

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА
Навчально-науковий інститут енергетичної, інформаційної
та транспортної інфраструктури

Кафедра комп'ютерних наук та інформаційних технологій

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Проектування telegram-бота для автоматизованого формування
квестів різних типів»

Виконала: здобувачка вищої освіти 4 курсу,
групи КН 2022-1
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

Олександра ВЕНЖЕГА 
(ім'я та прізвище)

Керівник: Наталія БРАТЕРСЬКА 
(ім'я та прізвище)

Рецензент ас. Валерія ЧЕРКАСОВА 
(ім'я та прізвище)

м. Харків – 2026 рік

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)

Навчально-науковий Інститут енергетичної, інформаційної
та транспортної інфраструктури

Кафедра комп'ютерних наук та інформаційних технологій

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 «Комп'ютерні науки»

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КНтаІТ



Марина НОВОЖИЛОВА

«23» червня 2026 Року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Венжеги Олександри Євгенівни

(прізвище, ім'я, по батькові)

1. Тема роботи Проектування Telegram-бота для автоматизованого
формування квестів різних типів

керівник роботи ас. Братерська Наталія Миколаївна

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «22» травня 2026 р. № 440-03

2. Термін подання студентом роботи 22.06.2026р.

3. Вихідні дані до роботи Проектування та реалізація Telegram-бота для
автоматизованого формування квестів різних типів з використанням Python,
Aiogram, PostgreSQL, SQLAlchemy та генеративних моделей штучного інтелекту
(g4f)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметного середовища та аналіз існуючих аналогів; проектування
системи(діаграми варіантів використання, компонентів, класів, ER-діаграма бази
даних, алгоритми); розробка програмного забезпечення Telegram-бота
(реєстрація користувачів, створення та генерація квестів засобами штучного
інтелекту, механізм проходження квестів, робота з датасетами, система
збереження статистики); тестування системи та розробка керівництва
користувача; аналіз питань охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Наталія БРАТЕРСЬКА 	11.05.2026	15.05.2026
Розділ II	Наталія БРАТЕРСЬКА 	15.05.2026	26.05.2026
Розділ III	Наталія БРАТЕРСЬКА 	26.05.2026	01.06.2026
Розділ IV	Вікторія МАЛИШЕВА 	02.06.2026	05.06.2026

7. Дата видачі завдання 11.05.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми кваліфікаційної роботи	11.05.2026	Виконано
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки дипломної роботи	13.05.2026	Виконано
3	Написання I розділу	15.05.2026	Виконано
4	Написання II розділу	26.05.2026	Виконано
5	Написання III розділу	01.06.2026	Виконано
6	Написання IV розділу	06.06.2026	Виконано
7	Подання дипломної роботи керівнику	07.06.2026	Виконано
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	14.06.2026	Виконано
9	Подання доопрацьованого варіанту роботи керівнику	15.06.2026	Виконано
10	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	24.06.2026	Виконано

Здобувач


(підпис)

Олександра ВЕНЖЕГА
(прізвище та ініціали)

Керівник роботи


(підпис)

Наталія БРАТЕРСЬКА
(прізвище та ініціали)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи бакалавра студента групи КН 2022-1 спеціальності 122 Комп'ютерні науки Венжеги Олександри Євгенівни за темою «Проектування Telegram-бота для автоматизованого формування квестів різних типів» складається з 4 розділів, містить 30 рисунків, 2 таблиць, 18 джерел.

Кваліфікаційну роботу бакалавра присвячено проектуванню та реалізації Telegram-бота, який автоматизує створення, проходження та обмін квестами двох типів – вікторин і фізичних активностей.

У розділі «Загальні положення» наведено опис предметного середовища, проаналізовано існуючі аналоги, визначено мету та завдання дослідження.

У розділі «Проектування та інформаційне забезпечення системи» розроблено загальну концепцію системи, побудовано діаграми варіантів використання, компонентів і класів, ER-діаграму бази даних, а також алгоритми.

У розділі «Програмне та технічне забезпечення» наведено опис засобів розробки (Python, Aiogram, PostgreSQL, SQLAlchemy, g4f), вимоги до технічного та програмного забезпечення, детальний опис програмної реалізації ключових механізмів, результати тестування та керівництво користувача.

У розділі «Охорона праці» розглянуто законодавче регулювання питань охорони праці, виявлено потенційні небезпеки для ІТ-фахівців та запропоновано заходи для їх попередження.

Ключові слова: TELEGRAM-БОТ, ГЕНЕРАЦІЯ КВЕСТІВ, ВІКТОРИНА, ФІЗИЧНИЙ КВЕСТ, DATASET, AIGRAM, POSTGRESQL, SQLALCHEMY, ШТУЧНИЙ ІНТЕЛЕКТ.

ANNOTATION

Structure and scope of the thesis. The explanatory note of the bachelor's qualification thesis prepared by a student of group KN 2022-1, specialty 122 Computer Science, Venzheha Oleksandra Yevhenivna entitled "Design of a Telegram Bot for Automated Generation of Various Types of Quests" consists of 4 chapters, contains 30 figures, 2 tables, and 18 references.

The bachelor's qualification thesis is devoted to the design and implementation of a Telegram bot that automates the creation, completion, and sharing of two types of quests: quizzes and physical activity quests.

The chapter "General Provisions" presents a description of the subject domain, analyzes existing analogues, and defines the purpose and objectives of the study.

The chapter "System Design and Information Support" describes the development of the overall system concept, the construction of use case, component, and class diagrams, the development of the database ER diagram, as well as the design of the system algorithms.

The chapter "Software and Technical Support" provides a description of the development tools used (Python, Aiogram, PostgreSQL, SQLAlchemy, g4f), specifies the hardware and software requirements, presents a detailed description of the implementation of the key system mechanisms, reports the testing results, and includes a user guide.

The chapter "Occupational Health and Safety" examines the legislative regulation of occupational safety issues, identifies potential hazards for IT professionals, and proposes preventive measures to minimize these risks.

Keywords: TELEGRAM BOT, QUEST GENERATION, QUIZ, PHYSICAL QUEST, DATASET, AIOGRAM, POSTGRESQL, SQLALCHEMY, ARTIFICIAL INTELLIGENCE.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ.....	10
1.1 Опис предметного середовища.....	10
1.2 Огляд наявних аналогів	11
1.2.1 Аналіз платформи TeachQuest.....	11
1.2.2 Аналіз платформи Habitica	13
1.3 Постановка задачі	16
Висновки до розділу	17
РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ	19
2.1 Аналіз предметної області	19
2.1.1 Загальна концепція системи	20
2.2 Проєктування системи.....	22
2.2.1 Діаграма варіантів використання системи	22
2.2.2 Діаграма компонентів системи.....	24
2.2.3 Діаграма класів системи.....	26
2.2.4 Проєктування бази даних.....	28
2.2.5 ER-діаграма бази даних.....	34
2.3 Алгоритмічне забезпечення	38
2.3.1 Алгоритм створення користувацької вікторини.....	38
2.3.2 Алгоритм генерації вікторини засобами штучного інтелекту	40
2.3.3 Алгоритм проходження квесту.....	42
2.3.4 Алгоритм генерації фізичного квесту засобами штучного інтелекту	44
2.3.5 Алгоритм публікації квесту в датасет	46
2.3.6 Алгоритм пошуку квестів за кодом датасету.....	48
2.4 Проєктування взаємодії компонентів системи	50
2.4.1 Діаграма послідовності створення користувацького квесту	50
2.4.2 Діаграма послідовності генерації квесту засобами штучного інтелекту	51
2.4.3 Діаграма послідовності проходження квесту	53

2.4.4 Діаграма послідовності роботи з датасетами.....	54
2.5 Опис процесу діяльності користувача	56
Висновки до розділу	58
РОЗДІЛ 3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ.....	60
3.1 Засоби розробки системи	60
3.2 Вимоги до технічного та програмного забезпечення.....	61
3.3 Опис програмної реалізації.....	62
3.3.1 Реалізація механізму реєстрації користувачів	64
3.3.2 Реалізація функціоналу створення квестів.....	65
3.3.3 Реалізація генерації квестів засобами штучного інтелекту.....	68
3.3.4 Реалізація механізму проходження квестів.....	70
3.3.5 Реалізація механізму роботи з датасетами	73
3.3.6 Реалізація системи збереження статистики користувачів	77
3.4 Тестування системи, перевірка коректності роботи та аналіз результатів	80
3.5 Керівництво користувача	85
Висновки до розділу	87
РОЗДІЛ 4 ОХОРОНА ПРАЦІ	89
4.1 Організаційно-правові основи забезпечення безпеки праці.....	89
4.2 Характеристика об'єкта та виявлення потенційних небезпек	91
4.3 Дослідження ризику реалізації потенційних небезпек на об'єкті проєктування та розробка заходів щодо їх попередження	93
Висновки до розділу	98
ВИСНОВКИ.....	99
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	100

ВСТУП

Сучасні цифрові технології відкривають нові можливості для створення інтерактивного контенту, який поєднує навчальні та розважальні елементи. Одним із таких форматів є квести – послідовності завдань, що стимулюють логічне мислення, перевіряють знання або залучають до фізичної активності. Однак більшість існуючих платформ для створення квестів вимагають ручного наповнення контентом або не підтримують автоматичну генерацію нових завдань, що обмежує їхню варіативність.

Актуальність проєкту полягає в потребі розробки інструменту, який зменшує участь людини у створенні квестів завдяки використанню генеративних моделей, водночас надаючи можливість самостійного конструювання та обміну готовими наборами. Використання Telegram як платформи забезпечує доступність для широкої аудиторії без потреби встановлювати додаткові застосунки. Такий підхід дозволяє поєднати гейміфікацію з автоматизацією, що підвищує залученість користувачів.

Метою роботи є створення Telegram-бота для автоматизованого формування, проходження та обміну квестами двох типів – вікторин і фізичних активностей. Система повинна забезпечувати генерацію унікального контенту на основі параметрів, заданих користувачем, а також підтримувати публікацію власних квестів через механізм датасетів.

Для досягнення поставленої мети визначено такі завдання дослідження:

- аналіз предметної області та існуючих аналогів (TeachQuest, Habitica);
- проєктування архітектури системи, структури бази даних та ключових алгоритмів;
- вибір і обґрунтування технологічного стеку (Python, Aiogram, PostgreSQL, SQLAlchemy);
- реалізація функціоналу реєстрації, створення, генерації, проходження квестів, роботи з датасетами та статистикою;

– тестування працездатності та оцінювання зручності інтерфейсу.

Об’єкт дослідження – процес створення, проходження та поширення квестів у месенджері Telegram.

Предмет дослідження – методи, алгоритми та програмні засоби для автоматизованого формування інтерактивних квестів із використанням генеративних моделей.

У роботі застосовано такі методи: аналіз наукових джерел та існуючих платформ, об’єктно-орієнтоване проєктування, моделювання бази даних, асинхронне програмування, інтеграція з API генеративних моделей, тестування програмних модулів.

Результатом роботи є функціонуючий Telegram-бот, який дозволяє користувачам генерувати вікторини та фізичні квести за заданими параметрами, створювати власні набори завдань, публікувати їх з унікальним кодом доступу, проходити чужі квести та відстежувати власну статистику. Система забезпечує варіативність контенту завдяки стохастичній природі генерації.

Отримані результати мають практичну значущість для розробників подібних систем, викладачів, які можуть використовувати бот для створення тестових матеріалів, а також для користувачів, зацікавлених в інтерактивному навчанні чи підтримці фізичної активності в ігровій формі.

РОЗДІЛ 1

ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Предметне середовище проєкту охоплює процеси створення, проходження та обміну інтерактивними квестами в месенджері Telegram. Під квестами в цій роботі розуміються послідовності запитань або завдань, які користувач виконує покроково, отримуючи бали за правильні відповіді або виконані вправи. Такі активності можуть використовуватися як для розважальних цілей, так і для перевірки знань у навчальному контексті.

Платформа Telegram обрана через кілька причин. По-перше, вона надає зручний API для створення ботів із підтримкою клавіатур, кнопок і діалогових станів. По-друге, месенджер має велику аудиторію, що полегшує потенційне впровадження розробки. По-третє, для користувача взаємодія з ботом не потребує встановлення додаткових застосунків – достатньо наявного облікового запису Telegram.

У системі виділяються два типи квестів. Вікторини складаються з питань із чотирма варіантами відповідей, серед яких лише один правильний. Фізичні квести являють собою набір текстових описів вправ (наприклад, «10 присідань», «30 секунд планки»), виконання яких не потребує вибору варіанту, а лише підтвердження від користувача. Такий поділ дозволяє охопити різні сценарії використання: від перевірки знань до залучення до рухової активності.

Користувач у межах системи може виступати в трьох ролях: гравець (проходить готові квести), автор (створює власні квести через конструктор) та публікатор (викладає свої квести в загальний доступ через механізм датасетів). Датасет – це набір квестів, об'єднаних унікальним кодом, яким автор може поділитися з іншими.

Додатково система підтримує автоматичну генерацію квестів на основі зовнішнього API мовної моделі. Користувач задає параметри: тему (для вікторини), кількість запитань або вправ, рівень складності (для фізичних квестів). Бот формує промпт, надсилає запит, обробляє відповідь і пропонує отриманий квест до проходження. Усі згенеровані квести є унікальними завдяки стохастичній природі моделі.

Важливою складовою предметного середовища є збереження результатів діяльності. Система фіксує кожну спробу проходження квесту: кількість набраних балів, час початку та завершення. На основі цих даних формуються агреговані показники статистики (рівень, кількість пройдених квестів, відсоток правильних відповідей), які користувач може переглянути в будь-який момент.

Узагальнена схема взаємодії компонентів системи та основні функціональні блоки будуть детально розглянуті в наступних підрозділах.

1.2 Огляд наявних аналогів

1.2.1 Аналіз платформи TeachQuest

TeachQuest – це вебплатформа, яка перетворює навчальний процес на рольову гру (RPG). Вона призначена для вчителів та учнів: педагоги створюють квести, а учні виконують завдання, отримуючи ігрові бонуси. Основний акцент зроблено на гейміфікації класної кімнати, а не на автоматичній генерації контенту чи обміні квестами між користувачами [1].

На головній сторінці TeachQuest, рисунок 1.1, відображається загальна статистика використання: кількість вчителів, класів, учнів та країн, де застосовується платформа. Інтерфейс виконаний у темних тонах із елементами RPG-стилізації.

