections import defaultdict
import telegram
from telegram import Update, ReplyKeyboardMarkup, InlineKeyboardButton,
InlineKeyboardMarkup, InputFile, InputMediaPhoto, KeyboardButton
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters,
CallbackContext, CallbackQueryHandler, JobQueue, Application, PicklePersistence
from data import character_names_uk, character_names_en, weapon_names_uk,
weapon_names_en, \
                 artifact_names_uk, artifact_names_en, location_names_uk,
location_names_en, \
                 character_data, weapon_data, artifact_data
from localization import localized_texts
from broadcast_handler import     init, subscribe, subscribe_list, unsubscribe_list,
is_subscribed, unsubscribe, \
    upload_command, handle_uploaded_photos, handle_upload_text, send_command, \
    send_promo_command, handle_promo_text
from birthday_sender import send_birthday_greeting, load_character_birthdays
from scheduler_service import start_scheduler
from artifacts_handler import show_artifacts_menu, send_artifact_category_media
from character_data import fixed_character_order

PHOTO_PATH = r'D:\Games\дипломнаробота\photos\photo_2025-02-11_22-28-21.jpg'

async def show_curiosities_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['birthday_calendar_button'],
localized_texts[user_language]['promocodes']],
        [localized_texts[user_language]['unofficial'],
localized_texts[user_language]['pictures']],

[localized_texts[user_language]['change_language'],localized_texts[user_language]['bac
k_to_main_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['curiosities_title'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'curiosities_menu'

UNOFFICIAL_FOLDER = "D:\\Games\\дипломнаробота\\unofficial"  # Замініть на свій шлях
BANNERS_SUBFOLDER = "banners"

async def show_unofficial_keyboard(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
```

```
    keyboard = [
        [localized_texts[user_language]['characters_button'],
localized_texts[user_language]['future_banners_button']],
        [localized_texts[user_language]['locations_button'],
localized_texts[user_language]['curiosities_button']],
        [localized_texts[user_language]['back_to_curiosities_text']], # Використовуйте
існуючий текст "Назад"
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['unofficial_menu_caption'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'unofficial_menu'

UNOFFICIAL_FOLDER = "D:\\Games\\дипломнаробота\\unofficial"  # Замініть на свій шлях

async def send_unofficial_media(update: Update, context: CallbackContext, subfolder:
str, caption_key: str):
    chat_id = update.message.chat.id
    media_path = os.path.join(UNOFFICIAL_FOLDER, subfolder)
    user_language = context.user_data.get('language', 'uk') # Отримуємо мову
користувача

    try:
        all_files = os.listdir(media_path)
        image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg',
'.png', '.gif'))]

        if not image_files:
            await context.bot.send_message(chat_id=chat_id, text=f"У папці
'unofficial/{subfolder}' не знайдено зображень.")
            return

        media_group = []
        for image_file in image_files:
            image_path = os.path.join(media_path, image_file)
            with open(image_path, 'rb') as photo_file:
                # Отримуємо локалізований підпис за допомогою переданого ключа
                caption_text = localized_texts[user_language].get(caption_key, "")
                media_group.append(InputMediaPhoto(photo_file.read(),
caption=caption_text if not media_group else None))

        if media_group:
            if len(media_group) == 1:
```

```
                await context.bot.send_photo(chat_id=chat_id,
photo=media_group[0].media, caption=media_group[0].caption)
            else:
                await context.bot.send_media_group(chat_id=chat_id, media=media_group)

    except FileNotFoundError:
        await context.bot.send_message(chat_id=chat_id, text=f"Папку '{media_path}' не
знайдено.")
    except Exception as e:
        print(f"Помилка при відправці неофіційних {subfolder}: {e}")
        await context.bot.send_message(chat_id=chat_id, text=f"Виникла помилка при
відображенні неофіційних {subfolder}.")

async def unofficial_characters(update: Update, context: CallbackContext):
    await send_unofficial_media(update, context, "characters", 'FUTURE_CAPTION')

async def unofficial_banners(update: Update, context: CallbackContext):
    await send_unofficial_media(update, context, "banners", 'FUTURE_CAPTION')

async def unofficial_locations(update: Update, context: CallbackContext):
    await send_unofficial_media(update, context, "locations", 'FUTURE_CAPTION')

async def unofficial_curiosities(update: Update, context: CallbackContext):
    print("Функція unofficial_curiosities викликана!")
    await send_unofficial_media(update, context, "curiosities", 'FUTURE_CAPTION')

def get_curiosities_keyboard(user_language):
    keyboard = [
        [localized_texts[user_language]['birthday_calendar_button'],
localized_texts[user_language]['promocodes']],
        [localized_texts[user_language]['unofficial'],
localized_texts[user_language]['pictures']],

[localized_texts[user_language]['your_stats'],localized_texts[user_language]['change_l
anguage']],
        [KeyboardButton(localized_texts[user_language]['back_to_main_menu'])],
    ]
    return ReplyKeyboardMarkup(keyboard, resize_keyboard=True, one_time_keyboard=True)

IMAGE_FOLDER = 'D:\\Games\\дипломнаробота\\random_photos'

AUTHOR_DELIMITER = ";"  # Символ, що відокремлює ім'я автора в назві файлу

async def send_random_image_with_author(update: Update, context: CallbackContext):
    try:
```

```
        all_files = os.listdir(IMAGE_FOLDER)
        image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg',
'.png', '.gif'))]

        if not image_files:
            await context.bot.send_message(chat_id=update.effective_chat.id, text="У
папці не знайдено жодного зображення.")
            return

        random_image = random.choice(image_files)
        image_path = os.path.join(IMAGE_FOLDER, random_image)
        author = ""
        filename = os.path.splitext(random_image)[0]

        if filename.lower().startswith("photo"):
            author = "Офіційний арт"
        elif filename.lower().startswith("unknown"):
            author = "Автор невідомий"
        elif AUTHOR_DELIMITER in filename:
            author_part = filename.split(AUTHOR_DELIMITER)[0]
            author = f"Автор: {author_part.capitalize()}"
        else:
            # Якщо роздільник не знайдено, можете використати всю назву файлу або іншу
логіку
            parts = filename.split('_')
            if parts:
                author = f"Автор: {parts[0].capitalize()}"
            else:
                author = "Інформація про автора відсутня"

        with open(image_path, 'rb') as photo:
            caption = author
            await context.bot.send_photo(chat_id=update.effective_chat.id,
photo=photo, caption=caption)

    except FileNotFoundError:
        await context.bot.send_message(chat_id=update.effective_chat.id, text=f"Папку
'{IMAGE_FOLDER}' не знайдено.")
    except Exception as e:
        print(f"Помилка при відправці випадкового зображення з автором: {e}")
        await context.bot.send_message(chat_id=update.effective_chat.id, text="Виникла
помилка при відправці зображення.")

async def show_birthday_calendar(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
```

```
    chat_id = update.effective_chat.id
    if not isinstance(chat_id, int):
        chat_id = update.effective_user.id

    message = localized_texts[user_language]['birthday_calendar_info']
    is_subscribed_user = is_subscribed(chat_id, 'birthdays')
    keyboard =
[[InlineKeyboardButton(localized_texts[user_language]['unsubscribe_birthday_calendar']
if is_subscribed_user else
localized_texts[user_language]['subscribe_birthday_calendar'],
                                       callback_data='unsubscribe_birthday' if
is_subscribed_user else 'subscribe_birthday')]]
    reply_markup = InlineKeyboardMarkup(keyboard)

    if not context.user_data.get('birthday_message_id'):
        # Перше відображення: надсилаємо фото та повідомлення
        try:
            with open(PHOTO_PATH, 'rb') as photo_file:
                sent_message = await context.bot.send_photo(chat_id=chat_id,
photo=photo_file, caption=message, reply_markup=reply_markup)
                context.user_data['birthday_message_id'] = sent_message.message_id
        except FileNotFoundError:
            sent_message = await context.bot.send_message(chat_id=chat_id,
text=f"{message}\n\n{localized_texts[user_language]['photo_not_found']}",
reply_markup=reply_markup)
            context.user_data['birthday_message_id'] = sent_message.message_id
    else:
        # Повторне відображення (після натискання кнопки): редагуємо лише клавіатуру
        try:
            await context.bot.edit_message_reply_markup(chat_id=chat_id,
message_id=context.user_data['birthday_message_id'], reply_markup=reply_markup)
        except telegram.error.BadRequest as e:
            if "Message is not modified" not in str(e):
                raise e

async def show_promocodes(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    user_id = update.effective_chat.id
    message = localized_texts[user_language]['promocodes_info'] + "\n\n" +
localized_texts[user_language]['subscribe_promocodes_offer']
    keyboard = []
    if is_subscribed(user_id, 'promocodes'):
        keyboard =
[[InlineKeyboardButton(localized_texts[user_language]['unsubscribe_promocodes'],
callback_data='unsubscribe_promocodes')]]
```

```
    else:
        keyboard =
[[InlineKeyboardButton(localized_texts[user_language]['subscribe_promocodes'],
callback_data='subscribe_promocodes')]]
    reply_markup = InlineKeyboardMarkup(keyboard)
    if context.user_data.get('promocodes_message_id'):
        await context.bot.edit_message_text(chat_id=user_id,
message_id=context.user_data['promocodes_message_id'], text=message,
reply_markup=reply_markup)
    else:
        sent_message = await context.bot.send_message(chat_id=user_id, text=message,
reply_markup=reply_markup)
        context.user_data['promocodes_message_id'] = sent_message.message_id

async def handle_callback_query(update: Update, context: CallbackContext):
    print("Викликано handle_callback_query")
    query = update.callback_query
    await query.answer()
    user_language = context.user_data.get('language', 'uk')
    user_id = update.effective_chat.id

    if query.data == 'subscribe_birthday':
        await subscribe_list(update, context, 'birthdays')
        await
query.message.reply_text(localized_texts[user_language]['subscribed_calendar']) #
Додано повідомлення про підписку на календар
        await show_birthday_calendar(update, context) # Оновлюємо відображення
календаря
    elif query.data == 'unsubscribe_birthday':
        await unsubscribe_list(update, context, 'birthdays')
        await
query.message.reply_text(localized_texts[user_language]['unsubscribed_calendar']) #
Додано повідомлення про відписку від календаря
        await show_birthday_calendar(update, context) # Оновлюємо відображення
календаря
    elif query.data == 'subscribe_promocodes':
        await subscribe_list(update, context, 'promocodes')
        await
query.message.reply_text(localized_texts[user_language]['subscribed_success_promocodes
']) # Повідомлення про підписку на промокоди вже було
        await show_promocodes(update, context) # Оновлюємо відображення промокодів
    elif query.data == 'unsubscribe_promocodes':
        await unsubscribe_list(update, context, 'promocodes')
```

```
        await
query.message.reply_text(localized_texts[user_language]['unsubscribed_success_promocod
es']) # Повідомлення про відписку від промокодів вже було
        await show_promocodes(update, context) # Оновлюємо відображення промокодів
    elif query.data == 'birthday_photos': # Залишаємо для сумісності
        print("Отримано callback_data: birthday_photos")
    elif query.data == 'birthday_subscribe':
        await subscribe(update, context) # Загальна підписка
    elif query.data.startswith('show_bday_photo_'):
        pass
    elif query.data == 'curiosities_menu':
        await show_curiosities_menu(update, context)
    elif query.data == 'main_menu':
        await start(update, context)

async def language_command(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        ['Українська'],
        ['English']
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True,
one_time_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_language'], reply_markup=reply_markup)

async def set_language(update: Update, context: CallbackContext):
    language = 'uk' if update.message.text == 'Українська' else 'en'
    context.user_data['language'] = language
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[language]['language_changed'])
    await start(update, context)
```

Вміст файлу combat_handler.py

```
import os
import sys
import asyncio
import datetime
import random
from collections import defaultdict
import telegram
from telegram import Update, ReplyKeyboardMarkup, InlineKeyboardButton,
InlineKeyboardMarkup, InputFile, InputMediaPhoto, KeyboardButton
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters,
```

```
CallbackContext, CallbackQueryHandler, JobQueue, Application, PicklePersistence
from data import character_names_uk, character_names_en, weapon_names_uk,
weapon_names_en, \
                 artifact_names_uk, artifact_names_en, location_names_uk,
location_names_en, \
                 character_data, weapon_data, artifact_data
from localization import localized_texts
from broadcast_handler import     init, subscribe, subscribe_list, unsubscribe_list,
is_subscribed, unsubscribe, \
    upload_command, handle_uploaded_photos, handle_upload_text, send_command, \
    send_promo_command, handle_promo_text
from birthday_sender import send_birthday_greeting, load_character_birthdays
from scheduler_service import start_scheduler
from artifacts_handler import show_artifacts_menu, send_artifact_category_media
from character_data import fixed_character_order

async def show_combat_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['enemies'],
localized_texts[user_language]['abyss']],
        [localized_texts[user_language]['theater'],
localized_texts[user_language]['reactions']],
        [localized_texts[user_language]['back_to_main_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['combat_title'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'combat_menu'

BATTLE_PHOTO_FOLDER = 'combat'

async def send_battle_photos_keyboard(update: Update, context: CallbackContext,
subfolder: str, caption_key: str):
    chat_id = update.effective_chat.id
    full_path = os.path.join(BATTLE_PHOTO_FOLDER, subfolder)
    user_language = context.user_data.get('language', 'uk')
    caption_text = localized_texts[user_language].get(caption_key, "") # Отримуємо
локалізований текст підпису

    try:
        all_files = os.listdir(full_path)
        image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg',
'.png', '.gif', '.webp'))]
```

```
        if image_files:
            media_group = []
            for image_file in image_files:
                image_path = os.path.join(full_path, image_file)
                try:
                    with open(image_path, 'rb') as photo_file:
                        media_group.append(InputMediaPhoto(photo_file.read(),
caption=caption_text if not media_group else None)) # Використовуємо отриманий текст
підпису
                except Exception as e:
                    print(f"Помилка при читанні файлу {image_path}: {e}")
                    await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_reading_photo'].format(filename=image_file)
)
                    return

            if media_group:
                print(f"Надсилання {len(media_group)} фотографій з підписом:
'{caption_text}'")
                for i in range(0, len(media_group), 10):
                    chunk = media_group[i:i + 10]
                    try:
                        await context.bot.send_media_group(chat_id=chat_id,
media=chunk)
                    except Exception as e:
                        print(f"Помилка при відправці медіагрупи з підписом:
'{caption_text}': {e}")
                        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_sending_photos'].format(category=localized_
texts[user_language].get(subfolder, subfolder))) # Використовуємо назву підпапки для
повідомлення про помилку
                        return
            else:
                await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_in_folder'].format(folder=localized_tex
ts[user_language].get(subfolder, subfolder)))
        else:
            await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_in_folder'].format(folder=localized_tex
ts[user_language].get(subfolder, subfolder)))

    except FileNotFoundError:
        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['folder_not_found'].format(folder=localized_texts[
user_language].get(subfolder, subfolder), path=full_path))
```

```
    except Exception as e:
        print(f"Помилка при обробці папки '{subfolder}': {e}")
        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_processing_folder'].format(folder=localized
_texts[user_language].get(subfolder, subfolder)))

def get_enemies_data():
    return {
        'bosses': {
            'uk': ['Андріус', 'Аждаха', 'Синьйора'],
            'en': ['Andrius', 'Azhdaha', 'Signora'],
        },
        'common': {
            'uk': ['Хілічурл-стрілець', 'Велика слизь', 'Кам\'яна шкіра'],
            'en': ['Hilichurl Shooter', 'Large Slime', 'Stonehide Lawachurl'],
        },
        'elite': {
            'uk': ['Анемо борець', 'Кріо вісник Фатуї', 'Електро цицін маг'],
            'en': ['Anemo Samachurl', 'Cryo Cicin Mage', 'Electro Cicin Mage'],
        },
    }

async def show_enemies_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['bosses']],
        [localized_texts[user_language]['common_enemies']],
        [localized_texts[user_language]['elite_enemies']],
        [localized_texts[user_language]['back_to_combat_menu']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['enemies_menu_title'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'enemies_menu'

async def show_enemies_list(update: Update, context: CallbackContext, enemy_type:
str):
    user_language = context.user_data.get('language', 'uk')
    enemies_data = get_enemies_data() # Функція, яка повертає словник з даними про
ворогів
    enemy_list = enemies_data.get(enemy_type, {}).get(user_language, [])
    if not enemy_list:
        await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['no_enemies_found'].format(enemy_type=localized_te
xts[user_language].get(enemy_type, enemy_type)))
```

```
        return

    keyboard = [[enemy] for enemy in enemy_list]
    keyboard.append([localized_texts[user_language]['back_to_enemies_menu']])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['select_enemy'], reply_markup=reply_markup)
    context.user_data['current_menu'] = f'enemies_list_{enemy_type}'

async def send_enemy_photos(update: Update, context: CallbackContext, enemy_type: str,
enemy_name: str):
    chat_id = update.effective_chat.id
    full_path = os.path.join(BATTLE_PHOTO_FOLDER, 'enemies', enemy_type.lower(),
enemy_name.lower()) # Припускаємо таку структуру папок
    user_language = context.user_data.get('language', 'uk')

    try:
        all_files = os.listdir(full_path)
        image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg',
'.png', '.gif', '.webp'))]

        if image_files:
            media_group = []
            for image_file in image_files:
                image_path = os.path.join(full_path, image_file)
                try:
                    with open(image_path, 'rb') as photo_file:
                        caption =
localized_texts[user_language].get(enemy_name.lower(), enemy_name) # Підпис - ім'я
ворога
                        media_group.append(InputMediaPhoto(photo_file.read(),
caption=caption if not media_group else None))
                except Exception as e:
                    print(f"Помилка при читанні файлу {image_path}: {e}")
                    await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_reading_photo'].format(filename=image_file)
)
                    return

            if media_group:
                for i in range(0, len(media_group), 10):
                    chunk = media_group[i:i + 10]
                    try:
                        await context.bot.send_media_group(chat_id=chat_id,
media=chunk)
```

```
                    except Exception as e:
                        print(f"Помилка при відправці медіагрупи ворога
'{enemy_name}': {e}")
                        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_sending_photos'].format(category=enemy_name
))
                        return
            else:
                await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_for_enemy'].format(enemy=localized_text
s[user_language].get(enemy_name.lower(), enemy_name)))
        else:
            await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_for_enemy'].format(enemy=localized_text
s[user_language].get(enemy_name.lower(), enemy_name)))

    except FileNotFoundError:
        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['folder_not_found'].format(folder=enemy_name,
path=full_path))
    except Exception as e:
        print(f"Помилка при обробці папки ворога '{enemy_name}': {e}")
        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_processing_folder'].format(folder=enemy_nam
e))
```

Вміст файлу weapon_handlers.py

```
from telegram import Update, ReplyKeyboardMarkup, KeyboardButton
from telegram.ext import CallbackContext
from localization import localized_texts

localized_back = localized_texts['uk'].get('back', 'Назад') # Або інше значення за
замовчуванням
localized_back_sword = localized_back + " 🗡️"
localized_back_claymore = localized_back + " 🔪"
localized_back_polearm = localized_back + " 🪥"
localized_back_bow = localized_back + " 🏹"
localized_back_catalyst = localized_back + " 📖"

def get_weapon_data():
    return {
        'клятва_вільного_вітру': {
            'name': {
```