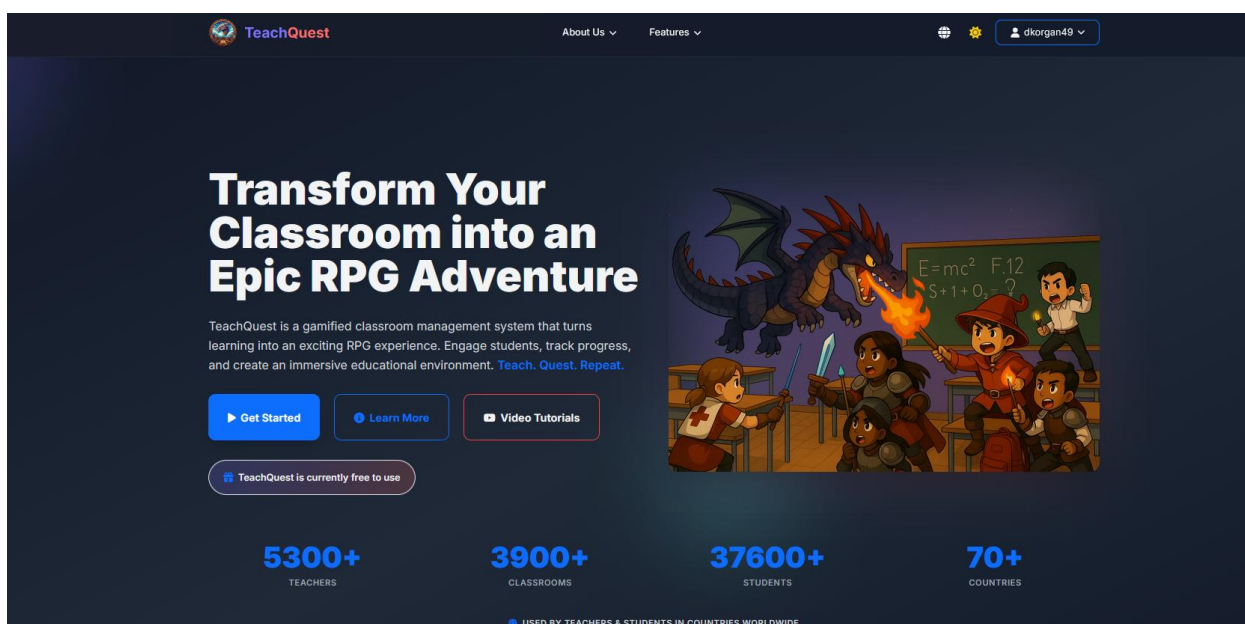


Рисунок 1.1 – Головна сторінка TeachQuest

Основні функціональні можливості TeachQuest включають:

- створення ігрових персонажів для учнів із рівнями, досвідом, характеристиками (HP, MP, XP);
- управління класами та групами учнів;
- нарахування нагород (GP – gold points) за виконані завдання;
- відстеження прогресу окремого учня через індивідуальний дашборд;
- можливість редагування статистики персонажа вручну.

На рисунку 1.2 показано сторінку профілю учня. Тут відображаються ім'я, рівень, клас персонажа наприклад: «Defender», поточні показники здоров'я, мана, досвіду та золотих монет. Також доступні кнопки для нагородження, зміни характеристик, перегляду активних ефектів та керування дашбордом.

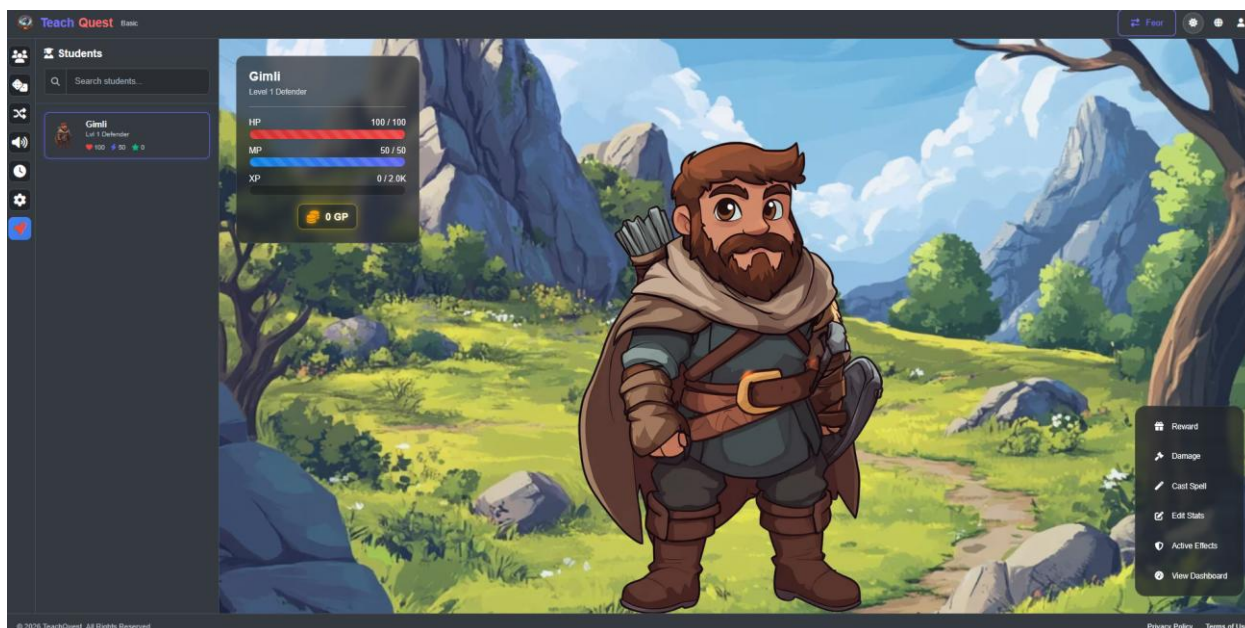


Рисунок 1.2 – Сторінка профілю учня в TeachQuest

TeachQuest пропонує готову RPG-обгортку для шкільних занять, але не дозволяє автоматично генерувати нові квести за допомогою штучного інтелекту. Весь контент має створювати вчитель самостійно. Також відсутній механізм обміну квестами між різними викладачами через унікальні коди або deer-link. Система орієнтована на синхронну роботу в класі, а не на індивідуальне проходження в будь-який час.

Порівняно з TeachQuest, розроблюваний Telegram-бот пропонує автоматичну генерацію квестів (вікторини та фізичні вправи) за темою або складністю, а також конструктор для створення власних квестів. Крім того, бот підтримує публікацію квестів у датасети з унікальним кодом, що дозволяє поширювати контент між користувачами без централізованого адміністрування. На відміну від TeachQuest, система не вимагає реєстрації окремого облікового запису – використовується Telegram ID.

1.2.2 Аналіз платформи Habbitica

Habbitica – це вебплатформа та мобільний застосунок, який перетворює управління завданнями, звичками та щоденними справами на рольову гру. Користувач створює персонажа, який отримує досвід, золото та рівні за виконання реальних справ, і втрачає здоров'я за невиконані завдання. Habbitica

орієнтована на особисту продуктивність, а не на освітні квести чи обмін контентом [2].

На рисунку 1.3 показано екран вибору категорій активностей: робота, спортивні вправи, здоров'я, школа. Користувач позначає сфери, над якими планує працювати, що дозволяє системі пропонувати відповідні завдання.

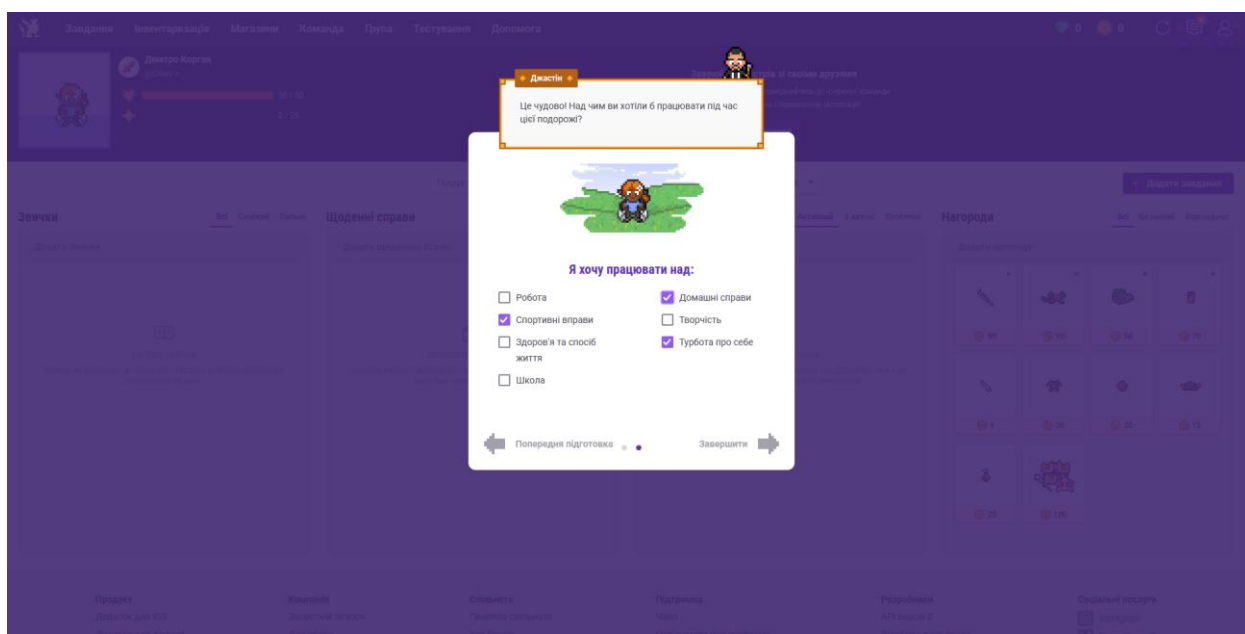


Рисунок 1.3 – Вибір категорій активностей у Habitica

Інтерфейс Habitica, рисунок 1.4, містить список завдань, які поділяються на звички, можна виконувати з плюсом або мінусом, тобто формувати позитивні або шкідливі звички, щоденні завдання та звичайні завдання. Користувач може додавати власні пункти, наприклад, «10 хвилин прибирання», «10 хв кардіо», а також позначати їх як звичку або щоденну активність. Система нараховує ігрові бали та досвід за виконані дії.

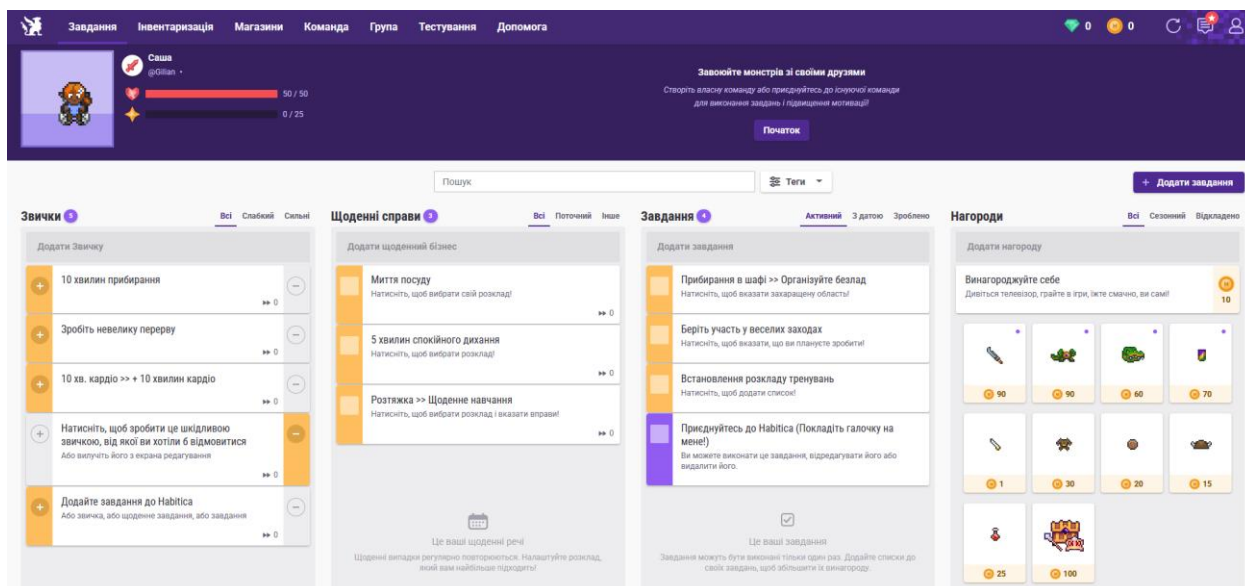


Рисунок 1.4 – Список завдань у Habitica

Основні функціональні можливості Habitica включають:

- створення персонажа з класом, рівнем, досвідом, здоров'ям, золотом;
- додавання звичок (із позитивним або негативним підкріпленням), щоденних завдань та разових завдань;
- налаштування складності завдання (впливає на нагороду);
- виконання завдань через чекбокси або кнопки «плюс/мінус»;
- отримання винагород (спорядження, навички) за внутрішньоігрову валюту;
- участь у групах та спільних викликах;
- перегляд статистики прогресу та рівнів.

Habitica пропонує гейміфікований трекінг завдань, але не дозволяє автоматично генерувати нові квести чи вікторини за темою. Весь контент створюється самим користувачем вручну. Також відсутній механізм обміну готовими наборами завдань через унікальні коди, хоча є спільні виклики в групах. Habitica не підтримує вікторини з варіантами відповідей, лише виконання дій «так/ні» або кількісне відстеження.

Порівняно з Habitica, розроблюваний Telegram-бот орієнтований саме на квести: послідовності запитань, вікторини або фізичних вправ. Бот генерує контент автоматично через ШІ, що знімає з користувача необхідність

самостійно придумувати завдання. Крім того, бот підтримує публікацію квестів у датасети з кодом доступу, чого немає в Habitica. Проте Habitica пропонує розвинену систему персонажів, груп та довгострокового розвитку, що виходить за межі поточного проекту.

1.3 Постановка задачі

Мета проекту полягає в розробці Telegram-бота, який автоматизує створення, виконання та поширення квестів двох типів: вікторин та фізичних активностей. Система має надати користувачеві можливість як самостійно конструювати завдання, так і отримувати їх у готовому вигляді за допомогою генерації на основі зовнішнього API.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- реалізувати автоматичне створення облікового запису в системі за ідентифікатором Telegram під час першого звернення;
- побудувати інтерфейс для конструювання вікторин із довільною кількістю питань, чотирма варіантами відповідей, указанням правильної відповіді та кількості балів;
- організувати створення фізичних квестів як набору текстових описів вправ із балами за виконання;
- забезпечити автоматичну генерацію вікторин за темою та кількістю питань через звернення до мовної моделі;
- забезпечити генерацію фізичних квестів із можливістю вибору рівня складності;
- реалізувати покрокове проходження квесту з накопиченням балів, перевіркою відповідей (для вікторин) або фіксацією виконання (для фізичних вправ);
- передбачити дострокове завершення квесту без збереження результату;

- організувати збереження статистики кожного користувача: кількість пройдених квестів, правильних і неправильних відповідей, середній бал, поточний рівень;
- розробити механізм публікації власних квестів у загальний доступ з унікальним кодом та пошуку чужих квестів за цим кодом;
- створити навігацію через текстові та інлайн-клавіатури для зручного переміщення між розділами.