```
            'uk': 'Клятва вільного вітру',
            'en': 'Freedom-Sworn',
        },
        'type': {
            'uk': 'Одноручна зброя',
            'en': 'Sword',
        },
        'rarity': '5',
        'folder_name': 'freedom-sworn',
        # Можете додати інші характеристики зброї, якщо потрібно
    },
    'небесний_клинок': {
        'name': {
            'uk': 'Небесний клинок',
            'en': 'Skyward Blade',
        },
        'type': {
            'uk': 'Одноручна зброя',
            'en': 'Sword',
        },
        'rarity': '5',
        'folder_name': 'skyward_blade',
    },
    'залізний_жало': {
        'name': {
            'uk': 'Залізний жало',
            'en': 'Iron Sting',
        },
        'type': {
            'uk': 'Одноручна зброя',
            'en': 'Sword',
        },
        'rarity': '4',
        'folder_name': 'iron_sting',
    },
    'меч_жертов': {
        'name': {
            'uk': 'Меч жертов',
            'en': 'Sacrificial Sword',
        },
        'type': {
            'uk': 'Одноручна зброя',
            'en': 'Sword',
        },
        'rarity': '4',
```

```
        'folder_name': 'sacrificial_sword',
    },
    'дворучний_меч_жертов': {
        'name': {
            'uk': 'Дворучний меч жертов',
            'en': 'Sacrificial Greatsword',
        },
        'type': {
            'uk': 'Дворучна зброя',
            'en': 'Claymore',
        },
        'rarity': '4',
        'folder_name': 'sacrificial_greatsword',
    },
    'велика_палиця_змія': {
        'name': {
            'uk': 'Велика палиця-змія',
            'en': 'Serpent Spine',
        },
        'type': {
            'uk': 'Дворучна зброя',
            'en': 'Claymore',
        },
        'rarity': '4',
        'folder_name': 'serpent_spine',
    },
    'могила_вовка': {
        'name': {
            'uk': 'Могила вовка',
            'en': 'Wolf\'s Gravestone',
        },
        'type': {
            'uk': 'Дворучна зброя',
            'en': 'Claymore',
        },
        'rarity': '5',
        'folder_name': 'wolf\'s_gravestone',
    },
    'небесна_гордість': {
        'name': {
            'uk': 'Небесна гордість',
            'en': 'Skyward Pride',
        },
        'type': {
            'uk': 'Дворучна зброя',
```

```
                'en': 'Claymore',
            },
            'rarity': '5',
            'folder_name': 'skyward_pride',
        },
        # Додайте дані для іншої зброї
    }

async def show_weapon_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [KeyboardButton(localized_texts[user_language]['sword'])],
        [KeyboardButton(localized_texts[user_language]['claymore'])],
        [KeyboardButton(localized_texts[user_language]['polearm'])],
        [KeyboardButton(localized_texts[user_language]['bow'])],
        [KeyboardButton(localized_texts[user_language]['catalyst'])],
        [KeyboardButton(localized_back)] # Загальна кнопка "Назад" (якщо ви
використовуєте її тут)
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_weapon_type'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'weapon_types'

async def show_sword_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [KeyboardButton(f"5★ {localized_texts[user_language]['sword']}")],
        [KeyboardButton(f"4★ {localized_texts[user_language]['sword']}")],
        [KeyboardButton(localized_back_sword)]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_rarity_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'sword_rarity'

async def show_claymore_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [KeyboardButton(f"5★ {localized_texts[user_language]['claymore']}")],
        [KeyboardButton(f"4★ {localized_texts[user_language]['claymore']}")],
        [KeyboardButton(localized_back_claymore)]
    ]
```

```
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_rarity_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'claymore_rarity'

async def show_polearm_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [KeyboardButton(f"5★ {localized_texts[user_language]['polearm']}")],
        [KeyboardButton(f"4★ {localized_texts[user_language]['polearm']}")],
        [KeyboardButton(localized_back_polearm)]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_rarity_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'polearm_rarity'

async def show_bow_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [KeyboardButton(f"5★ {localized_texts[user_language]['bow']}")],
        [KeyboardButton(f"4★ {localized_texts[user_language]['bow']}")],
        [KeyboardButton(localized_back_bow)]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_rarity_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'bow_rarity'

async def show_catalyst_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [KeyboardButton(f"5★ {localized_texts[user_language]['catalyst']}")],
        [KeyboardButton(f"4★ {localized_texts[user_language]['catalyst']}")],
        [KeyboardButton(localized_back_catalyst)]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await
update.message.reply_text(localized_texts[user_language]['choose_rarity_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'catalyst_rarity'
```

```
from telegram import Update, ReplyKeyboardMarkup, KeyboardButton
from telegram.ext import CallbackContext

async def show_sword_5_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    sword_weapons = [weapon for _, weapon in get_weapon_data().items()
                     if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['sword'].lower() and weapon['rarity'] == '5']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in sword_weapons]
    keyboard.append([KeyboardButton(localized_back_sword)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'sword_5_star_list'

async def show_sword_4_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    sword_weapons = [weapon for _, weapon in get_weapon_data().items()
                     if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['sword'].lower() and weapon['rarity'] == '4']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in sword_weapons]
    keyboard.append([KeyboardButton(localized_back_sword)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'sword_4_star_list'

async def show_claymore_5_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    claymore_weapons = [weapon for _, weapon in get_weapon_data().items()
                        if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['claymore'].lower() and weapon['rarity'] == '5']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in claymore_weapons]
    keyboard.append([KeyboardButton(localized_back_claymore)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'claymore_5_star_list'

async def show_claymore_4_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
```

```
    claymore_weapons = [weapon for _, weapon in get_weapon_data().items()
                        if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['claymore'].lower() and weapon['rarity'] == '4']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in claymore_weapons]
    keyboard.append([KeyboardButton(localized_back_claymore)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'claymore_4_star_list'

async def show_polearm_5_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    polearm_weapons = [weapon for _, weapon in get_weapon_data().items()
                       if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['polearm'].lower() and weapon['rarity'] == '5']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in polearm_weapons]
    keyboard.append([KeyboardButton(localized_back_polearm)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'polearm_5_star_list'

async def show_polearm_4_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    polearm_weapons = [weapon for _, weapon in get_weapon_data().items()
                       if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['polearm'].lower() and weapon['rarity'] == '4']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in polearm_weapons]
    keyboard.append([KeyboardButton(localized_back_polearm)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'polearm_4_star_list'

async def show_bow_5_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    bow_weapons = [weapon for _, weapon in get_weapon_data().items()
                   if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['bow'].lower() and weapon['rarity'] == '5']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in bow_weapons]
    keyboard.append([KeyboardButton(localized_back_bow)])
```

```
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'bow_5_star_list'

async def show_bow_4_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    bow_weapons = [weapon for _, weapon in get_weapon_data().items()
                   if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['bow'].lower() and weapon['rarity'] == '4']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in bow_weapons]
    keyboard.append([KeyboardButton(localized_back_bow)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'bow_4_star_list'

async def show_catalyst_5_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    catalyst_weapons = [weapon for _, weapon in get_weapon_data().items()
                        if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['catalyst'].lower() and weapon['rarity'] == '5']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in catalyst_weapons]
    keyboard.append([KeyboardButton(localized_back_catalyst)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'catalyst_5_star_list'

async def show_catalyst_4_star_weapons(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    catalyst_weapons = [weapon for _, weapon in get_weapon_data().items()
                        if weapon['type'].get('uk', '').lower() ==
localized_texts[user_language]['catalyst'].lower() and weapon['rarity'] == '4']
    keyboard = [[KeyboardButton(weapon['name'].get(user_language,
weapon['name']['uk']))] for weapon in catalyst_weapons]
    keyboard.append([KeyboardButton(localized_back_catalyst)])
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await update.message.reply_text(localized_texts[user_language]['choose_weapon'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'catalyst_4_star_list'

async def handle_weapon_selection_from_list(update: Update, context: CallbackContext,
```

```
weapon_type: str, rarity: str, weapon_name: str):
    user_language = context.user_data.get('language', 'uk')
    weapon_data = get_weapon_data()

    selected_weapon_info = None
    for key, data in weapon_data.items():
        if (data['name'].get(user_language, '').lower() == weapon_name.lower() and
            data['type'].get('uk', '').lower() ==
localized_texts[user_language].get(weapon_type, weapon_type).lower() and
            data['rarity'] == rarity):
            selected_weapon_info = data
            break

    if selected_weapon_info:
        caption = f"{localized_texts[user_language]['name']}:
{selected_weapon_info['name'].get(user_language, weapon_name)}\n"
        caption += f"{localized_texts[user_language]['type']}:
{selected_weapon_info['type'].get(user_language, '')}\n"
        caption += f"{localized_texts[user_language]['rarity']}:
{selected_weapon_info['rarity']}★\n"
        await update.message.reply_text(caption)
        # Тут ви можете додати подальші дії, наприклад, показ фото зброї
        # await show_weapon_photo(update, context, weapon_type, weapon_name)
    else:
        await
update.message.reply_text(localized_texts[user_language]['weapon_not_found'])

from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext

async def show_weapon_types_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['sword'],
localized_texts[user_language]['claymore']],
        [localized_texts[user_language]['polearm'],
localized_texts[user_language]['bow']],
        [localized_texts[user_language]['catalyst'],
localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_weapon_type'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'weapon_types'
```

```
from telegram import ReplyKeyboardMarkup

async def show_weapon_rarity_menu(update: Update, context: CallbackContext,
weapon_type: str):
    user_language = context.user_data.get('language', 'uk')
    weapon_type_localized = localized_texts[user_language].get(weapon_type,
weapon_type)
    keyboard = [
        [f"5★ {weapon_type_localized}"],
        [f"4★ {weapon_type_localized}"],
        [f"3★ {weapon_type_localized}"],
        [f"2★ {weapon_type_localized}"],
        [f"1★ {weapon_type_localized}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_weapon_rarity'].format(weapon_type=weapon_
type_localized), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'weapon_rarity'
    context.user_data['selected_weapon_type'] = weapon_type # Переконайтеся, що тип
зброї збережено