Технічна реалізація передбачає використання мови Python, бібліотеки Aiogram для асинхронної обробки повідомлень [6], PostgreSQL як сховища даних, SQLAlchemy ORM для доступу до бази. Генерація контенту виконується через запити до OpenAI API. Такий стек забезпечує стабільну роботу застосунку та його здатність одночасно обслуговувати багатьох користувачів.

Кінцевим результатом стане функціонуючий Telegram-бот, який дозволяє користувачеві отримувати нові квести в автоматичному режимі, створювати власні, ділитися ними з іншими та відстежувати прогрес. Генерація щоразу дає різний набір завдань, що підтримує інтерес до використання. Публікація квестів через датасети створює передумови для масштабування контенту силами самих користувачів.

Висновки до розділу

У першому розділі проведено аналіз предметного середовища, визначено основні сутності та процеси, що стосуються створення, проходження та обміну квестами в месенджері Telegram. Розглянуто два типи квестів – вікторини та фізичні активності, а також ролі користувача: гравець, автор, публікатор. Описано механізм автоматичної генерації контенту через зовнішнє API та необхідність збереження статистики для відстеження прогресу.

Виконано огляд двох існуючих систем. TeachQuest – це вебплатформа для гейміфікації навчання в класі, яка використовує RPG-елементи, але не підтримує автоматичну генерацію квестів або обмін ними між користувачами через коди. Habbitica перетворює управління завданнями та звичками на гру, однак її контент повністю створюється вручну, відсутні вікторини з варіантами відповідей і механізми публікації наборів завдань. Обидва аналоги не забезпечують інтеграції з Telegram та не пропонують автоматизованої генерації.

На основі виявлених недоліків існуючих систем сформульовано мету та завдання проєкту. Основними з них є реалізація реєстрації через Telegram ID, створення власних квестів через конструктор, генерація вікторин і фізичних активностей за допомогою ШІ, збереження статистики, а також розробка механізму датасетів для обміну квестами з унікальними кодами доступу. Технічний стек обрано з урахуванням вимог до асинхронності, надійного зберігання даних та простоти розгортання.

Таким чином, перший розділ закладає теоретичне підґрунтя для подальшого проєктування та реалізації системи.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Аналіз предметної області

Предметною областю є автоматизоване формування, проходження та обмін інтерактивними квестами через месенджер Telegram. Квест у загальному розумінні – це послідовність завдань, об'єднаних спільною темою або сценарієм. Виконання завдань супроводжується нарахуванням балів, відстеженням прогресу та, як правило, елементами змагальності. У контексті роботи квести поділяються на два основні типи: вікторини та фізичні квести.

Вікторина – це набір питань із варіантами відповідей, де лише один варіант є правильним. Користувач отримує бали за кожну правильну відповідь, а після завершення бачить підсумковий результат. Такий формат підходить для перевірки знань, навчання або розваги. Фізичний квест складається з описів вправ: наприклад, «10 присідань», «30 секунд планки», які користувач має виконати в реальному житті. Бали нараховуються за фактом виконання, а правильність не перевіряється, лише самостійна відмітка.

Окремим напрямом є генерація контенту за допомогою штучного інтелекту. Великі мовні моделі здатні створювати унікальні питання на задану тему, формулювати варіанти відповідей і визначати правильний, а також генерувати описи фізичних вправ із урахуванням рівня складності. Це дозволяє оновлювати базу завдань без участі людини та забезпечувати варіативність при кожному зверненні.

Важливою складовою є можливість обміну квестами між користувачами. Для цього використовується поняття датасету – набору квестів, об'єднаних унікальним кодом. Автор публікує свій квест, отримує код, а інші користувачі можуть ввести цей код і отримати доступ до проходження. Такий механізм не вимагає централізованого каталогу і дозволяє поширювати контент через будь-які канали зв'язку.

Гейміфікація підвищує залученість: у системі передбачено рівні користувача, які підвищуються після певної кількості пройдених квестів, а також загальну статистику (правильні відповіді, середній бал). Це мотивує повертатися до бота та змагатися із самим собою або іншими [3], [4], [5].

Аналіз предметної області показує, що існує потреба в інструменті, який поєднує генерацію, проходження та обмін квестами в одному середовищі. Розроблений бот закриває цю потребу, пропонуючи два типи квестів, інтелектуальну генерацію, систему датасетів та накопичення статистики.

2.1.1 Загальна концепція системи

Основна ідея полягає в поєднанні кількох функціональних можливостей в одному середовищі. Користувач отримує доступ до генерації квестів за обраною тематикою, може проходити їх у покроковому режимі, а також створювати власні набори завдань. Це дозволяє використовувати систему як для навчальних цілей, так і для розважального контенту.

Архітектурно система реалізована за клієнт-серверним підходом. Користувач взаємодіє з Telegram-ботом через мобільний або десктопний клієнт, який виступає інтерфейсом доступу. Основна логіка обробки запитів зосереджена на серверній частині, де відбувається формування квестів, перевірка відповідей та управління даними користувача. Збереження інформації реалізоване за допомогою бази даних, що забезпечує фіксацію результатів проходження, історії активності та створених квестів. На рисунку 2.1 представлено загальну архітектуру системи та взаємодію її основних компонентів.

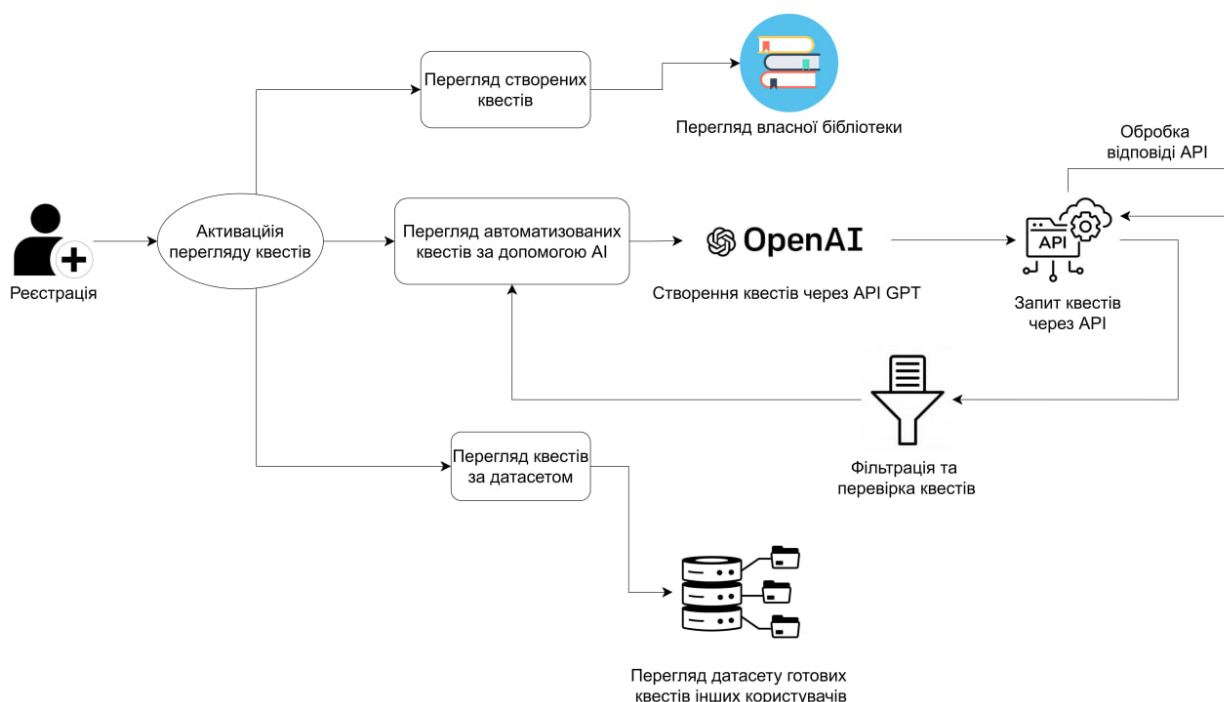


Рисунок 2.1 – Загальна архітектура системи

Особливістю системи є використання інтелектуальних механізмів генерації контенту. Це дозволяє формувати унікальні квести з різною структурою, рівнем складності та тематикою. При кожному запуску користувач отримує новий набір завдань, що підвищує варіативність та повторну зацікавленість у використанні системи.

Крім того, передбачено можливість обміну квестами між користувачами. Для цього квести об'єднуються в набори: датасети, яким присвоюється унікальний код. Такий підхід дозволяє поширювати контент через посилання або спеціальні коди, розширює функціональність системи та сприяє її масштабуванню.

Загальна концепція системи орієнтована на створення універсального інструменту, який поєднує генерацію, проходження та управління квестами в єдиному середовищі з простим і зрозумілим інтерфейсом.

2.2 Проєктування системи

2.2.1 Діаграма варіантів використання системи

Діаграма варіантів використання відображає, які функції надає система різним типам зовнішніх сутностей – акторам. Telegram-бот для автоматизованого формування квестів виділено два актори [12]: «Користувач» людина, яка взаємодіє з ботом через інтерфейс Telegram та «AI-модуль (GPT)» зовнішній сервіс генерації контенту на основі мовної моделі. Сам бот представлений як система, у межах якої визначено 12 варіантів використання. Діаграму наведено на рисунку 2.2.

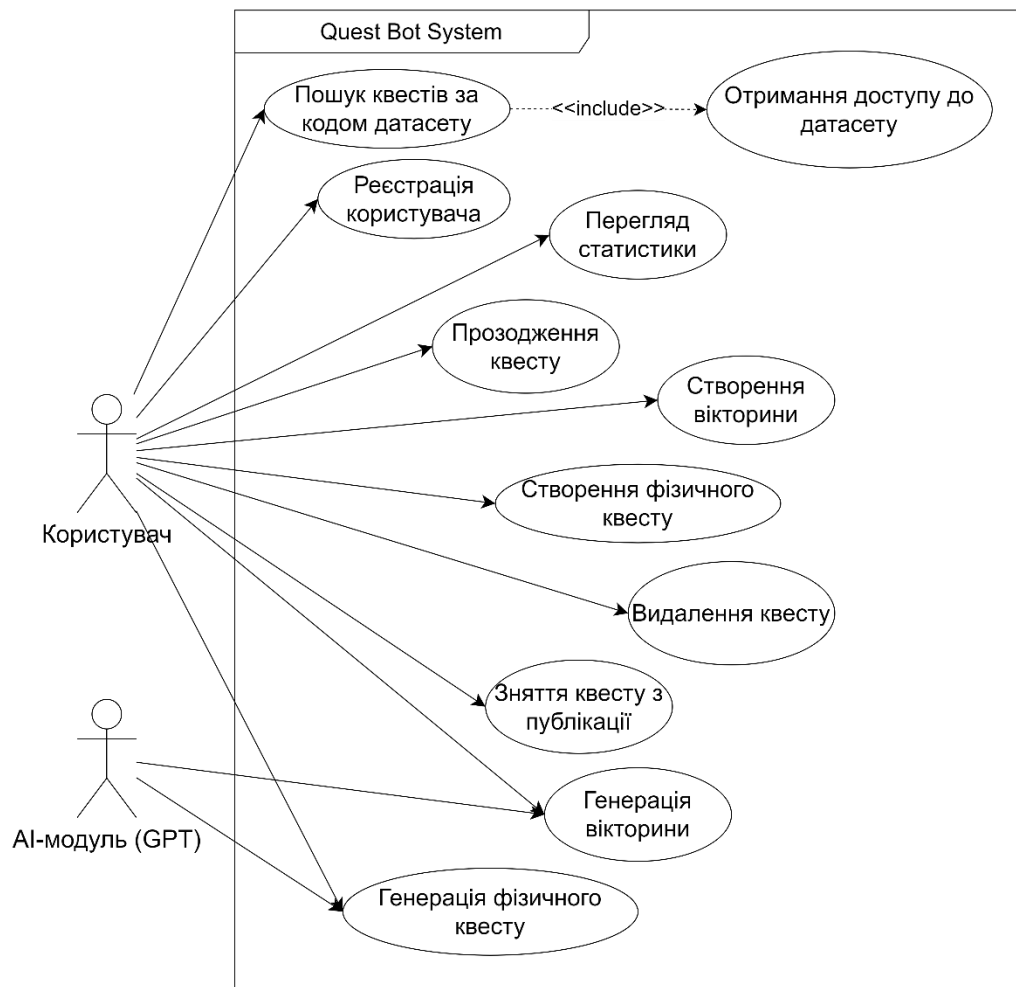


Рисунок 2.2 – Діаграма варіантів використання Telegram-бота для генерації квестів

Усі варіанти використання, доступні користувачеві, можна поділити на кілька функціональних груп: управління обліковим записом, робота з квестами, обмін датасетами та генерація за допомогою ШІ.

1. Управління обліковим записом і статистикою: користувач має можливість зареєструватися в системі «Реєстрація користувача». Реєстрація відбувається автоматично при першому запуску бота, але з точки зору діаграми це окремий варіант використання, який запускається командою /start. Після реєстрації стає доступним перегляд статистики «Перегляд статистики», де відображаються рівень, кількість пройдених квестів, правильних і неправильних відповідей.

2. Проходження квесту: центральний варіант використання «Проходження квесту». Він включає отримання питань або завдань, введення відповідей, нарахування балів та фіксацію результату. Користувач може проходити як створені самостійно квести, так і згенеровані ШІ або отримані через датасети.

3. Створення та управління квестами: користувач може створювати два типи квестів, вікторини «Створення вікторини» та фізичні квести «Створення фізичного квесту». У першому випадку додаються питання з варіантами відповідей, у другому, описи вправ із балами. Для вже створених квестів передбачено видалення «Видалення квесту». Крім того, автор може опублікувати свій квест «Публікація квесту», після чого він стає доступним для інших через код датасету. За потреби публікацію можна скасувати «Зняття квесту з публікації».

4. Робота з датасетами: це набір квестів, об'єднаних спільним кодом. Користувач може знайти квести за кодом «Пошук квестів за кодом датасету», та отримати доступ до вмісту датасету «Отримання доступу до датасету». Зв'язок між «Пошук квестів за кодом датасету» та «Отримання доступу до датасету» позначено як include, тобто пошук за кодом завжди передбачає подальший доступ до знайденого датасету, перелік квестів у ньому.

5. Генерація квестів за допомогою ШІ: два варіанти використання, «Генерація вікторини» та «Генерація фізичного квесту», реалізуються із залученням зовнішнього актора «AI-модуль (GPT)». Користувач ініціює генерацію, передаючи параметри, кількість питань, тему або складність, а бот надсилає запит до GPT API. Отриманий контент обробляється та повертається користувачеві у форматі квесту. На діаграмі це показано зв'язками між актором AI та відповідними варіантами використання.

Усі інші варіанти використання виконуються виключно актором «Користувач», без участі AI-модуля.

2.2.2 Діаграма компонентів системи

Діаграма компонентів показує архітектуру програмного забезпечення на рівні великих блоків, компонентів, їхніх залежностей та інтерфейсів взаємодії. Для Telegram-бота, який автоматизує формування квестів, виділено сім основних компонентів, одну базу даних та два зовнішні об'єкти: клієнт Telegram і хмарне API OpenAI. Діаграму компонентів наведено на рисунку 2.3 [13].

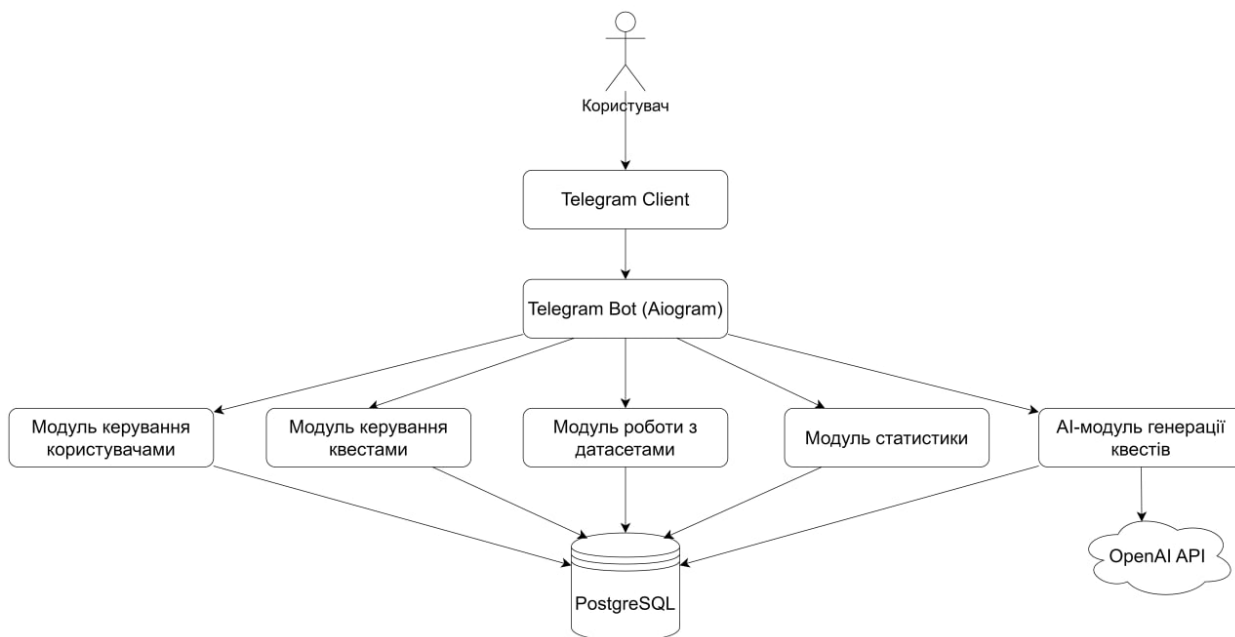


Рисунок 2.3 – Діаграма компонентів системи

Зовнішнім актором, як і раніше, виступає «Користувач». Він взаємодіє з системою через Telegram Client, мобільний або десктопний застосунок Telegram. Клієнт передає повідомлення та callback-запити до основного компонента Telegram Bot (Aiogram) [12]. Цей компонент реалізований на базі фреймворку Aiogram, який забезпечує маршрутизацію повідомлень, обробку станів (FSM) та виклик бізнес-логіки.

Компонент «Telegram Bot» не виконує всю логіку самостійно, а делегує її чотирьом спеціалізованим модулям керування:

1. Модуль керування користувачами – відповідає за реєстрацію нових користувачів, пошук за Telegram ID, отримання або створення запису в базі даних.
2. Модуль керування квестами – містить функції створення, редагування, видалення квестів, додавання питань і варіантів відповідей, а також логіку проходження квесту: перевірка відповідей, нарахування балів.
3. Модуль роботи з датасетами – забезпечує створення датасетів, генерацію унікального коду, додавання та видалення квестів з датасету, пошук датасету за кодом.
4. Модуль статистики – відстежує прогрес користувача: кількість пройдених квестів, правильних і неправильних відповідей, середній бал, рівень.

Усі перелічені модулі мають прямий доступ до PostgreSQL. Кожен модуль виконує запити на читання або запис через SQLAlchemy Core або ORM.

Окремо виділено AI-модуль генерації квестів. Він отримує запит від основного бота, наприклад, команда згенерувати вікторину на певну тему, формує промпти та надсилає їх до зовнішнього хмарного сервісу OpenAI API [11]. Отримана відповідь парситься, JSON або текст, перетворюється у формат запитань і відповідей, після чого зберігається через модуль керування квестами або одразу повертається користувачеві. AI-модуль також може

звертатися до бази даних, наприклад, для збереження згенерованих квестів або для визначення поточної складності.

Зв'язки між компонентами є прямими синхронними викликами [8], через функції або методи класів. Telegram Bot виступає як фасад, який маршрутизує запити. База даних PostgreSQL слугує єдиним джерелом істини для всіх модулів, що забезпечує цілісність даних: користувачі, квести, прогрес, статистика, датасети. OpenAI API, на відміну від інших компонентів, не зберігає стан і не має зворотного зв'язку з системою, крім відповіді на поточний запит.

Така архітектура дозволяє незалежно розвивати окремі модулі, наприклад, замінити AI-провайдера без змін у всій системі, наприклад підключити свою модель навчену на своїх даних та спрощує тестування.

2.2.3 Діаграма класів системи

Діаграма класів відображає об'єктно-орієнтовану структуру системи автоматизованого формування квестів. Основу моделі становлять доменні сутності User, Quest, Question, Answer, Dataset, QuestProgress та UserStatistic, які реалізовані засобами SQLAlchemy ORM. Окрім цього, до діаграми включено сервісний клас AIQuestGenerator, що відповідає за взаємодію із сервісами штучного інтелекту та автоматичне формування квестів. Такий підхід дозволяє відобразити не лише структуру даних системи, а й ключові компоненти її функціональної архітектури. Діаграму наведено на рисунку 2.4.

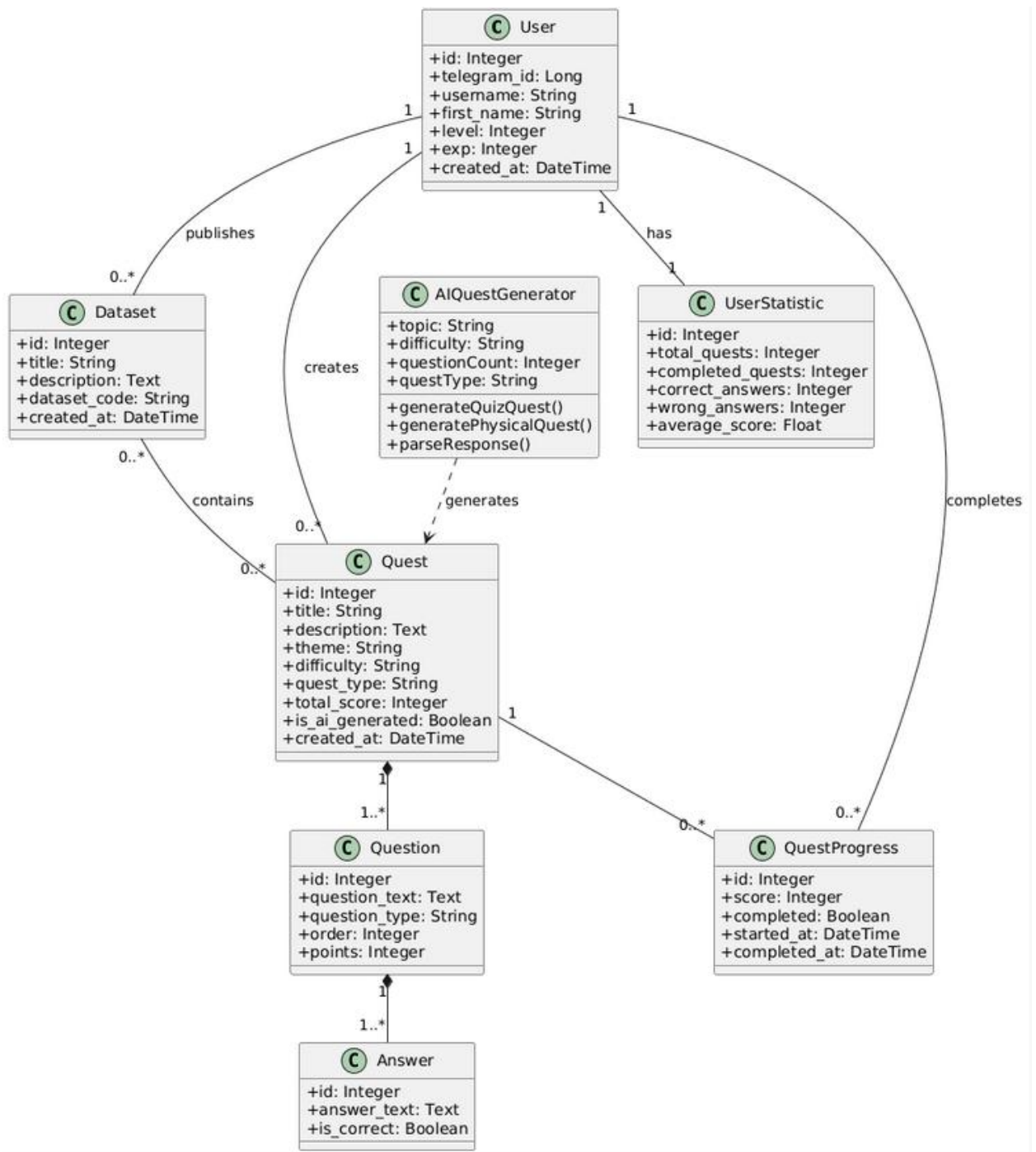


Рисунок 2.4 – Діаграма класів системи

User зберігає ідентифікаційні дані користувача: telegram_id, username, first_name, а також ігрові характеристики, level та exp. Quest представляє квест: title, description, theme, difficulty, quest_type, total_score, is_ai_generated. Question містить текст питання, його тип, порядок у квесті та кількість балів. Answer – варіант відповіді з текстом та позначкою правильності is_correct.

Dataset призначений для об'єднання квестів у набори; має `dataset_code`, унікальний рядок для доступу. QuestProgress фіксує прогрес користувача: `score`, `completed`, `started_at`, `completed_at`. UserStatistic збирає узагальнену статистику: кількість пройдених квестів, правильних і неправильних відповідей, середній бал.

Між User і Quest зв'язок «один до багатьох»: один користувач може створити багато квестів. User також пов'язаний із Dataset, публікація датасетів та QuestProgress, проходження квестів так само «один до багатьох». З UserStatistic зв'язок «один до одного»: кожен користувач має рівно один запис статистики.

Quest містить Question через композицію: один до багатьох, оскільки питання не існує окремо від квесту. Аналогічно Question містить Answer. Quest також пов'язаний із QuestProgress, один квест може мати багато записів прогресу від різних користувачів.

Dataset і Quest з'єднані асоціацією «багато-до-багатьох»: один датасет містить багато квестів, один квест може входити до багатьох датасетів. Цей зв'язок реалізований через окрему допоміжну таблицю `dataset_questions`.

Клас AIQuestGenerator не зберігається в базі даних, а використовується для генерації квестів через зовнішнє API. Він має атрибути: `topic`, `difficulty`, `questionCount`, `questType`. Методи: `generateQuizQuest()` формує запит до AI та повертає структуровану вікторину; `generatePhysicalQuest()` генерує набір описів фізичних вправ; `parseResponse()` обробляє відповідь у форматі JSON або тексту. Залежність між AIQuestGenerator та Quest позначена як пунктирна стрілка, залежність, оскільки генератор створює об'єкти квестів, але не володіє ними постійно. Така архітектура дозволяє змінювати або замінювати механізм генерації без порушення основної структури даних.

2.2.4 Проектування бази даних

Для зберігання інформації про користувачів, квести, питання, відповіді, прогрес та датасети використано реляційну базу даних PostgreSQL [10]. Доступ до даних реалізовано через об'єктно-реляційне відображення ORM

(Object-Relational Mapping) за допомогою бібліотеки SQLAlchemy. ORM-підхід дозволяє працювати з таблицями бази даних як із Python-класами, що спрощує написання запитів, підвищує читабельність коду та зменшує ризик помилок, пов'язаних із ручним складанням SQL-рядків.

Вибір SQLAlchemy зумовлений кількома факторами. По-перше, вона підтримує асинхронний режим роботи [9], через `async_sessionmaker` та `create_async_engine`, що критично для Telegram-бота, який одночасно обробляє запити багатьох користувачів. По-друге, SQLAlchemy надає декларативний синтаксис опису моделей, що робить структуру бази даних прозорою та легкою для підтримки. По-третє, бібліотека забезпечує незалежність від конкретної СУБД, розробку можна вести на SQLite, а в продакшені використовувати PostgreSQL без змін у коді моделей [9]. Підключення до бази даних наведено на лістингу 2.1

Лістинг 2.1 – Підключення до бази даних та створення таблиць

```
from sqlalchemy.ext.asyncio import create_async_engine,
async_sessionmaker, AsyncSession
from sqlalchemy.orm import DeclarativeBase
from dotenv import load_dotenv
import os
load_dotenv()
engine = create_async_engine(url=os.getenv("SQLALCHEMY_URL"),
echo=True)
async_session = async_sessionmaker(bind=engine,
class_=AsyncSession, expire_on_commit=False)
class Base(DeclarativeBase):
pass
```

Engine створюється на основі URL-рядка зі змінної оточення `SQLALCHEMY_URL` [9]. Параметр `echo=True` увімкнює логування всіх SQL-запитів, що корисно під час налагодження. Фабрика сесій `async_session` генерує асинхронні сесії для роботи з базою даних. Базовий клас `Base` є предком для всіх ORM-моделей. Таблиця зв'язку між датасетами та квестами наведена на лістингу 2.2.