WEAPON_PHOTO_FOLDER = "D:\\Games\\дипломнаробота\\weapon"

import os

async def show_weapon_photo(update: Update, context: CallbackContext, weapon_type:
str, weapon_name: str):
    user_language = context.user_data.get('language', 'uk')
    print(f"DEBUG: user_language = {user_language}")  # Додайте цей рядок
    weapon_data = get_weapon_data()

    selected_weapon_info = None
    for key, data in weapon_data.items():
        if (data['name'].get(user_language, '').lower() == weapon_name.lower() and
                data['type'].get('uk', '').lower() ==
localized_texts[user_language].get(weapon_type,

weapon_type).lower()):
            selected_weapon_info = data
            break

    if selected_weapon_info:
```

```
        print(f"DEBUG: selected_weapon_info = {selected_weapon_info}")
        caption = f"{localized_texts[user_language]['name']}:
{selected_weapon_info['name'].get(user_language, weapon_name)}\n"
        caption += f"{localized_texts[user_language]['type']}:
{selected_weapon_info['type'].get(user_language, '')}\n"
        caption += f"{localized_texts[user_language]['rarity']}:
{selected_weapon_info['rarity']}★\n"
        # ... решта коду ...


        folder_name = selected_weapon_info.get('folder_name')
        base_path = "D:\\Games\\дипломнаробота\\weapon\\"
        weapon_folder_path = os.path.join(base_path, folder_name)

        photo_path = None
        if os.path.exists(weapon_folder_path):
            for filename in os.listdir(weapon_folder_path):
                if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
                    photo_path = os.path.join(weapon_folder_path, filename)
                    break

        if update.callback_query:
            await update.callback_query.answer()
            if photo_path:
                try:
                    with open(photo_path, 'rb') as photo:
                        await context.bot.send_photo(chat_id=update.effective_chat.id,
photo=photo, caption=caption)
                except FileNotFoundError:
                    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['photo_not_found'].format(weapon_name=weapon_name)
)
            else:
                await context.bot.send_message(chat_id=update.effective_chat.id,
text=caption)
        elif update.message:
            if photo_path:
                try:
                    with open(photo_path, 'rb') as photo:
                        await context.bot.send_photo(chat_id=update.effective_chat.id,
photo=photo, caption=caption)
                except FileNotFoundError:
                    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['photo_not_found'].format(weapon_name=weapon_name)
)
```

```
            else:
                await context.bot.send_message(chat_id=update.effective_chat.id,
text=caption)
    else:
        await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['weapon_not_found'])
```

Вміст файлу character_handlers.py

```
from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
from localization import localized_texts
from telegram.ext import Updater, CommandHandler, MessageHandler, filters, Application
# Зверніть увагу на мале 'f' у 'filters'


import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext
from localization import localized_texts
from data import character_data


from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
from localization import localized_texts

from telegram import Update, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
from localization import localized_texts

import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext
from localization import localized_texts
from character_data import fixed_character_order


# --- Меню вибору елементів ---
async def show_character_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['anemo']],
        [localized_texts[user_language]['pyro']],
        [localized_texts[user_language]['cryo']],
```

```
        [localized_texts[user_language]['electro']],
        [localized_texts[user_language]['geo']],
        [localized_texts[user_language]['hydro']],
        [localized_texts[user_language]['dendro']],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_element'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'character_elements'

# --- Меню вибору елементів ---
async def show_character_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['anemo'],
localized_texts[user_language]['pyro']],
        [localized_texts[user_language]['cryo'],
localized_texts[user_language]['electro']],
        [localized_texts[user_language]['geo'],
localized_texts[user_language]['hydro']],
        [localized_texts[user_language]['dendro'],
localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_element'], reply_markup=reply_markup)
    context.user_data['current_menu'] = 'character_elements'

async def show_anemo_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_anemo = localized_texts[user_language]['anemo']
    keyboard = [
        [f"5★ {localized_anemo}"],
        [f"4★ {localized_anemo}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_rarity'].format(element=localized_anemo),
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'anemo_rarity'

async def show_anemo_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
```

```
    localized_anemo = localized_texts[user_language]['anemo']
    anemo_4_star = fixed_character_order.get(user_language, {}).get(localized_anemo,
{}).get('4', [])
    if anemo_4_star:
        keyboard = []
        for i in range(0, len(anemo_4_star), 2):
            row = [anemo_4_star[i]]
            if i + 1 < len(anemo_4_star):
                row.append(anemo_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} "])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['anemo_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_anemo)
        keyboard = [[f"{localized_texts[user_language]['back']} "]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'anemo_4_star_list'

async def show_anemo_5_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_anemo = localized_texts[user_language]['anemo']
    anemo_5_star = fixed_character_order.get(user_language, {}).get(localized_anemo,
{}).get('5', [])
    if anemo_5_star:
        keyboard = []
        for i in range(0, len(anemo_5_star), 2):
            row = [anemo_5_star[i]]
            if i + 1 < len(anemo_5_star):
                row.append(anemo_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} "])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['anemo_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_anemo)
        keyboard = [[f"{localized_texts[user_language]['back']} "]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
```

```
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'anemo_5_star_list'

# --- Меню вибору рідкості для Піро ---
async def show_pyro_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [f"5★ {localized_texts[user_language]['pyro']}"],
        [f"4★ {localized_texts[user_language]['pyro']}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_rarity'].format(element=localized_texts[us
er_language]['pyro']), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'pyro_rarity'

async def show_pyro_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_pyro = localized_texts[user_language]['pyro']
    pyro_4_star = fixed_character_order.get(user_language, {}).get(localized_pyro,
{}).get('4', [])
    if pyro_4_star:
        keyboard = []
        for i in range(0, len(pyro_4_star), 2):
            row = [pyro_4_star[i]]
            if i + 1 < len(pyro_4_star):
                row.append(pyro_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🔥"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['pyro_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_pyro)
        keyboard = [[f"{localized_texts[user_language]['back']} 🔥"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'pyro_4_star_list'

async def show_pyro_5_star_characters(update: Update, context: CallbackContext):
```

```
    user_language = context.user_data.get('language', 'uk')
    localized_pyro = localized_texts[user_language]['pyro']
    pyro_5_star = fixed_character_order.get(user_language, {}).get(localized_pyro,
{}).get('5', [])
    if pyro_5_star:
        keyboard = []
        for i in range(0, len(pyro_5_star), 2):
            row = [pyro_5_star[i]]
            if i + 1 < len(pyro_5_star):
                row.append(pyro_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🔥"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['pyro_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_pyro)
        keyboard = [[f"{localized_texts[user_language]['back']} 🔥"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'pyro_5_star_list'

# --- Меню вибору рідкості для Кріо ---
async def show_cryo_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [f"5★ {localized_texts[user_language]['cryo']}"],
        [f"4★ {localized_texts[user_language]['cryo']}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_rarity'].format(element=localized_texts[us
er_language]['cryo']), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'cryo_rarity'

async def show_cryo_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_cryo = localized_texts[user_language]['cryo']
    cryo_4_star = fixed_character_order.get(user_language, {}).get(localized_cryo,
{}).get('4', [])
    if cryo_4_star:
```

```
        keyboard = []
        for i in range(0, len(cryo_4_star), 2):
            row = [cryo_4_star[i]]
            if i + 1 < len(cryo_4_star):
                row.append(cryo_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🧊"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['cryo_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_cryo)
        keyboard = [[f"{localized_texts[user_language]['back']} 🧊"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'cryo_4_star_list'


async def show_cryo_5_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_cryo = localized_texts[user_language]['cryo']
    cryo_5_star = fixed_character_order.get(user_language, {}).get(localized_cryo,
{}).get('5', [])
    if cryo_5_star:
        keyboard = []
        for i in range(0, len(cryo_5_star), 2):
            row = [cryo_5_star[i]]
            if i + 1 < len(cryo_5_star):
                row.append(cryo_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} ❄️"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['cryo_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_cryo)
        keyboard = [[f"{localized_texts[user_language]['back']} ❄️"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'cryo_5_star_list'
```

```
# --- Меню вибору рідкості для Електро ---
async def show_electro_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [f"5★ {localized_texts[user_language]['electro']}"],
        [f"4★ {localized_texts[user_language]['electro']}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_rarity'].format(element=localized_texts[us
er_language]['electro']), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'electro_rarity'

async def show_electro_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_electro = localized_texts[user_language]['electro']
    electro_4_star = fixed_character_order.get(user_language,
{}).get(localized_electro, {}).get('4', [])
    if electro_4_star:
        keyboard = []
        for i in range(0, len(electro_4_star), 2):
            row = [electro_4_star[i]]
            if i + 1 < len(electro_4_star):
                row.append(electro_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} "])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['electro_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_electro)
        keyboard = [[f"{localized_texts[user_language]['back']} "]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'electro_4_star_list'

async def show_electro_5_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_electro = localized_texts[user_language]['electro']
    electro_5_star = fixed_character_order.get(user_language,
```

```
{}).get(localized_electro, {}).get('5', [])
    if electro_5_star:
        keyboard = []
        for i in range(0, len(electro_5_star), 2):
            row = [electro_5_star[i]]
            if i + 1 < len(electro_5_star):
                row.append(electro_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} "])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['electro_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_electro)
        keyboard = [[f"{localized_texts[user_language]['back']} "]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'electro_5_star_list'

# --- Меню вибору рідкості для Гео ---
async def show_geo_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [f"5★ {localized_texts[user_language]['geo']}"],
        [f"4★ {localized_texts[user_language]['geo']}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['choose_rarity'].format(element=localized_texts[us
er_language]['geo']), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'geo_rarity'

async def show_geo_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_geo = localized_texts[user_language]['geo']
    geo_4_star = fixed_character_order.get(user_language, {}).get(localized_geo,
{}).get('4', [])
    if geo_4_star:
        keyboard = []
        for i in range(0, len(geo_4_star), 2):
            row = [geo_4_star[i]]
```