Лістинг 2.2 – Таблиця зв'язку між датасетами та квестами

```
from sqlalchemy import Table, Column, ForeignKey
dataset_quests = Table(
    "dataset_quests", Base.metadata,
    Column("dataset_id", ForeignKey("datasets.id"),
primary_key=True),
    Column("quest_id", ForeignKey("quests.id"),
primary_key=True)
)
```

Це допоміжна таблиця для реалізації зв'язку «багато-до-багатьох» між датасетами та квестами. Вона містить два зовнішні ключі, які разом утворюють складений первинний ключ. Модель користувача, «таблиця User», наведена на лістингу 2.3 [7].

Лістинг 2.3 – Модель користувача

```
class User(Base):
    __tablename__ = "users"
    id: Mapped[int] = mapped_column(primary_key=True)
    telegram_id: Mapped[int] = mapped_column(BigInteger,
unique=True)
    username: Mapped[str] = mapped_column(String(64),
nullable=True)
    first_name: Mapped[str] = mapped_column(String(64))
    level: Mapped[int] = mapped_column(Integer, default=1)
    exp: Mapped[int] = mapped_column(Integer, default=0)
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)
    quests = relationship("Quest", back_populates="creator")
    progresses = relationship("QuestProgress",
back_populates="user")
    datasets = relationship("Dataset", back_populates="creator")
    statistics = relationship("UserStatistic",
back_populates="user", uselist=False)
```

Таблиця users зберігає облікові записи. Поле telegram_id є унікальним ідентифікатором користувача в Telegram, за яким виконується пошук при реєстрації. Атрибути level та exp відповідають за ігровий прогрес. Зв'язки через relationship визначають, що користувач може мати багато квестів, багато записів прогресу, багато датасетів та один запис статистики (uselist=False). Модель квесту, «таблиця Quest» наведена на лістингу 2.4.

Лістинг 2.4 – Модель квесту

```
class Quest(Base):
    __tablename__ = "quests"
    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str] = mapped_column(String(255))
    description: Mapped[str] = mapped_column(Text)
    theme: Mapped[str] = mapped_column(String(100))
    difficulty: Mapped[str] = mapped_column(String(20))
    quest_type: Mapped[str] = mapped_column(String(20),
default="quiz")
    total_score: Mapped[int] = mapped_column(Integer, default=0)
    is_ai_generated: Mapped[bool] = mapped_column(Boolean,
default=False)
    creator_id: Mapped[int] =
mapped_column(ForeignKey("users.id"))
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)
    creator = relationship("User", back_populates="quests")
    questions = relationship("Question", back_populates="quest",
cascade="all, delete-orphan")
    progresses = relationship("QuestProgress",
back_populates="quest", cascade="all, delete-orphan")
    datasets = relationship("Dataset", secondary=dataset_quests,
back_populates="quests")
```

Таблиця `quests` містить усі квести. `quest_type` розрізняє вікторини: `quiz` та фізичні квести `physical`. `total_score` обчислюється як сума балів усіх питань. Параметр `cascade="all, delete-orphan"` у зв'язках гарантує, що при видаленні квесту автоматично видаляються всі його питання та записи прогресу. Моделі питань та відповідей, «таблиця `Question`» наведена на лістингу 2.5.

Лістинг 2.5 – Моделі питань та відповідей

```
class Question(Base):
    __tablename__ = "questions"
    id: Mapped[int] = mapped_column(primary_key=True)
    quest_id: Mapped[int] =
mapped_column(ForeignKey("quests.id"))
    question_text: Mapped[str] = mapped_column(Text)
    question_type: Mapped[str] = mapped_column(String(50))
    order: Mapped[int] = mapped_column(Integer)
    points: Mapped[int] = mapped_column(Integer, default=1)

    quest = relationship("Quest", back_populates="questions")
    answers = relationship("Answer", back_populates="question",
cascade="all, delete-orphan")
class Answer(Base):
```

```

__tablename__ = "answers"
id: Mapped[int] = mapped_column(primary_key=True)
question_id: Mapped[int] =
mapped_column(ForeignKey("questions.id"))
answer_text: Mapped[str] = mapped_column(Text)
is_correct: Mapped[bool] = mapped_column(Boolean,
default=False)
question = relationship("Question",
back_populates="answers")

```

Таблиці questions та answers зберігають структуру вікторини. Поле order визначає порядок показу питань. Для фізичних квестів question_type встановлюється в physical, а відповіді мають лише одну фіктивну правильну відповідь «Виконано». Каскадне видалення забезпечує очищення відповідей при видаленні питання. Модель прогресу проходження, «таблиця QuestProgress» наведена на лістингу 2.6.

Лістинг 2.6 – Модель прогресу проходження

```

class QuestProgress(Base):
    __tablename__ = "quest_progress"
    id: Mapped[int] = mapped_column(primary_key=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("users.id"))
    quest_id: Mapped[int] =
mapped_column(ForeignKey("quests.id", ondelete="CASCADE"))
    score: Mapped[int] = mapped_column(Integer, default=0)
    completed: Mapped[bool] = mapped_column(Boolean,
default=False)
    started_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)
    completed_at: Mapped[datetime] = mapped_column(DateTime,
nullable=True)
    user = relationship("User", back_populates="progresses")
    quest = relationship("Quest", back_populates="progresses")

```

Таблиця quest_progress фіксує кожну спробу проходження квесту. Це дозволяє користувачеві переривати квест і починати заново, не втрачаючи історію попередніх спроб. Значення ondelete="CASCADE" гарантує, що при видаленні квесту всі пов'язані записи прогресу також будуть видалені. Модель датасету, «таблиця Dataset» наведена на лістингу 2.7.

Лістинг 2.7 – Модель датасету

```
class Dataset(Base):
    __tablename__ = "datasets"
    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str] = mapped_column(String(255))
    description: Mapped[str] = mapped_column(Text)
    dataset_code: Mapped[str] = mapped_column(String(32),
unique=True, index=True)
    creator_id: Mapped[int] =
mapped_column(ForeignKey("users.id"))
    created_at: Mapped[datetime] = mapped_column(DateTime,
default=datetime.utcnow)
    creator = relationship("User", back_populates="datasets")
    quests = relationship("Quest", secondary=dataset_quests,
back_populates="datasets")
```

Датасети є контейнерами для публікації квестів. Поле `dataset_code` унікальний шестисимвольний код наприклад: «A3F9K2», за яким інші користувачі можуть знайти набір. Індекс на цьому полі прискорює пошук. Модель статистики користувача, «таблиця `UserStatistic`» наведена на лістингу 2.8.

Лістинг 2.8 – Модель статистики користувача

```
class UserStatistic(Base):
    __tablename__ = "user_statistics"
    id: Mapped[int] = mapped_column(primary_key=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("users.id"),
unique=True)
    total_quests: Mapped[int] = mapped_column(Integer,
default=0)
    completed_quests: Mapped[int] = mapped_column(Integer,
default=0)
    correct_answers: Mapped[int] = mapped_column(Integer,
default=0)
    wrong_answers: Mapped[int] = mapped_column(Integer,
default=0)
    average_score: Mapped[float] = mapped_column(Float,
default=0)
    user = relationship("User", back_populates="statistics")
```

Таблиця `user_statistics` агрегує показники ефективності користувача. `total_quests`: кількість розпочатих квестів, `completed_quests`: успішно

завершених. `average_score` обчислюється як відношення правильних відповідей до загальної кількості відповідей.

Таким чином, ORM-моделі повністю описують логічну схему бази даних. Кожен клас відповідає одній таблиці, атрибути – стовпцям, а зв'язки через `Relationship` та `ForeignKey` визначають реляційну структуру. Такий підхід забезпечує безпечний доступ до даних, спрощує міграції та робить код бота більш підтримуваним.

2.2.5 ER-діаграма бази даних

ER-діаграма відображає логічну структуру бази даних у вигляді сутностей, їхніх атрибутів та зв'язків. Для системи автоматизованого формування квестів розроблено реляційну схему, яка складається з семи основних таблиць та однієї допоміжної для реалізації зв'язку «багато-до-багатьох». ER-діаграму наведено на рисунку 2.5.

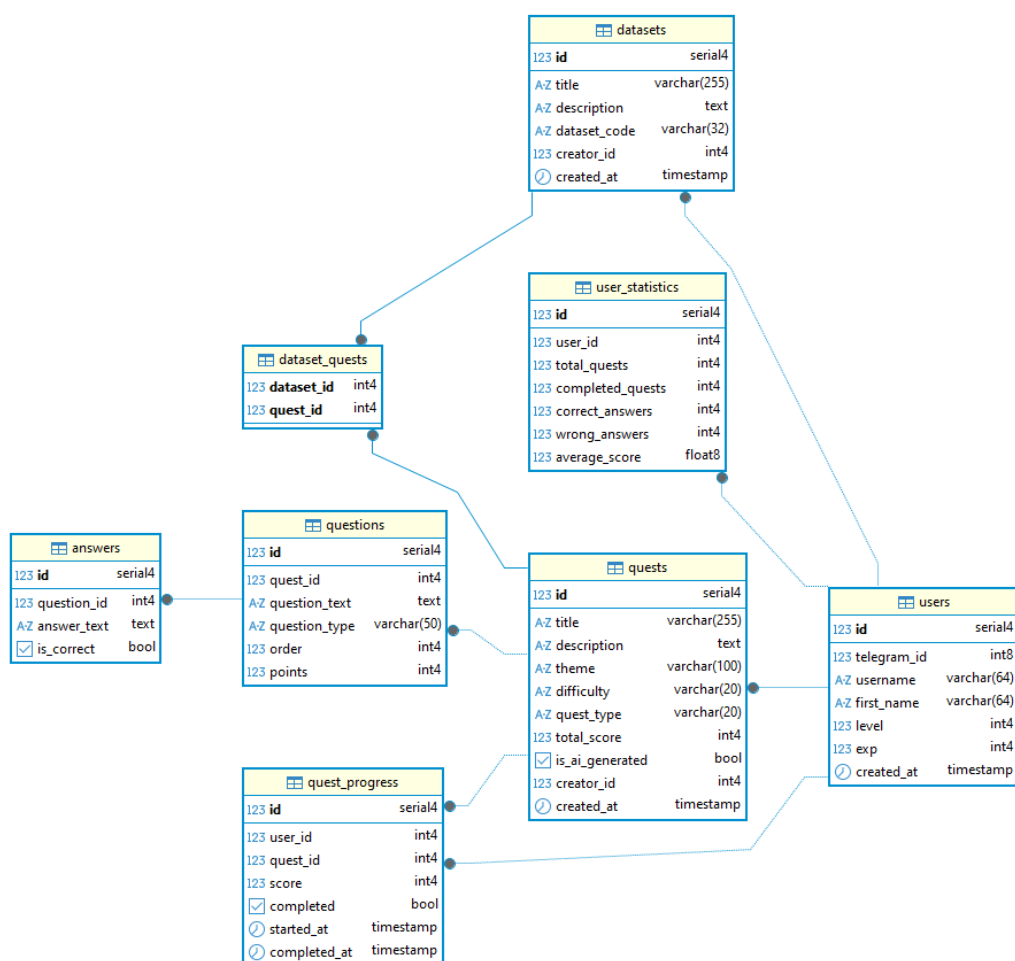


Рисунок 2.5 – ER-діаграма бази даних

Таблиця «Users» призначена для зберігання облікових записів користувачів, вона містить такі поля:

- id – унікальний ідентифікатор користувача (первинний ключ);
- telegram_id – унікальний ідентифікатор користувача в Telegram (тип int8);
- username – ім'я користувача в Telegram (може бути відсутнім);
- first_name – ім'я, вказане в профілі Telegram;
- level – поточний рівень користувача (зростає після певної кількості пройдених квестів);
- exp – накопичений досвід;
- created_at – дата та час реєстрації.

Таблиця «Quest» зберігає всі квести, створені користувачами або згенеровані ШІ, вона містить такі поля:

- id – унікальний ідентифікатор квесту;
- title – назва квесту;
- description – опис (для вікторин – загальний опис, для фізичних квестів – інструкція);
- theme – тематика (наприклад, «Історія», «Фізика»);
- difficulty – рівень складності (easy, medium, hard);
- quest_type – тип квесту (quiz або physical);
- total_score – максимальна сума балів;
- is_ai_generated – ознака того, чи створено квест штучним інтелектом;
- created_at – дата створення.

До таблиці «Quest» неявно додається зовнішній ключ creator_id, який посилається на users.id, але в наведеній схемі він відсутній. У реальній реалізації цей зв'язок існує, тому що кожен квест належить автору.

Таблиця «Questions» містить питання або завдання квесту, вона містить такі поля:

- id – унікальний ідентифікатор питання;

- `quest_id` – зовнішній ключ, що пов’язує питання з конкретним квестом;
- `question_text` – текст питання (або опис фізичної вправи);
- `question_type` – тип (`single` для вікторини, `physical` для вправи);
- `order` – порядковий номер питання в межах квесту;
- `points` – кількість балів за правильну відповідь (або за виконання вправи).

Таблиця «`Answers`» зберігає варіанти відповідей для питань-вікторин. , вона містить такі поля:

- `id` – унікальний ідентифікатор відповіді;
- `question_id` – зовнішній ключ, що вказує на відповідне питання;
- `answer_text` – текст варіанта відповіді;
- `is_correct` – логічне поле, що визначає правильний варіант (лише один із чотирьох має значення `true`).

Таблиця «`Quest_progress`» фіксує хід проходження квестів користувачами, вона містить такі поля:

- `id` – унікальний ідентифікатор запису прогресу;
- `user_id` – зовнішній ключ до `users.id`;
- `quest_id` – зовнішній ключ до `quest.id`;
- `score` – набрана кількість балів;
- `completed` – позначка завершення квесту;
- `started_at` – час початку проходження;
- `completed_at` – час завершення (може бути `null`, якщо квест не завершено).

Таблиця «`Datasets`» призначена для об’єднання квестів у публічні набори, вона містить такі поля:

- `id` – унікальний ідентифікатор датасету;
- `title` – назва датасету;
- `description` – опис вмісту;

- `dataset_code` – унікальний код (шість символів), за яким інші користувачі знаходять датасет;
- `creator_id` – зовнішній ключ до `users.id` (автор, який опублікував набір);
- `created_at` – дата публікації.

Таблиця «`User_statistics`» агрегує показники ефективності користувача, вона містить такі поля:

- `id` – унікальний ідентифікатор запису статистики;
- `user_id` – зовнішній ключ до `users.id` (зв'язок «один-до-одного», кожен користувач має один запис);
- `total_quests` – загальна кількість розпочатих квестів;
- `completed_quests` – кількість успішно завершених квестів;
- `correct_answers` – кількість правильних відповідей;
- `wrong_answers` – кількість неправильних відповідей;
- `average_score` – середній бал (обчислюється як відношення правильних відповідей до загальної кількості).