```
            if i + 1 < len(geo_4_star):
                row.append(geo_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🔙"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['geo_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_geo)
        keyboard = [[f"{localized_texts[user_language]['back']} 🔙"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'geo_4_star_list'

async def show_geo_5_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_geo = localized_texts[user_language]['geo']
    geo_5_star = fixed_character_order.get(user_language, {}).get(localized_geo,
{}).get('5', [])
    if geo_5_star:
        keyboard = []
        for i in range(0, len(geo_5_star), 2):
            row = [geo_5_star[i]]
            if i + 1 < len(geo_5_star):
                row.append(geo_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🔙"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['geo_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_geo)
        keyboard = [[f"{localized_texts[user_language]['back']} 🔙"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'geo_5_star_list'


# --- Меню вибору рідкості для Гідро ---
async def show_hydro_rarity_menu(update: Update, context: CallbackContext):
```

```
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [f"5★ {localized_texts[user_language]['hydro']}"],
        [f"4★ {localized_texts[user_language]['hydro']}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['choose_rarity'].format(

element=localized_texts[user_language]['hydro']), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'hydro_rarity'


async def show_hydro_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_hydro = localized_texts[user_language]['hydro']
    hydro_4_star = fixed_character_order.get(user_language, {}).get(localized_hydro,
{}).get('4', [])
    if hydro_4_star:
        keyboard = []
        for i in range(0, len(hydro_4_star), 2):
            row = [hydro_4_star[i]]
            if i + 1 < len(hydro_4_star):
                row.append(hydro_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 💧"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['hydro_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_hydro)
        keyboard = [[f"{localized_texts[user_language]['back']} 💧"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'hydro_4_star_list'


async def show_hydro_5_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_hydro = localized_texts[user_language]['hydro']
```

```
    hydro_5_star = fixed_character_order.get(user_language, {}).get(localized_hydro,
{}).get('5', [])
    if hydro_5_star:
        keyboard = []
        for i in range(0, len(hydro_5_star), 2):
            row = [hydro_5_star[i]]
            if i + 1 < len(hydro_5_star):
                row.append(hydro_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 💧"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['hydro_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_hydro)
        keyboard = [[f"{localized_texts[user_language]['back']} 💧"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'hydro_5_star_list'


# --- Меню вибору рідкості для Дендро ---
async def show_dendro_rarity_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [f"5★ {localized_texts[user_language]['dendro']}"],
        [f"4★ {localized_texts[user_language]['dendro']}"],
        [localized_texts[user_language]['back']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,

text=localized_texts[user_language]['choose_rarity'].format(

element=localized_texts[user_language]['dendro']), reply_markup=reply_markup)
    context.user_data['current_menu'] = 'dendro_rarity'


async def show_dendro_4_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_dendro = localized_texts[user_language]['dendro']
    dendro_4_star = fixed_character_order.get(user_language, {}).get(localized_dendro,
```

```
{}).get('4', [])
    if dendro_4_star:
        keyboard = []
        for i in range(0, len(dendro_4_star), 2):
            row = [dendro_4_star[i]]
            if i + 1 < len(dendro_4_star):
                row.append(dendro_4_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🔙"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['dendro_4_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='4★',
element=localized_dendro)
        keyboard = [[f"{localized_texts[user_language]['back']} 🔙"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'dendro_4_star_list'


async def show_dendro_5_star_characters(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    localized_dendro = localized_texts[user_language]['dendro']
    dendro_5_star = fixed_character_order.get(user_language, {}).get(localized_dendro,
{}).get('5', [])
    if dendro_5_star:
        keyboard = []
        for i in range(0, len(dendro_5_star), 2):
            row = [dendro_5_star[i]]
            if i + 1 < len(dendro_5_star):
                row.append(dendro_5_star[i + 1])
            keyboard.append(row)
        keyboard.append([f"{localized_texts[user_language]['back']} 🔙"])
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
        text = f"{localized_texts[user_language]['dendro_5_star_characters']}:"
    else:
        text =
localized_texts[user_language]['no_characters_found'].format(rarity='5★',
element=localized_dendro)
        keyboard = [[f"{localized_texts[user_language]['back']} 🔙"]]
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id, text=text,
```

```
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'dendro_5_star_list'



CHARACTERS_PHOTO_FOLDER = "D:\\Games\\дипломнаробота\\builds"  # Вам потрібно
визначити цю константу, якщо її немає

# async def show_selected_characters_from_list(update: Update, context:
CallbackContext, selected_key: str):
#     chat_id = update.effective_chat.id
#     selected_characters = context.user_data.get(selected_key, set())
#     user_language = context.user_data.get('language', 'uk')
#
#     if not selected_characters:
#         await
update.message.reply_text(localized_texts[user_language]['no_characters_selected'])
#         return
#
#     media_group = []
#     for name in selected_characters:
#         if name in character_data and 'photo_folder' in character_data[name]:
#             folder_path = character_data[name]['photo_folder']
#             full_path = os.path.join(CHARACTERS_PHOTO_FOLDER, folder_path)  #
Використовуйте CHARACTERS_PHOTO_FOLDER
#             try:
#                 all_files = os.listdir(full_path)
#                 image_files = [f for f in all_files if f.lower().endswith(('.jpg',
'.jpeg', '.png', '.gif'))]
#
#                 if image_files:
#                     image_path = os.path.join(full_path, image_files[0])
#                     with open(image_path, 'rb') as photo_file:
#                         media_group.append(InputMediaPhoto(photo_file.read(),
caption=name if not media_group else None))
#                 else:
#                     await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photos_found'].format(character=name))
#
#             except FileNotFoundError:
#                 await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['folder_not_found'].format(character=name,
path=full_path))
#             except Exception as e:
```

```
#                 print(f"Помилка при відправці фото {name}: {e}")
#                 await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_sending_character_photo'].format(character=
name))
#         else:
#             await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['no_photo_info'].format(character=name))
#
#     if media_group:
#         if len(media_group) == 1:
#             await context.bot.send_photo(chat_id=chat_id,
photo=media_group[0].media, caption=media_group[0].caption)
#         else:
#             await context.bot.send_media_group(chat_id=chat_id, media=media_group)

import os
from telegram import Update, InputMediaPhoto
from telegram.ext import CallbackContext
from localization import localized_texts
from data import character_data

CHARACTERS_PHOTO_FOLDER = "D:\\Games\\дипломнаробота\\builds"  # Переконайтеся, що ця
константа визначена

async def show_character_info(update: Update, context: CallbackContext,
requested_name: str):
    chat_id = update.effective_chat.id
    user_language = context.user_data.get('language', 'uk')
    found_character_name = None
    character_info = None

    # Пошук персонажа за українським або англійським ім'ям
    for internal_id, char_data in character_data.items():
        if requested_name.lower() == char_data['name']['uk'].lower() or
requested_name.lower() == char_data['name']['en'].lower():
            found_character_name = internal_id  # Використовуємо внутрішній ID для
подальшого пошуку фото
            character_info = char_data
            break

    if found_character_name and 'photo_folder' in character_info:
        folder_path = character_info['photo_folder']
        full_path = os.path.join(CHARACTERS_PHOTO_FOLDER, folder_path)
        try:
            all_files = os.listdir(full_path)
```

```
            image_files = [f for f in all_files if f.lower().endswith(('.jpg',
'.jpeg', '.png', '.gif', '.webp'))]

            if image_files:
                media_group = []
                for image_file in image_files:
                    image_path = os.path.join(full_path, image_file)
                    print(f"Додано до media_group: {image_path}")
                    with open(image_path, 'rb') as photo_file:
                        media_group.append(
                            InputMediaPhoto(photo_file.read(),

caption=f"{character_info['name'][user_language]}\n" \

f"{character_info['element'][user_language]},
{character_info['weapon'][user_language]}\n" \
                                                    f"{'★' *
character_info['rarity']}\n" \
                                                    "-~-~-~-~-~-~-~-~-~-\n"

f"{character_info['description'][user_language]}"
                            if not media_group else None)) # Використовуємо українське
ім'я для підпису

                if media_group:
                    print(f"Надсилання {len(media_group)} фотографій групами:")
                    for i in range(0, len(media_group), 10):
                        chunk = media_group[i:i + 10]
                        print(f"Надсилання групи з {len(chunk)} фотографій")
                        if len(chunk) == 1:
                            await context.bot.send_photo(chat_id=chat_id,
photo=chunk[0].media,
                                                         caption=chunk[0].caption)
                        else:
                            await context.bot.send_media_group(chat_id=chat_id,
media=chunk)
                return  # Важливо вийти з функції після надсилання фото

            else:
                await
update.message.reply_text(localized_texts[user_language]['no_photos_found'].format(cha
racter=character_info['name']['uk']))
                return

        except FileNotFoundError:
```

```
            await
update.message.reply_text(localized_texts[user_language]['folder_not_found'].format(ch
aracter=character_info['name']['uk'], path=full_path))
            return
        except Exception as e:
            print(f"Помилка при відправці фото {character_info['name']['uk']}: {e}")
            await
update.message.reply_text(localized_texts[user_language]['error_sending_character_phot
o'].format(character=character_info['name']['uk']))
            return
    else:
        await
update.message.reply_text(localized_texts[user_language]['character_info_not_found'].f
ormat(character=requested_name))


async def handle_character_selection_from_list(update: Update, context:
CallbackContext, element: str, rarity: str):
    user_language = context.user_data.get('language', 'uk')
    localized_element = context.user_data.get('selected_element_localized')
    selected_rarity = context.user_data.get('selected_rarity') # Використовуйте
правильний ключ для рідкості

    print(f"localized_element перед отриманням списку: '{localized_element}'")
    print(f"rarity перед отриманням списку: '{selected_rarity}'")

    characters = fixed_character_order.get(user_language,
{}).get(localized_texts[user_language].get(element.lower()),
                                                                {}).get(rarity, [])
    user_message = update.message.text.strip() # Додаємо .strip()

    print(f"Список персонажів для порівняння: {characters}")
    print(f"Отримане повідомлення для порівняння: '{user_message}'")

    if user_message in characters:
        print(f"Викликається show_character_info з ім'ям: {user_message}")
        await show_character_info(update, context, user_message)
    elif user_message.lower() == localized_texts[user_language].get('back',
"").lower():
        if element == 'anemo':
            await show_anemo_rarity_menu(update, context)
        elif element == 'pyro':
            await show_pyro_rarity_menu(update, context)
        elif element == 'cryo':
            await show_cryo_rarity_menu(update, context)
```