Між `users` та `quest` існує зв'язок «один-до-багатьох»: один користувач може створити багато квестів. Аналогічні зв'язки: `users` – `quest_progress`: один користувач має багато записів прогресу, `users` – `datasets`: один користувач публікує багато датасетів, `users` – `user_statistics`: один користувач має один запис статистики, зв'язок «один-до-одного».

`Quest` – `questions` – зв'язок «один-до-багатьох»: один квест містить багато питань. `questions` – `answers` – теж «один-до-багатьох»: одне питання має чотири варіанти відповідей.

`Uest` – `quest_progress` – «один-до-багатьох»: один квест може бути пройдений багатьма користувачами, кожен з яких має окремий запис прогресу.

Зв'язок між `datasets` та `quest` є «багато-до-багатьох»: один датасет може містити багато квестів, і один квест може входити до багатьох датасетів. Цей зв'язок реалізований через допоміжну таблицю `dataset_quests`. Вона містить

два зовнішні ключі – `dataset_id` та `quest_id` – які разом утворюють складений первинний ключ.

2.3 Алгоритмічне забезпечення

2.3.1 Алгоритм створення користувацької вікторини

Наведена блок-схема, рисунок 2.6, ілюструє процес створення вікторини самим користувачем без залучення штучного інтелекту. Цей алгоритм реалізований у відповідних обробниках станів «CreateQuest» і є одним із ключових сценаріїв конструктора квестів.

Процес можна розділити на три великі блоки: підготовчий етап, цикл додавання питань та фінальне збереження.

Підготовчий етап: після того, як користувач обирає пункт «Створити вікторину» в меню, бот переходить у стан очікування назви. Користувач вводить довільну назву, вона стане заголовком майбутнього квесту. Без назви подальше створення неможливе.

Основний цикл, додавання питань: далі починається повторювана послідовність дій для кожного питання:

- користувач вводить текст питання;
- додає чотири варіанти відповіді (кожен з нового рядка);
- обирає один правильний варіант, вказуючи його номер (1, 2, 3 або 4);
- визначає кількість балів, яку отримає гравець за правильну відповідь.

Після цих кроків питання разом із варіантами, правильною відповіддю та балами тимчасово зберігається в пам'яті, у стані FSM, але ще не записується в базу даних. Бот пропонує користувачеві додати ще одне питання. Якщо той погоджується, цикл повторюється з початку. Якщо ні, алгоритм переходить до завершального етапу.

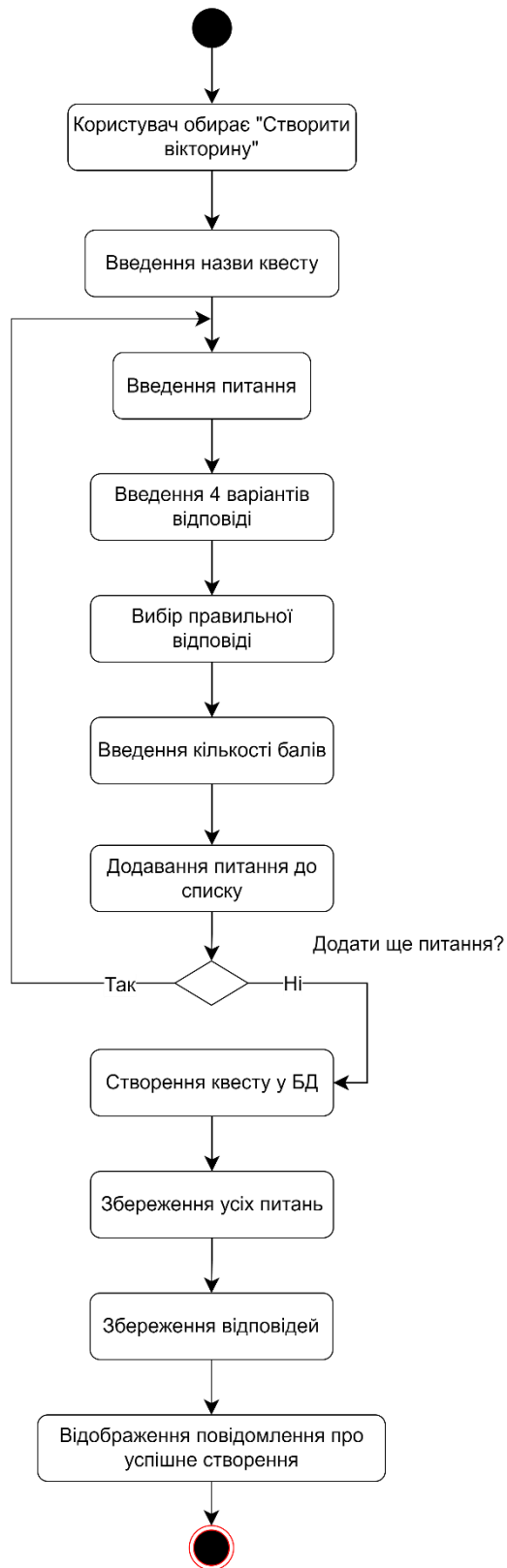


Рисунок 2.6 – Алгоритм створення користувацької вікторини

Завершальний етап (збереження): коли користувач вирішив не додавати більше питань, бот створює запис квесту в таблиці `quests` бази даних.

Отриманий `quest_id` використовується для зв'язування з питаннями. Далі для кожного зібраного питання створюється окремий запис у таблиці `questions`, а для кожного з чотирьох варіантів, запис у таблиці `answers` з позначкою `is_correct`, лише один варіант має значення `True`. Після успішного збереження всіх даних бот відправляє користувачеві повідомлення про успішне створення квесту.

2.3.2 Алгоритм генерації вікторини засобами штучного інтелекту

Алгоритм, зображений на рисунку 2.7, відображає процес автоматизованого створення вікторини з використанням мовної моделі (через OpenAI API). Цей сценарій реалізований у станах «GenerateQuiz» і дозволяє користувачеві отримати унікальний набір питань без необхідності вручну їх формулювати.

Алгоритм починається з того, що користувач обирає режим «Квест-вікторина» у головному меню або в меню вибору типів квестів. Після цього бот послідовно запитує два параметри: кількість питань, доступні варіанти: 10, 20, 30 або 40 та тему. Теми можуть бути як конкретними: історія, українська література, філософія, хімія, фізика, так і випадковою «рандом», коли ШІ обирає тему самостійно.

Коли обидва параметри визначено, бот формує текстовий промпт. Промпт містить чітку інструкцію: згенерувати задану кількість питань на обрану тему, кожне з чотирма варіантами відповідей, лише одна з яких правильна. Результат має бути поданий у вигляді списку JSON-об'єктів з ключами «question», «options» та «correct». Цей промпт надсилається до AI-модуля, який виступає посередником між ботом і хмарним API OpenAI.

Наступним кроком алгоритм перевіряє, чи отримано відповідь від AI-модуля. У разі успіху отриманий текст очищується від можливих маркерів коду наприклад: `json`, і парситься як JSON. Якщо структура відповідає очікуваній масив об'єктів, питання з варіантами та правильною відповіддю зберігаються у тимчасовій пам'яті FSM (Finite State Machine). Після цього бот

одразу переходить до режиму проходження згенерованої вікторини, показуючи перше питання.

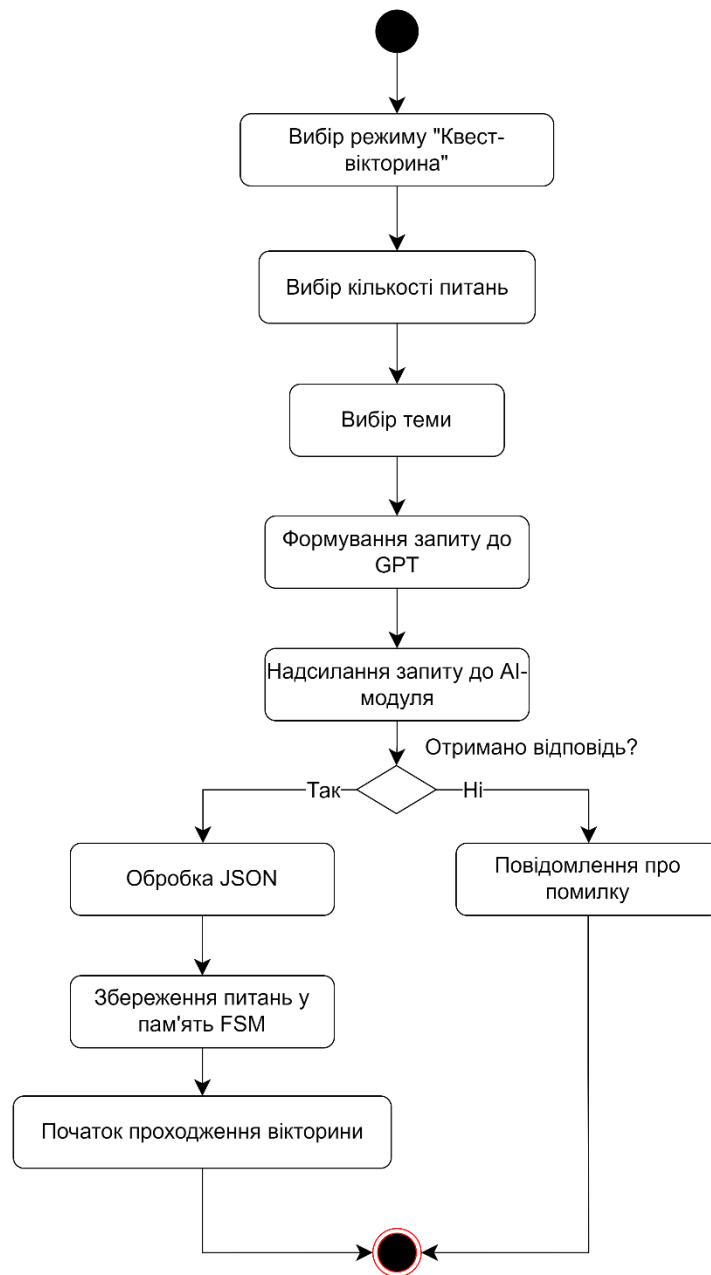


Рисунок 2.7 – Алгоритм генерації вікторини засобами штучного інтелекту

Якщо відповідь не отримано, наприклад, мережева помилка, тайм-аут або некоректний JSON, алгоритм переходить до гілки «Ні». У цьому випадку користувачеві виводиться повідомлення про помилку, пропонується спробувати пізніше, а стан генерації очищується. Жодні дані не зберігаються, що запобігає появі пошкоджених квестів.

Важливо, що згенерована вікторина не зберігається в базі даних автоматично. Користувач може пройти її одразу, а за потреби повторити генерацію з тими самими параметрами, отримавши щоразу новий набір питань, завдяки стохастичності LLM. Такий підхід забезпечує варіативність контенту при кожному запуску. Однак, отриманий бал за проходження вікторини, нараховується на аккаунт користувача.

2.3.3 Алгоритм проходження квесту

На рисунку 2.8 подано алгоритм, який реалізує основний ігровий цикл, послідовне виконання запитань квесту з накопиченням балів та фіксацією результату. Цей сценарій є спільним як для вікторин, так і для фізичних квестів, хоча механізм отримання відповіді дещо відрізняється, вибір варіанту проти позначки «Виконав».

Алгоритм починається з отримання об'єкта квесту з бази даних разом із усіма питаннями та відповідями, або описами вправ. Потім ініціалізується прогрес: створюється або отримується запис у таблиці `quest_progress` для поточного користувача і заданого квесту, початковий рахунок встановлюється в нуль.

Далі починається цикл для кожного питання зі списку. На кожній ітерації бот відображає текст питання, або опис фізичної вправи, а також кількість балів, яку можна отримати. У випадку вікторини користувачеві пропонуються кнопки з варіантами відповідей, у випадку фізичного квесту, кнопки «Виконав» або «Пропустити».

Після отримання дії користувача алгоритм перевіряє, чи відповідь правильна, для вікторини, зіставлення з `is_correct`, для фізичного квесту, натискання «Виконав» вважається успішним виконанням. Якщо відповідь правильна, до поточного рахунку додаються бали питання, зазначені в полі `points`. Якщо неправильна, бали не нараховуються, а користувачу показується правильна відповідь, для вікторин, текст правильної відповіді; для фізичних вправ, повідомлення про пропуск. Після цього алгоритм перевіряє, чи залишилися ще питання. Якщо так, цикл повторюється з наступним питанням.



Рисунок 2.8 – Алгоритм проходження квесту

Коли всі питання вичерпано, алгоритм виходить із циклу та оновлює статистику користувача. Це включає збільшення лічильника пройдених квестів, додавання кількості правильних і неправильних відповідей, перерахунок середнього балу та, за потреби, підвищення рівня. Далі результати проходження зберігаються в таблиці `quest_progress`, фінальний рахунок, позначка `completed = True`, час завершення. Наостанок бот

відображає фінальне повідомлення з результатом: отримані бали, максимально можливі, а також пропозицію продовжити, обрати наступний квест або повернутися в головне меню.

Окремо варто зазначити, що під час виконання циклу передбачено можливість дострокового переривання квесту, кнопка «Завершити квест». У цьому випадку статистика не оновлюється, а прогрес не зберігається як завершений. Така поведінка дозволяє користувачеві вийти з квесту без штрафу.

2.3.4 Алгоритм генерації фізичного квесту засобами штучного інтелекту

Алгоритм, наведений на рисунку 2.9, ілюструє процес автоматизованого створення фізичного квесту, набору вправ, які користувач має виконати в реальному житті. На відміну від генерації вікторини, тут результатом є не структуровані питання з варіантами, а послідовність текстових описів дій, наприклад, «10 присідань», «30 секунд планки». Генерація також використовує зовнішній AI-модуль.

Першим кроком користувач обирає режим «Фізичні квести» у меню вибору типу квесту. Далі бот запитує кількість вправ, доступні варіанти: 5, 10, 15 або 20, та рівень складності. Рівень складності впливає на два аспекти: по-перше, на формулювання промпту для ШІ, легкий рівень передбачає прості вправи з невисокою інтенсивністю, важкий – складні вправи на витривалість, по-друге, на кількість балів за кожну виконану вправу, легкий – 5 балів, середній – 10, важкий – 15.

Після вибору параметрів бот формує текстовий промпт. Промпт містить чітку вимогу: згенерувати задану кількість різноманітних фізичних вправ визначеного рівня складності, кожна у вигляді короткого опису дії, без нумерації, без зайвих пояснень. Приклад опису вправи: «15 стрибків на місці» або «20 секунд бігу на місці». Цей промпт надсилається до AI-модуля, який виконує запит до OpenAI API.