```
        elif element == 'electro':
            await show_electro_rarity_menu(update, context)
        elif element == 'geo':
            await show_geo_rarity_menu(update, context)
        elif element == 'hydro':
            await show_hydro_rarity_menu(update, context)
        elif element == 'dendro':
            await show_dendro_rarity_menu(update, context)
    else:
        await update.message.reply_text(f"Персонаж '{user_message}' не знайдений у
списку.")
```

Вміст файлу localization.py

```
localized_texts = {
    'uk': {
        'greeting': 'Привіт! Чим цікавишся?',
        'characters': 'Персонажі',
        'weapons': 'Зброя',
        'artifacts': 'Артефакти',
        'main_menu': 'Головне меню',
        'change_language_button': 'Змінити мову',
        'choose_language': 'Виберіть мову:',
        'language_changed': 'Мову змінено на українську.',
        'weapon_types': 'Виберіть тип зброї:',
        'sword': 'Одноручна зброя',
        'polearm': 'Древкова зброя',
        'bow': 'Луки',
        'claymore': 'Дворучна зброя',
        'catalyst': 'Каталізатори',
        'choose_weapon_rarity': 'Виберіть рідкість зброї:',
        'choose_weapon': 'Виберіть зброю:',
        'weapon_list_title': 'Зброя типу "{weapon_type}" ({rarity}★):',
        'no_weapons_found': 'Зброї типу "{weapon_type}" рідкості {rarity}★ не
знайдено.',
        'weapon_photo_not_found': 'Фото для зброї "{weapon}" не знайдено.',
        'today': 'Що сьогодні?',
        'today_info': 'Що вас цікавить на сьогодні?',
        'back_to_main_menu': 'Назад',
        'your_stats': 'Ваша статистика',
        'locations_routes': 'Локації та маршрути',
        'spiral_abyss': 'Безодня',
        'the_chasm': 'Театр',
        'elemental_reactions': 'Елементальні реакції',
```

```
        'banners': 'Які зараз банери?',
        'events': 'Події',
        'change_language': 'Змінити мову',
        'curiosities': 'Цікавинки',
        'curiosities_title': 'Що вас цікавить?',
        'birthday_subscribe': 'Підписатися на дні народження',
        'birthday_unsubscribe': 'Відписатися від днів народження',
        'unofficial': 'Неофіційне',
        'pictures': 'Картинки',
        'birthday': 'День народження', # Залишаємо для сумісності
        'birthday_subscribed': 'Ви підписалися на розсилку про дні народження
персонажів.',
        'birthday_already_subscribed': 'Ви вже підписані на розсилку про дні
народження.',
        'birthday_unsubscribed': 'Ви відписалися від розсилки про дні народження
персонажів.',
        'birthday_not_subscribed': 'Ви не підписані на розсилку про дні народження.',
        'unofficial_info': 'Тут буде неофіційна інформація.',
        'pictures_info': 'Тут будуть картинки.',
        'character_material_calculator': 'Калькулятор матеріалів персонажів',
        'weapon_material_calculator': 'Калькулятор матеріалів зброї',
        'birthday_photos': 'Фото з днями народження',
        'not_found_birthday_photos_folder': 'Папка "{folder}" з фотографіями не
знайдена.',
        'choose_birthday_photo': 'Виберіть персонажа, чиє фото дня народження ви
хочете побачити:',
        'not_found_photo': 'Фото для {character} не знайдено.',
        'back_to_curiosities': 'Назад до цікавинок',
        'combat': "Бойове",
        'combat_title': "⚔️ Що вас цікавить у бойовому плані?",
        'enemies': "👺 Вороги",
        'abyss': "🌀 Безодня",
        'theater': "🎭 Театр",
        'reactions': "🔥 Елементальні реакції",
        'birthday_subscribe_button': "🔔 Підписатися на оновлення",
        'birthday_calendar_button': "🗓️Календар днів народження",
        'promocodes': "🎁Промокоди",
        'birthday_calendar_info': "Перегляньте календар днів народження персонажів.",
        'subscribe_birthday_calendar': "🔔 Підписатися на календар",
        'unsubscribe_birthday_calendar': "🔕 Відписатися від календаря",
        'promocodes_info': "Тут буде інформація про актуальні промокоди.",
        'subscribe_promocodes': "🔔 Підписатися на промокоди",
        'unsubscribe_promocodes': "🔕 Відписатися від промокодів",
```

```
        'subscribe_promocodes_offer': "Натисніть кнопку нижче, щоб отримувати
сповіщення про нові промокоди.",
        'already_subscribed_calendar': "ℹ️ Ви вже підписані на оновлення календаря
днів народження.",
        'already_subscribed_promocodes': "ℹ️ Ви вже підписані на оновлення
промокодів.",
        'subscribed_success_calendar': "✅ Ви успішно підписалися на оновлення
календаря днів народження!",
        'subscribed_success_promocodes': "✅ Ви успішно підписалися на оновлення
промокодів!",
        'unsubscribed_success_calendar': "❌ Ви успішно відписалися від оновлень
календаря днів народження.",
        'unsubscribed_success_promocodes': "❌ Ви успішно відписалися від оновлень
промокодів.",

        'birthday_calendar_subscribed_info': "Ви підписані на оновлення календаря днів
народження.",
        'birthday_calendar_unsubscribed_info': "Ви не підписані на оновлення календаря
днів народження.",
        'subscribed_calendar': "✅ Ви підписалися на оновлення календаря.",
        'unsubscribed_calendar': "❌ Ви відписалися від оновлень календаря.",
        'subscribed_promocodes': "✅ Ви підписалися на оновлення промокодів.",
        'unsubscribed_promocodes': "❌ Ви відписалися від оновлень промокодів.",

        'help_text': """Я бот-помічник по грі Genshin Impact. Ось основні розділи:

○ *Сьогодні*: Інформація про сьогоднішні матеріали для прокачки, банери.
○ *Персонажі*: Перегляд інформації про персонажів за елементом та рідкістю.
○ *Зброя*: Перегляд інформації про зброю за типом та рідкістю.
○ *Артефакти*: Розділ у розробці.
○ *Цікавинки*: Календар днів народжень, промокоди, неофіційна інформація, зображення,
зміна мови.
○ *Бій*: Інформація про ворогів, Безодню, реакції (у розробці).""",
        'rarity': 'Рідкість', # ДОДАНО УКРАЇНСЬКИЙ ПЕРЕКЛАД
        'choose_character_category': 'Виберіть категорію персонажа:', # ДОДАНО

        "today_talents": "Сьогоднішні таланти",
        "today_weapons": "Сьогоднішня зброя",

        "today_talents_caption": "Таланти, доступні сьогодні:",
        "today_weapons_caption": "Зброя, матеріали для якої доступні сьогодні:",
        "photo_not_found": "Фото файлу '{filename}' не знайдено.",
```

```
        "error_sending_photo": "Виникла помилка при відправці фото: {error}",
        "current_banners": "Які зараз банери?",
        "character_banners_caption": "Поточні банери персонажів:",
        "weapon_banners_caption": "Поточні банери зброї:",
        "no_banners_found": "Зображення банерів не знайдено.",
        "back_to_menu": "Назад до меню",

        'back_button': "Назад",
        'back_to_curiosities_text': "Назад",
        'unofficial_menu_caption': "Неофіційна інформація:",
        'characters_button': "Майбутні персонажі",
        'future_banners_button': "Майбутні банери",
        'locations_button': "Майбутні локації",
        'curiosities_button': "Майбутні цікавинки",
        'start_command': "Ласкаво просимо!",
        'language_choice': "Будь ласка, оберіть мову:",
        'language_ukrainian': "Українська",
        'language_english': "Англійська",

        'main_menu_text': "Оберіть розділ:",
        'main_menu_artifacts': "Артефакти",
        'main_menu_characters': "Персонажі",
        'artifact_choose_characteristic': "Оберіть характеристику артефактів:",

        'artifact_sent': "Відправляю артефакти...",
        'artifact_not_found': "Набір артефактів не знайдено.",
        'main_menu_returned': "Повернуто до головного меню.",
        'main_menu_button': "Назад",
        'FUTURE_CAPTION' : "Інформація може змінитися.",
        'artifact_attack_button': "Сили атаки.",
        'artifact_attack_caption': "Артефакти з бонусом до сили атаки.",
        'artifact_talent_bonus_button': "Бонус талантів.",
        'artifact_talent_bonus_caption': "Артефакти з бонусом до талантів.",
        'artifact_elemental_bonus_button': "Елементальна шкода.",
        'artifact_elemental_bonus_caption': "Артефакти з бонусом до елементальної
шкоди.",
        'artifact_elemental_mastery_button': "Майстерність стихій.",
        'artifact_elemental_mastery_caption': "Артефакти з бонусом до майстерності
стихій.",
        'artifact_healing_button': "Лікування.",
        'artifact_healing_caption': "Артефакти з бонусом до лікування.",
        'artifact_defensive_button' : "HP та Захист.",
        'artifact_defensive_caption': "Артефакти з бонусом до HP та Захисту.",
        'artifact_other_button': "Інші бонуси.",
        'artifact_other_caption': "Артефакти з іншими бонусами.",
```