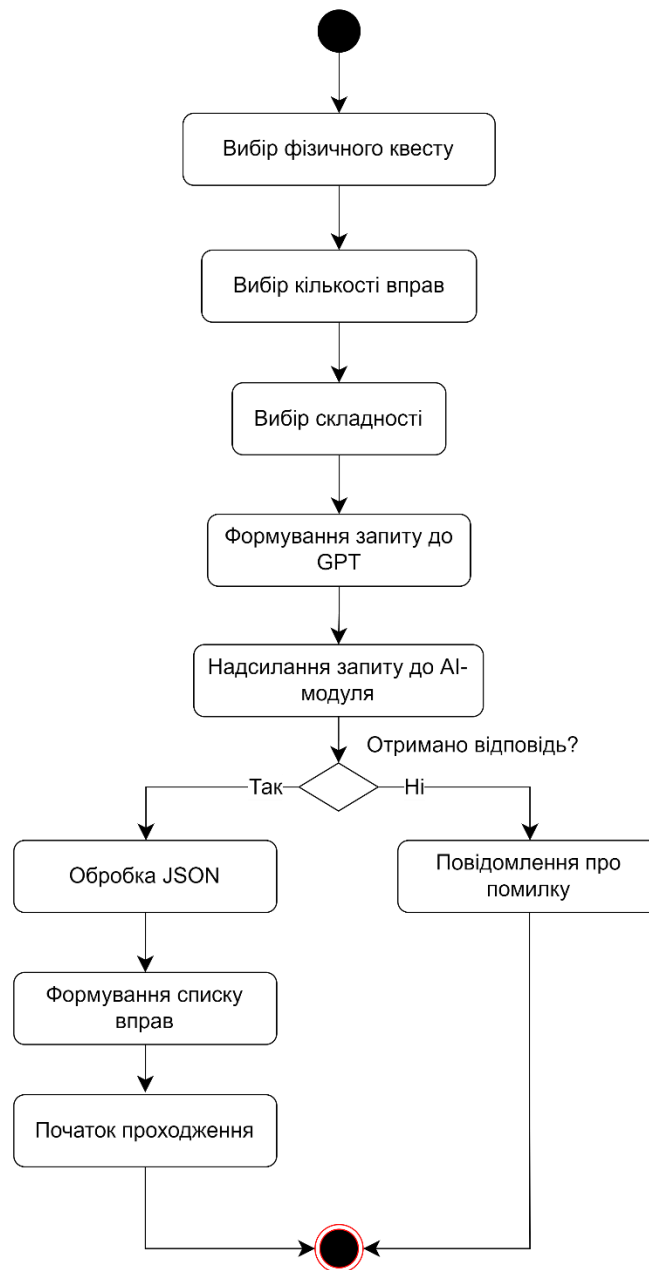


Рисунок 2.9 – Алгоритм генерації фізичного квесту засобами штучного інтелекту

Якщо відповідь не отримано, мережева помилка, тайм-аут або некоректний вміст, алгоритм переходить до гілки «Ні»: користувач отримує повідомлення про помилку з пропозицією спробувати пізніше, а стан генерації очищується. У разі успішної відповіді отриманий текст обробляється: розбивається на окремі рядки, кожен непорожній рядок вважається однією вправою. Якщо кількість отриманих вправ менша за замовлену, недостатня кількість заповнюється стандартними вправами-заповнювачами, наприклад,

«Вправа 1». Якщо вправ більше, список обрізається до потрібної кількості. Сформований масив вправ зберігається у пам'яті FSM разом із параметрами складності, кількість балів за виконання. Після цього бот одразу переходить до режиму проходження, показуючи першу вправу з кнопками «Виконав», «Пропустити» та «Скасувати квест».

Важлива особливість цього алгоритму – згенерований фізичний квест, як і згенерована вікторина, не зберігається в базі даних. Він існує лише протягом поточної сесії гравця. Якщо користувач захоче повторно пройти фізичний квест із тими самими параметрами, система згенерує новий набір вправ, що забезпечує варіативність і унеможливорює завчання послідовності. Бали нараховуються лише за ті вправи, які користувач позначив як виконані; пропущені вправи не додають балів і не штрафують.

2.3.5 Алгоритм публікації квесту в датасет

Алгоритм, зображений на рисунку 2.10, демонструє процес публікації вже створеного користувацького квесту у відкритий доступ через механізм датасетів. Публікація дозволяє іншим користувачам знаходити цей квест за унікальним кодом і додавати його до себе.

Алгоритм починається з того, що користувач обирає певний квест зі свого списку, наприклад, через меню «Свої квести» та натискання на квест і натискає кнопку «Опублікувати». Після цього система виконує перевірку: чи цей квест вже опублікований раніше. Перевірка реалізована через запит до таблиці зв'язку `dataset_quests`, якщо існує запис, де `quest_id` дорівнює поточному, то квест вважається вже опублікованим.

Якщо квест уже опублікований, алгоритм переходить до гілки «Так»: користувач отримує повідомлення про помилку з текстом на зразок «Цей квест вже опубліковано». Жодних змін у базі даних не відбувається, і виконання припиняється. Це запобігає дублюванню одного й того самого квесту в різних датасетах.



Рисунок 2.10 – Алгоритм публікації квесту в датасет

Якщо квест ще не опублікований, виконується гілка «Ні». Система створює новий датасет. Назва датасету формується автоматично як «Квест: [назва квесту]», опис «Містить квест «[назва квесту]»». Далі генерується унікальний код датасету. Код складається з шести символів: великі літери латинського алфавіту та цифри, наприклад, «A3F9K2». Генерація відбувається в циклі з перевіркою унікальності: якщо випадково згенерований код вже існує в таблиці `datasets`, процес повторюється до отримання унікального значення. Після створення запису датасету в базі даних система додає зв'язок між цим датасетом і квестом у таблицю `dataset_quests`.

Фінальним кроком алгоритму є відображення згенерованого коду користувачеві. Повідомлення містить сам код: візуально виділений та інструкцію, як ним поділитися з іншими. Користувач може надіслати цей код будь-кому, отримувач зможе ввести його в розділі «Знайти квести» та отримати доступ до опублікованого квесту. Важливо, що один датасет може містити лише один квест, при створенні через публікацію, але за потреби архітектуру можна розширити для об'єднання кількох квестів в один датасет.

Алгоритм не передбачає автоматичного створення deep-link або інтеграції з Telegram-посиланнями, але код можна поширювати як звичайний текст. Зняття квесту з публікації, видалення датасету реалізоване окремим алгоритмом.

2.3.6 Алгоритм пошуку квестів за кодом датасету

Алгоритм, наведений на рисунку 2.11, ілюструє процес отримання доступу до опублікованих квестів інших користувачів через унікальний код датасету. Цей механізм є ключовим для обміну контентом між користувачами без необхідності надавати прямі права доступу або додавати користувачів у спільники.

Алгоритм починається з того, що користувач вводить код датасету в чат з ботом. Введення здійснюється в текстовому вигляді після вибору пункту меню «Знайти квести». Бот очікує повідомлення з кодом, яке може складатися з шести символів: великі літери та цифри. Код не чутливий до регістру, система автоматично перетворює введений рядок у верхній регістр перед пошуком.

Далі виконується запит до бази даних: пошук запису в таблиці `datasets` за полем `dataset_code`. Якщо запис існує, гілка «Так», система отримує всі квести, пов'язані з цим датасетом через таблицю `dataset_quests`. Отриманий список може містити один або кілька квестів, хоча в поточній реалізації при публікації додається лише один квест, архітектура підтримує об'єднання кількох. Після отримання списку бот відображає користувачеві квести у вигляді кнопок, назва квесту та емодзі, що позначає тип. Користувач обирає

потрібний квест, після чого система переходить до сценарію проходження, показ інформації про квест із кнопкою «Почати» або одразу запуск.

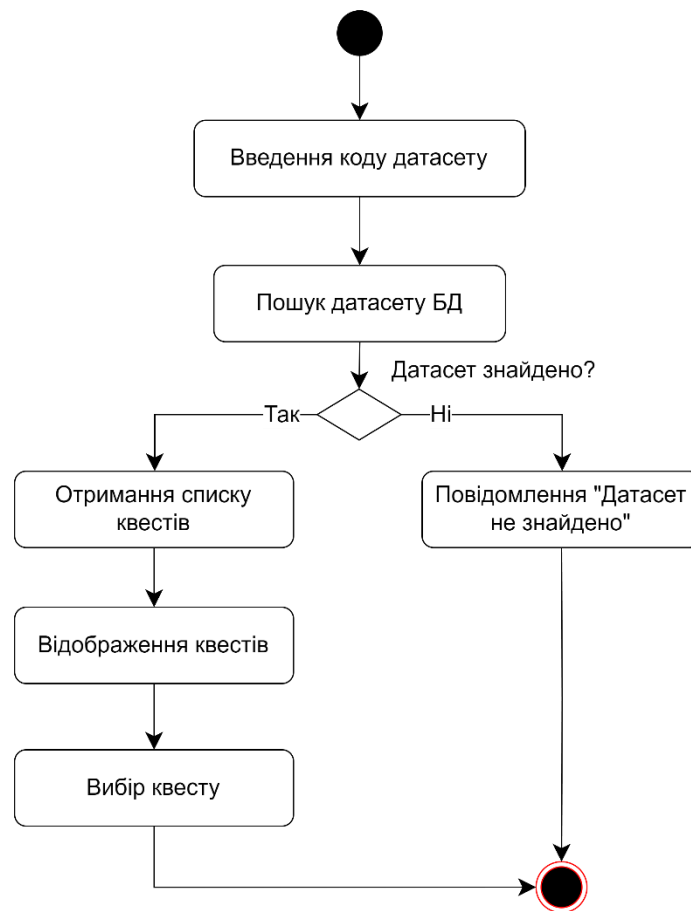


Рисунок 2.11 – Алгоритм пошуку квестів за кодом датасету

Якщо датасет із введеним кодом не знайдено, гілка «Ні», бот виводить повідомлення про помилку: «Датасет з таким кодом не знайдено. Спробуйте ще раз або натисніть /start». Стан очікування коду очищується, і користувач повертається до головного меню або меню вибору типу квестів. Алгоритм не передбачає автоматичного повторного запиту, користувач має самостійно обрати пункт «Знайти квести» знову, якщо хоче спробувати інший код.

Важливою особливістю є те, що знайдений квест не копіюється в особисту бібліотеку користувача. Користувач отримує тимчасовий доступ до проходження, але сам квест залишається у власності автора та не дублюється в базі. Статистика проходження записується на користувача, який проходив

квест, але не впливає на автора. Якщо автор зніме квест із публікації, видалить зв'язок із датасетом, інші користувачі більше не зможуть його знайти за цим кодом. Це забезпечує контроль автора над своїм контентом.

2.4 Проєктування взаємодії компонентів системи

2.4.1 Діаграма послідовності створення користувацького квесту

Діаграма послідовності показує впорядкованість обміну повідомленнями між об'єктами під час виконання певного сценарію. На рисунку 2.12 зображено діаграму для процесу створення користувацької вікторини, або фізичного квесту без залучення ІІІ. У діаграмі беруть участь три елементи: актор «Користувач», учасник «Telegram Bot» та база даних «PostgreSQL».

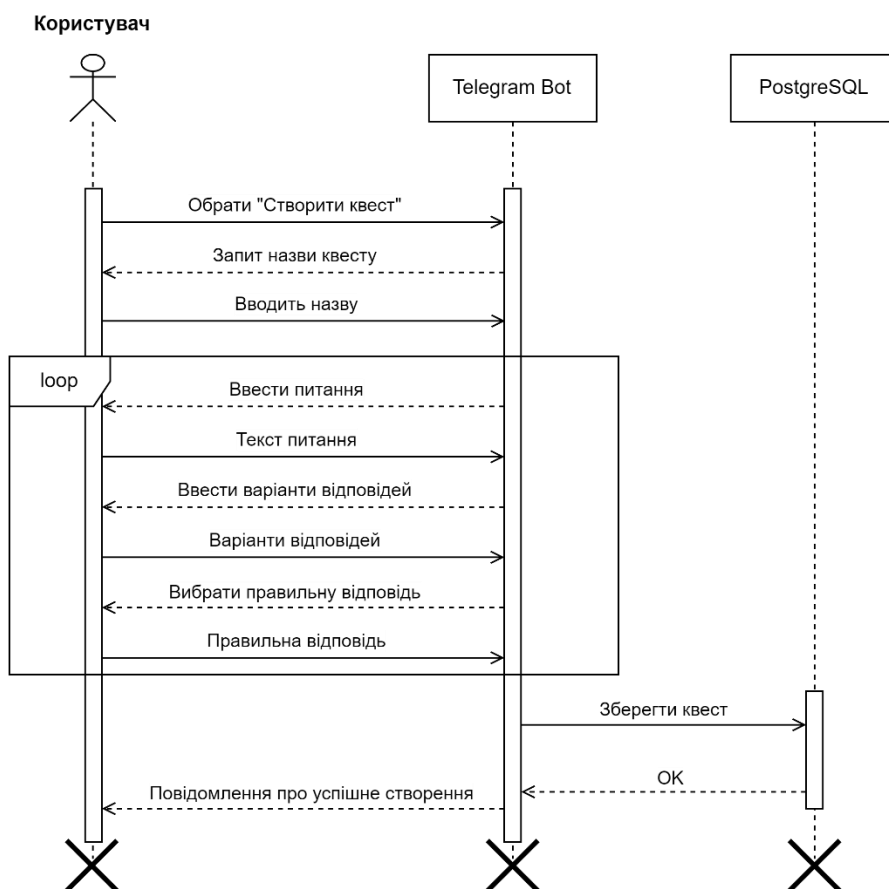


Рисунок 2.12 – Діаграма послідовності створення користувацького квесту

Сценарій починається з того, що користувач обирає пункт «Створити квест» у головному меню. Бот у відповідь надсилає запит на введення назви майбутнього квесту. Користувач вводить назву, після чого починається основний блок, цикл додавання питань. Цикл позначений на діаграмі як loop для кожного питання.

У межах циклу бот послідовно запитує текст питання, чотири варіанти відповідей та номер правильної відповіді. Після кожного введення користувач надсилає відповідні дані.

Після завершення циклу, коли користувач вирішив не додавати більше питань, бот виконує збереження даних у базі даних. Це відбувається у вигляді транзакції: спочатку створюється запис у таблиці `quests`, потім для кожного питання, запис у `questions`, а для кожного варіанту відповіді, у `answers`. Після успішного виконання всіх операцій база даних повертає підтвердження: повідомлення «ОК». Отримавши його, бот надсилає користувачеві фінальне повідомлення про успішне створення квесту.