```
'artifact_from_category': "Артефакти з категорії: {category}",
        'anemo': "Анемо",
        'electro': "Електро",
        'geo': "Гео",
        'cryo': "Кріо",
        'pyro': "Піро",
        'hydro': "Гідро",
        'dendro': "Дендро",
        'back': "Назад",

        'select_character': "Оберіть елемент персонажа:",


        'choose_element': 'Оберіть елемент персонажа:',
        'choose_rarity': 'Оберіть рідкість персонажа:',  # Загальний, якщо десь
використовується
        'choose_rarity_anemo': 'Оберіть рідкість Анемо персонажа:',
        'choose_rarity_electro': 'Оберіть рідкість Електро персонажа:',
        'choose_rarity_geo': 'Оберіть рідкість Гео персонажа:',
        'choose_rarity_cryo': 'Оберіть рідкість Кріо персонажа:',
        'choose_rarity_pyro': 'Оберіть рідкість Піро персонажа:',
        'choose_rarity_hydro': 'Оберіть рідкість Гідро персонажа:',
        'choose_rarity_dendro': 'Оберіть рідкість Дендро персонажа:',
        'choose_rarity_weapon': "Оберіть рідкість зброї:",

        'four_star': '⭐⭐⭐⭐',
        'five_star': '⭐⭐⭐⭐⭐',
        'not_found_character': "Персонажів '{character_name}' не знайдено.",
        'choose_character': 'Оберіть персонажа:',
        'error_selection': 'Помилка вибору. Будь ласка, спробуйте ще раз.',
        'error_character_info': 'Інформацію про цього персонажа не знайдено.',
        'anemo_4_star_characters': 'Список 4★ Анемо персонажів',
        'anemo_5_star_characters': 'Список 5★ Анемо персонажів',
        'pyro_4_star_characters': 'Список 4★ Піро персонажів',
        'pyro_5_star_characters': 'Список 5★ Піро персонажів',
        'cryo_4_star_characters': 'Список 4★ Кріо персонажів',
        'cryo_5_star_characters': 'Список 5★ Кріо персонажів',
        'electro_4_star_characters': 'Список 4★ Електро персонажів',
        'electro_5_star_characters': 'Список 5★ Електро персонажів',
        'geo_4_star_characters': 'Список 4★ Гео персонажів',
        'geo_5_star_characters': 'Список 5★ Гео персонажів',
        'hydro_4_star_characters': 'Список 4★ Гідро персонажів',
```

```
        'hydro_5_star_characters': 'Список 5★ Гідро персонажів',
        'dendro_4_star_characters': 'Список 4★ Дендро персонажів',
        'dendro_5_star_characters': 'Список 5★ Дендро персонажів',

'show_selected_characters_button': "Показати обраних",
        'no_characters_found': 'Персонажів {rarity} {element} не знайдено.',  #
Додайте цей ключ
        'character_info_not_found': 'Інформація про персонажа {character} не
знайдена.',
        'no_photo_info': 'Інформація про фото для {character} відсутня.',
        'no_photos_found': 'Фотографій для {character} не знайдено.',
        'folder_not_found': 'Папку з фотографіями для {character} не знайдено за
шляхом: {path}',
        'error_sending_character_photo': 'Виникла помилка при відправці фотографії
{character}.',
        'no_characters_selected': 'Ви не обрали жодного персонажа.',
        'abyss_caption': 'Інформація про Безодню.',
        'theater_caption': 'Інформація про Театр.',
        'reactions_caption': 'Інформація про елементальні реакції.',
        'enemies_caption': 'Інформація про ворогів.',

        'enemies_menu_title': 'Оберіть тип ворогів:',
        'bosses': 'Боси',
        'common_enemies': 'Звичайні',
        'elite_enemies': 'Елітні',
        'back_to_combat_menu': 'Назад до бойового меню',
        'back_to_enemies_menu': 'Назад до меню ворогів',
        'select_enemy': 'Оберіть ворога:',
        'no_enemies_found': 'Ворогів типу "{enemy_type}" не знайдено.',
        'no_photos_for_enemy': 'Для ворога "{enemy}" немає фотографій.',
        'no_photos_in_folder': 'У папці "{folder}" немає фотографій.',

        'error_reading_photo': 'Помилка при читанні файлу: {filename}',
        'error_sending_photos': 'Помилка при відправці фотографій з розділу
"{category}".',
        'error_processing_folder': 'Помилка при обробці папки "{folder}".',

        'choose_weapon_type': 'Виберіть тип зброї:',


        'name': 'Назва',# Якщо ви використовуєте це поле у підписі до фото зброї,
        'weapon_not_found': 'Вибачте, інформацію про цю зброю не знайдено.',
        'type': 'Тип',

    },
```

```
    'en': {
        'greeting': 'Hello! What are you interested in?',
        'characters': 'Characters',
        'weapons': 'Weapons',
        'artifacts': 'Artifacts',
        'main_menu': 'Main Menu',
        'change_language_button': 'Change Language',
        'choose_language': 'Choose language:',
        'language_changed': 'Language changed to English.',
        'weapon_types': 'Choose a weapon type:',
        'sword': 'Sword',
        'polearm': 'Polearm',
        'bow': 'Bow',
        'claymore': 'Claymore',
        'catalyst': 'Catalyst',
        'choose_weapon_rarity': 'Choose weapon rarity:',
        'choose_weapon': 'Choose a weapon:',
        'today': 'What\'s today?',
        'today_info': 'What are you interested in today?',
        'back_to_main_menu': 'Back',
        'your_stats': 'Your stats',
        'locations_routes': 'Locations & routes',
        'spiral_abyss': 'Spiral Abyss',
        'the_chasm': 'Theater',
        'elemental_reactions': 'Elemental reactions',
        'banners': 'What are the banners now?',
        'events': 'Events',
        'change_language': 'Change Language',
        'curiosities': 'Curiosities',
        'curiosities_title': 'What are you interested in?',
        'birthday_subscribe': 'Subscribe to birthdays',
        'birthday_unsubscribe': 'Unsubscribe from birthdays',
        'unofficial': 'Unofficial',
        'pictures': 'Pictures',
        'birthday': 'Birthday', # Залишаємо для сумісності
        'birthday_subscribed': 'You have subscribed to character birthday
notifications.',
```

```
        'birthday_already_subscribed': 'You are already subscribed to birthday
notifications.',
        'birthday_unsubscribed': 'You have unsubscribed from character birthday
notifications.',
        'birthday_not_subscribed': 'You are not subscribed to birthday
notifications.',
        'unofficial_info': 'Unofficial information will be here.',
        'pictures_info': 'Pictures will be here.',
        'character_material_calculator': 'Character material calculator',
        'weapon_material_calculator': 'Weapon material calculator',
        'birthday_photos': 'Birthday Photos',
        'not_found_birthday_photos_folder': 'The "{folder}" folder with photos was not
found.',
        'choose_birthday_photo': 'Choose the character whose birthday photo you want
to see:',
        'not_found_photo': 'Photo for {character} not found.',
        'back_to_curiosities': 'Back to curiosities',
        'combat': "⚔️ Combat",
        'combat_title': "⚔️ What are you interested in regarding combat?",
        'enemies': "👹 Enemies",
        'abyss': "🌀 Abyss",
        'theater': "🎭 Theater",
        'reactions': "🔥 Elemental Reactions",
        'birthday_subscribe_button': "🔔 Subscribe to updates",
        'birthday_calendar_button': "🗓️Birthday Calendar",
        'promocodes': "🎁Promocodes",
        'birthday_calendar_info': "View the character birthday calendar.",
        'subscribe_birthday_calendar': "🔔 Subscribe to calendar",
        'unsubscribe_birthday_calendar': "🔕 Unsubscribe from calendar",
        'promocodes_info': "Information about current promocodes will be here.",
        'subscribe_promocodes': "🔔 Subscribe to promocodes",
        'unsubscribe_promocodes': "🔕 Unsubscribe from promocodes",
        'subscribe_promocodes_offer': "Click the button below to receive notifications
about new promocodes.",
        'already_subscribed_calendar': "ℹ️ You are already subscribed to birthday
calendar updates.",
        'already_subscribed_promocodes': "ℹ️ You are already subscribed to promocode
updates.",
        'subscribed_success_calendar': "✅ You have successfully subscribed to
birthday calendar updates!",
        'subscribed_success_promocodes': "✅ You have successfully subscribed to
promocode updates!",
```

```
        'unsubscribed_success_calendar': "❌ You have successfully unsubscribed from
birthday calendar updates.",
        'unsubscribed_success_promocodes': "❌ You have successfully unsubscribed from
promocode updates.",

        'birthday_calendar_subscribed_info': "You are subscribed to birthday calendar
updates.",
        'birthday_calendar_unsubscribed_info': "You are not subscribed to birthday
calendar updates.",
        'subscribed_calendar': "✅ You have subscribed to calendar updates.",
        'unsubscribed_calendar': "❌ You have unsubscribed from calendar updates.",
        'subscribed_promocodes': "✅ You have subscribed to promocode updates.",
        'unsubscribed_promocodes': "❌ You have unsubscribed from promocode updates.",

        'help_text': """I'm a helper bot for Genshin Impact. Here are the main
sections:

○ **Today**: Information about today's upgrade materials, banners.
○ **Characters**: View information about characters by element and rarity.
○ **Weapons**: View information about weapons by type and rarity.
○ **Artifacts**: Section under development.
○ **Curiosities**: Birthday calendar, promocodes, unofficial information, pictures,
change language.
○ **Combat**: Information about enemies, Abyss, reactions (under development).""",
        'rarity': 'Rarity', # ДОДАНО АНГЛІЙСЬКИЙ ПЕРЕКЛАД
        'choose_character_category': 'Choose a character category:', # ДОДАНО
                'select_character': 'Select a character:',
        "today_talents": "Today's Talents",
        "today_weapons": "Today's Weapons",

        "today_talents_caption": "Talents available today:",
        "today_weapons_caption": "Weapon materials available today:",
        "photo_not_found": "Photo file '{filename}' not found.",
        "error_sending_photo": "An error occurred while sending the photo: {error}",
        "current_banners": "What are the banners now?",

        "back_to_menu": "Back to Menu",
        'back_button': "Back",
        'back_to_curiosities_text': "Back",
        'unofficial_menu_caption': "Unofficial Information:",
        'characters_button': "Future Characters",
        'future_banners_button': "Future Banners",
        'locations_button': "Future Locations",
```

```
        'curiosities_button': "Future Curiosities",
        'start_command': "Welcome!",
        'language_choice': "Please choose your language:",
        'language_ukrainian': "Ukrainian",
        'language_english': "English",

        'main_menu_text': "Choose a section:",
        'main_menu_artifacts': "Artifacts",
        'main_menu_characters': "Characters",
        'artifact_choose_characteristic': "Choose an artifact characteristic or enter
a set name:",

        'artifact_sent': "Sending artifacts...",
        'artifact_not_found': "Artifact set not found.",
        'main_menu_returned': "Returned to the main menu.",

        'artifact_attack': "Attack",
        'artifact_talent_bonus': "Talent Bonus",
        'artifact_elemental_bonus': "Elemental DMG Bonus",
        'elemental_mastery': "Elemental Mastery +80",
        'main_menu_button': "Back",
        'character_banners_caption': "Current Character Banners:",
        'weapon_banners_caption': "Current Weapon Banners:",
        'no_banners_found': "No banner images found.",
        'FUTURE_CAPTION' : 'Subject To Change',
        'artifact_attack_button': "Attack Power.",
        'artifact_attack_caption': "Artifacts with an attack power bonus.",
        'artifact_talent_bonus_button': "Talent Bonus.",
        'artifact_talent_bonus_caption': "Artifacts with a talent bonus.",
        'artifact_elemental_bonus_button': "Elemental Damage.",
        'artifact_elemental_bonus_caption': "Artifacts with an elemental damage
bonus.",
        'artifact_elemental_mastery_button': "Elemental Mastery.",
        'artifact_elemental_mastery_caption': "Artifacts with an elemental mastery
bonus.",
        'artifact_healing_button': "Healing Bonus.",
        'artifact_healing_caption': "Artifacts with a healing bonus.",
        'artifact_defensive_button': "HP and Defense.",
        'artifact_defensive_caption': "Artifacts with an HP and Defense bonus.",
        'artifact_other_button': "Other Bonuses.",
        'artifact_other_caption': "Artifacts with other bonuses.",

        'anemo': "Anemo",
        'electro': "Electro",
        'geo': "Geo",
```

```
'cryo': "Cryo",
'pyro': "Pyro",
'hydro': "Hydro",
'dendro': "Dendro",
'back': "Back",

# 'select_character': "Choose a character's element:",
'choose_rarity': "Choose the character's rarity:",
'choose_rarity_weapon': "Choose the weapons\'s rarity:",


'choose_element': 'Choose a character\'s element:',

'choose_rarity_anemo': 'Choose the rarity of Anemo character:',
'choose_rarity_electro': 'Choose the rarity of Electro character:',
'choose_rarity_geo': 'Choose the rarity of Geo character:',
'choose_rarity_cryo': 'Choose the rarity of Cryo character:',
'choose_rarity_pyro': 'Choose the rarity of Pyro character:',
'choose_rarity_hydro': 'Choose the rarity of Hydro character:',
'choose_rarity_dendro': 'Choose the rarity of Dendro character:',
'four_star': '⭐⭐⭐⭐',
'five_star': '⭐⭐⭐⭐⭐',
'not_found_character': "Characters '{character_name}' not found.",
'choose_character': 'Choose a character:',
'error_selection': 'Selection error. Please try again.',
'error_character_info': 'Character information not found.',
'anemo_4_star_characters': 'List of 4★ Anemo characters',
'anemo_5_star_characters': 'List of 5★ Anemo characters',
'pyro_4_star_characters': 'List of 4★ Pyro characters',
'pyro_5_star_characters': 'List of 5★ Pyro characters',
'cryo_4_star_characters': 'List of 4★ Cryo characters',
'cryo_5_star_characters': 'List of 5★ Cryo characters',
'electro_4_star_characters': 'List of 4★ Electro characters',
'electro_5_star_characters': 'List of 5★ Electro characters',
'geo_4_star_characters': 'List of 4★ Geo characters',
'geo_5_star_characters': 'List of 5★ Geo characters',
'hydro_4_star_characters': 'List of 4★ Hydro characters',
'hydro_5_star_characters': 'List of 5★ Hydro characters',
'dendro_4_star_characters': 'List of 4★ Dendro characters',
'dendro_5_star_characters': 'List of 5★ Dendro characters',

'show_selected_characters_button': 'Show selected',
```

```
        'no_characters_found': 'No {rarity} {element} characters found.',  # Додайте
цей ключ
        'character_info_not_found': 'Information about {character} not found.',
        'no_photo_info': 'No photo information for {character}.',
        'no_photos_found': 'No photos found for {character}.',
        'folder_not_found': 'Photo folder for {character} not found at path: {path}',
        'error_sending_character_photo': 'An error occurred while sending the photo of
{character}.',
        'no_characters_selected': 'You have not selected any characters.',
        'abyss_caption': 'Information about the Abyss.',
        'theater_caption': 'Information about the Theater.',
        'reactions_caption': 'Information about elemental reactions.',
        'enemies_caption': 'Information about enemies.',

        'enemies_menu_title': 'Choose enemy type:',
        'bosses': 'Bosses',
        'common_enemies': 'Common',
        'elite_enemies': 'Elite',
        'back_to_combat_menu': 'Back to combat menu',
        'back_to_enemies_menu': 'Back to enemies menu',
        'select_enemy': 'Select an enemy:',
        'no_enemies_found': 'No enemies of type "{enemy_type}" found.',
        'no_photos_for_enemy': 'No photos found for the enemy "{enemy}".',
        'no_photos_in_folder': 'No photos found in the "{folder}" folder.',

        'error_reading_photo': 'Error reading file: {filename}',
        'error_sending_photos': 'Error sending photos from the "{category}" section.',
        'error_processing_folder': 'Error processing the "{folder}" folder.',
        'weapon_list_title': '"{weapon_type}" weapons ({rarity}★):',
        'no_weapons_found': 'No "{weapon_type}" weapons of {rarity}★ rarity found.',
        'weapon_photo_not_found': 'Photo for "{weapon}" not found.',
        # 'error_sending_photo': 'Error sending photo of "{item}".',
        'choose_weapon_type': 'Choose a weapon type:',
        'weapon_not_found': 'Sorry, information about this weapon was not found.',
        'type': 'Type',
    }
}
```

Вміст файлу artifacts_handler.py

```
import os
from telegram import Update, InputMediaPhoto
```

```
from telegram.ext import CallbackContext
from localization import localized_texts
from telegram import Update, InputMediaPhoto, ReplyKeyboardMarkup
from telegram.ext import CallbackContext
import os

ARTIFACTS_FOLDER = "D:\\Games\\дипломнаробота\\artifacts"  # Шлях до вашої папки з
артефактами

async def send_artifact_category_media(update: Update, context: CallbackContext,
category: str, caption_key: str):
    chat_id = update.message.chat.id
    media_path = os.path.join(ARTIFACTS_FOLDER, category)
    user_language = context.user_data.get('language', 'uk')

    try:
        all_files = os.listdir(media_path)
        image_files = [f for f in all_files if f.lower().endswith(('.jpg', '.jpeg',
'.png', '.gif'))]

        if not image_files:
            await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['artifact_not_found_category'].format(category=loc
alized_texts[user_language].get(f'artifact_{category.lower().replace(" ", "_")}',
category)))
            return

        media_group = []
        for image_file in image_files:
            image_path = os.path.join(media_path, image_file)
            with open(image_path, 'rb') as photo_file:
                # Отримуємо локалізований підпис за допомогою переданого ключа
                caption_text = localized_texts[user_language].get(caption_key, "")
                media_group.append(InputMediaPhoto(photo_file.read(),
caption=caption_text if not media_group else None))

        if media_group:
            if len(media_group) == 1:
                await context.bot.send_photo(chat_id=chat_id,
photo=media_group[0].media, caption=media_group[0].caption)
            else:
                await context.bot.send_media_group(chat_id=chat_id, media=media_group)

    except FileNotFoundError:
        await context.bot.send_message(chat_id=chat_id,
```

```
text=localized_texts[user_language]['folder_not_found'].format(folder=media_path))
    except Exception as e:
        print(f"Помилка при відправці артефактів з категорії {category}: {e}")
        await context.bot.send_message(chat_id=chat_id,
text=localized_texts[user_language]['error_sending_artifacts_category'].format(categor
y=localized_texts[user_language].get(f'artifact_{category.lower().replace(" ", "_")}',
category)))

async def show_artifacts_menu(update: Update, context: CallbackContext):
    user_language = context.user_data.get('language', 'uk')
    keyboard = [
        [localized_texts[user_language]['artifact_attack_button'],
localized_texts[user_language]['artifact_talent_bonus_button']],
        [localized_texts[user_language]['artifact_elemental_bonus_button'],
localized_texts[user_language]['artifact_healing_button']],
        [localized_texts[user_language]['artifact_elemental_mastery_button'],
localized_texts[user_language]['artifact_defensive_button']],
        [localized_texts[user_language]['artifact_other_button'],
localized_texts[user_language]['main_menu_button']]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True)
    await context.bot.send_message(chat_id=update.effective_chat.id,
text=localized_texts[user_language]['artifact_choose_characteristic'],
reply_markup=reply_markup)
    context.user_data['current_menu'] = 'artifacts_menu'
```