Діаграма ілюструє синхронний характер взаємодії: кожен запит бота передбачає відповідь користувача, і лише після неї виконується наступна дія. Збереження в базу даних також є синхронною операцією, бот очікує на підтвердження, перш ніж повідомити про результат. Такий підхід спрощує логіку та унеможливорює втрату даних через асинхронні збої.

2.4.2 Діаграма послідовності генерації квесту засобами штучного інтелекту

Діаграма послідовності, наведена на рисунку 2.13, ілюструє сценарій автоматизованого створення квесту, вікторини або фізичних вправ із використанням зовнішнього AI-модуля. У діаграмі беруть участь чотири елементи: актор «Користувач», учасник «Telegram Bot», учасник «AI Generator», який виступає посередником між ботом і хмарним API OpenAI та база даних «PostgreSQL».

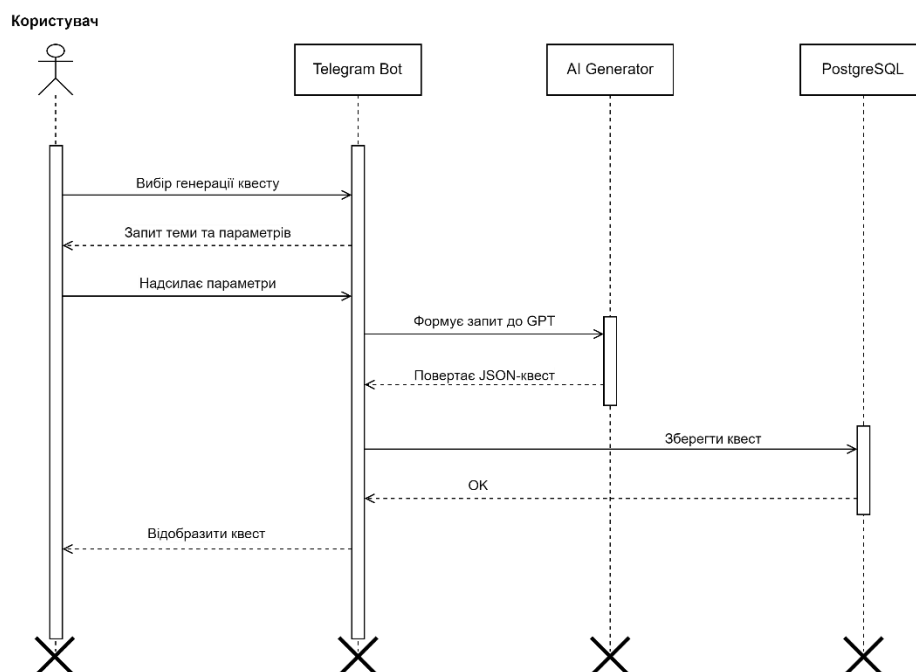


Рисунок 2.13 – Діаграма послідовності генерації квесту засобами штучного інтелекту

Сценарій починається з того, що користувач обирає режим генерації квесту, наприклад, «Квест-вікторина» або «Фізичні квести». Бот у відповідь надсилає запит на введення теми, для вікторини або складності, для фізичних вправ, а також кількості питань/вправ. Користувач надсилає ці параметри.

Отримавши всі необхідні дані, бот формує текстовий промпт і надсилає його до AI-генератора. AI-генератор, у свою чергу, виконує запит до GPT, або іншої мовної моделі, та очікує відповідь. Коли відповідь надходить у форматі JSON, що містить питання, варіанти відповідей та правильну відповідь, або список текстових описів вправ, AI-генератор повертає її боту.

Далі бот зберігає отриманий квест у базі даних. Цей крок передбачає створення запису в таблиці `quests`, а також відповідних записів у `questions` та `answers`, для вікторини, або лише в `questions` для фізичного квесту. База даних підтверджує успішне виконання операції. Після цього бот відображає згенерований квест користувачеві, наприклад, показує перше питання або першу вправу або виводить повідомлення з інформацією про квест і кнопкою «Почати».

2.4.3 Діаграма послідовності проходження квесту

Діаграма послідовності, наведена на рисунку 2.14, відображає сценарій активної фази гри, послідовне виконання запитань квесту з накопиченням балів та фіксацією підсумків. У діаграмі беруть участь три елементи: актор «Користувач», учасник «Telegram Bot» та база даних «PostgreSQL».

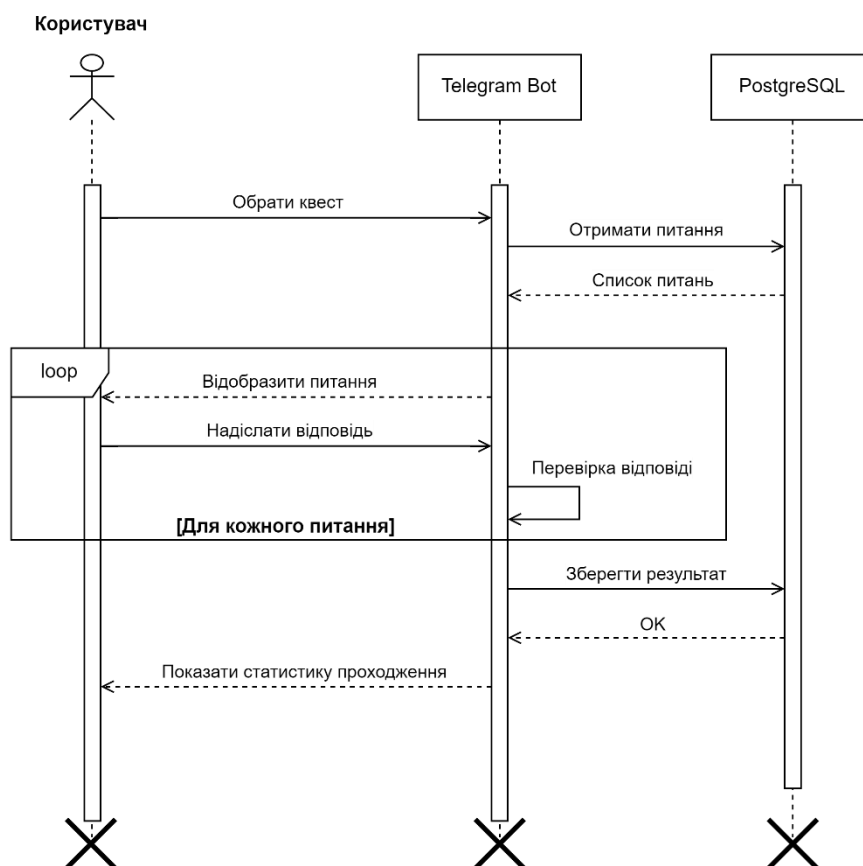


Рисунок 2.14 – Діаграма послідовності проходження квесту

Сценарій починається після того, як користувач обрав конкретний квест, через меню пошуку, список своїх квестів або генерацію. Бот звертається до бази даних із запитом на отримання всіх питань, разом із варіантами відповідей для вікторини або описами вправ для фізичного квесту. База даних повертає впорядкований список питань згідно з полем order.

Далі починається цикл loop Для кожного питання. У межах циклу бот відображає користувачеві поточне питання: текст, бали, а для вікторини кнопки з варіантами. Користувач надсилає відповідь, натискання на варіант

або кнопку «Виконав». Бот виконує перевірку, зіставляє отриману відповідь із правильною для вікторини, порівняння `answer_id` із `is_correct`, для фізичного квесту наявність позначки «Виконав». Результат перевірки впливає на поточний рахунок, нарахування балів або показ правильної відповіді без нарахування, але сам цей крок на діаграмі подано як внутрішню дію бота, самоповідомлення «Перевірка відповіді». Після цього цикл повторюється для наступного питання.

Коли всі питання вичерпано, бот ініціює збереження результату в базі даних. У межах цієї операції оновлюються дві таблиці: `quest_progress`, фіксується фінальний рахунок, позначка `completed = True` та час завершення і `user_statistics`, збільшуються лічильники пройдених квестів, правильних і неправильних відповідей, перераховується середній бал і рівень користувача. База даних підтверджує успішне виконання. Після отримання підтвердження бот відображає користувачеві підсумкову статистику: набрані бали, відсоток правильних відповідей, можливість продовжити або повернутися в меню.

2.4.4 Діаграма послідовності роботи з датасетами

Діаграма послідовності, наведена на рисунку 2.15, ілюструє сценарій обміну квестами між двома користувачами через механізм датасетів. На відміну від попередніх діаграм, тут задіяні два актори: «Користувач 1», автор, який публікує квест та «Користувач 2», той, хто шукає та отримує доступ. Учасниками виступають «Telegram Bot» та база даних «PostgreSQL».

Сценарій складається з двох незалежних частин, які можуть відбуватися в різний час. Перша частина, публікація квесту Користувачем 1. Користувач 1 обирає один зі своїх квестів і натискає кнопку «Опублікувати». Бот отримує цю команду та звертається до бази даних із запитом на створення датасету. У базі даних створюється новий запис у таблиці `datasets`, з автоматично згенерованим унікальним кодом та додається зв'язок між цим датасетом і квестом у таблицю `dataset_quests`. Після успішного виконання операцій база даних повертає боту ідентифікатор створеного датасету (`dataset_id`). Бот

формує повідомлення з кодом доступу наприклад, «A3F9K2» та надсилає його Користувачу 1. На цьому етапі для автора «Користувач 1» завершується.

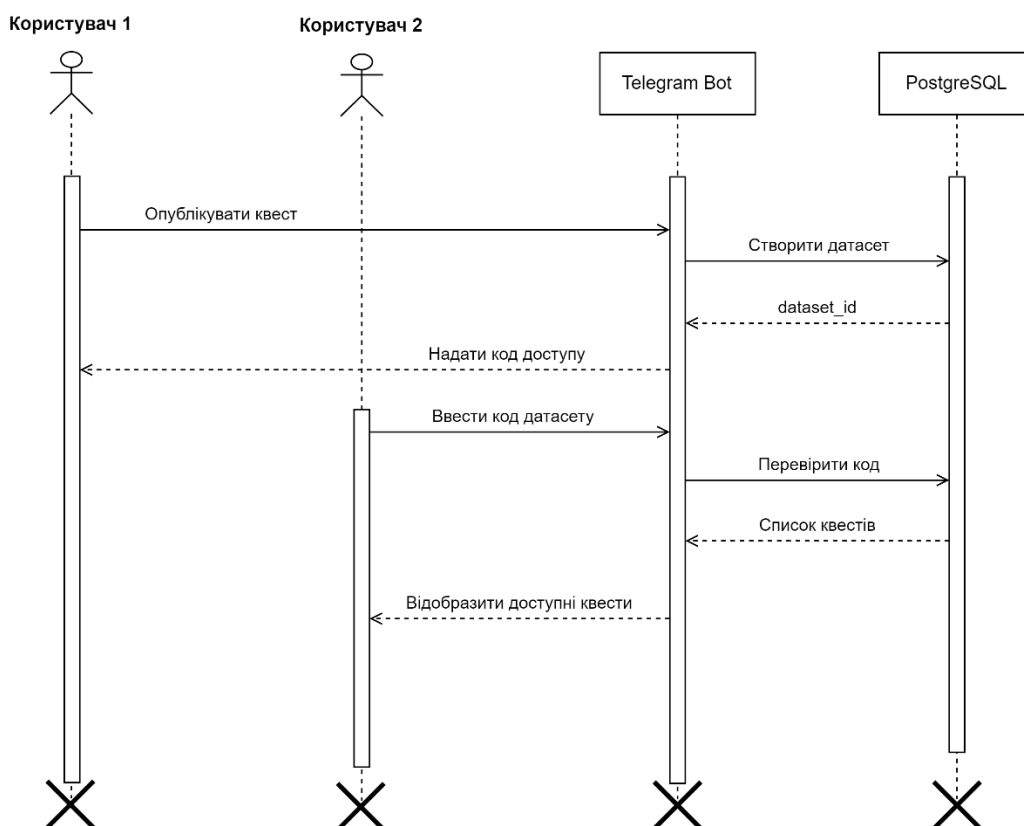


Рисунок 2.15 – Діаграма послідовності роботи з датасетами

Друга частина сценарію починається, коли Користувач 2 отримує цей код, наприклад, від Користувача 1 через будь-який зовнішній канал. Користувач 2 вводить код у чат з ботом після вибору пункту «Знайти квести». Бот виконує запит до бази даних на перевірку коду, пошук датасету за полем `dataset_code`. Якщо код правильний, база даних повертає список квестів, пов'язаних із цим датасетом, через зв'язок `dataset_quests`. Бот отримує цей список і відображає Користувачеві 2 доступні квести у вигляді кнопок або текстового списку. Після цього Користувач 2 може обрати будь-який квест із цього датасету для проходження.

Важливо, що на діаграмі немає прямого зв'язку між двома користувачами, обмін відбувається виключно через бота та базу даних. Такий

підхід унеможливилює пряме спілкування або передачу прав, що спрощує модель безпеки.

2.5 Опис процесу діяльності користувача

Взаємодія користувача з Telegram-ботом реалізована у вигляді послідовності кроків, які виконуються через текстові команди та інтерактивні кнопки. Загальну логіку роботи відображено на блок-схемі рисунку 2.16, яка демонструє основні етапи та можливі варіанти переходів між ними.

Робота з ботом починається з його запуску та натискання кнопки «Почати гру». Система перевіряє, чи існує користувач у базі даних за його Telegram ID. Якщо це перша взаємодія, відбувається автоматична реєстрація: створюються записи в таблицях `users` та `user_statistics`. Зареєстрований користувач одразу потрапляє до головного меню.

Головне меню є центральним вузлом навігації. Через нього доступні такі напрямки: проходження квестів, створення власних квестів, перегляд статистики, робота з датасетами. Кожен напрямок реалізований через окремі обробники повідомлень.

У разі вибору «Почати квест» система пропонує обрати тип: фізичні квести, квест-вікторина, свої квести або пошук за кодом датасету. Для вікторин і фізичних квестів, згенерованих ШІ, користувач додатково вказує кількість завдань, тему або складність. Бот формує запит до AI-модуля, отримує JSON-відповідь або список текстових описів вправ, для фізичного квесту, та одразу розпочинає проходження. Згенерований квест не зберігається в базі даних, а існує лише в межах поточної сесії.

Для готових квестів, створених раніше користувачем або отриманих через датасети, система виконує запит до бази даних, отримує список доступних квестів і відображає його у вигляді кнопок. Після вибору конкретного квесту бот показує його опис та кнопку «Почати». Проходження

включає послідовне відображення питань або вправ, перевірку відповідей, нарахування балів та фіксацію результату.

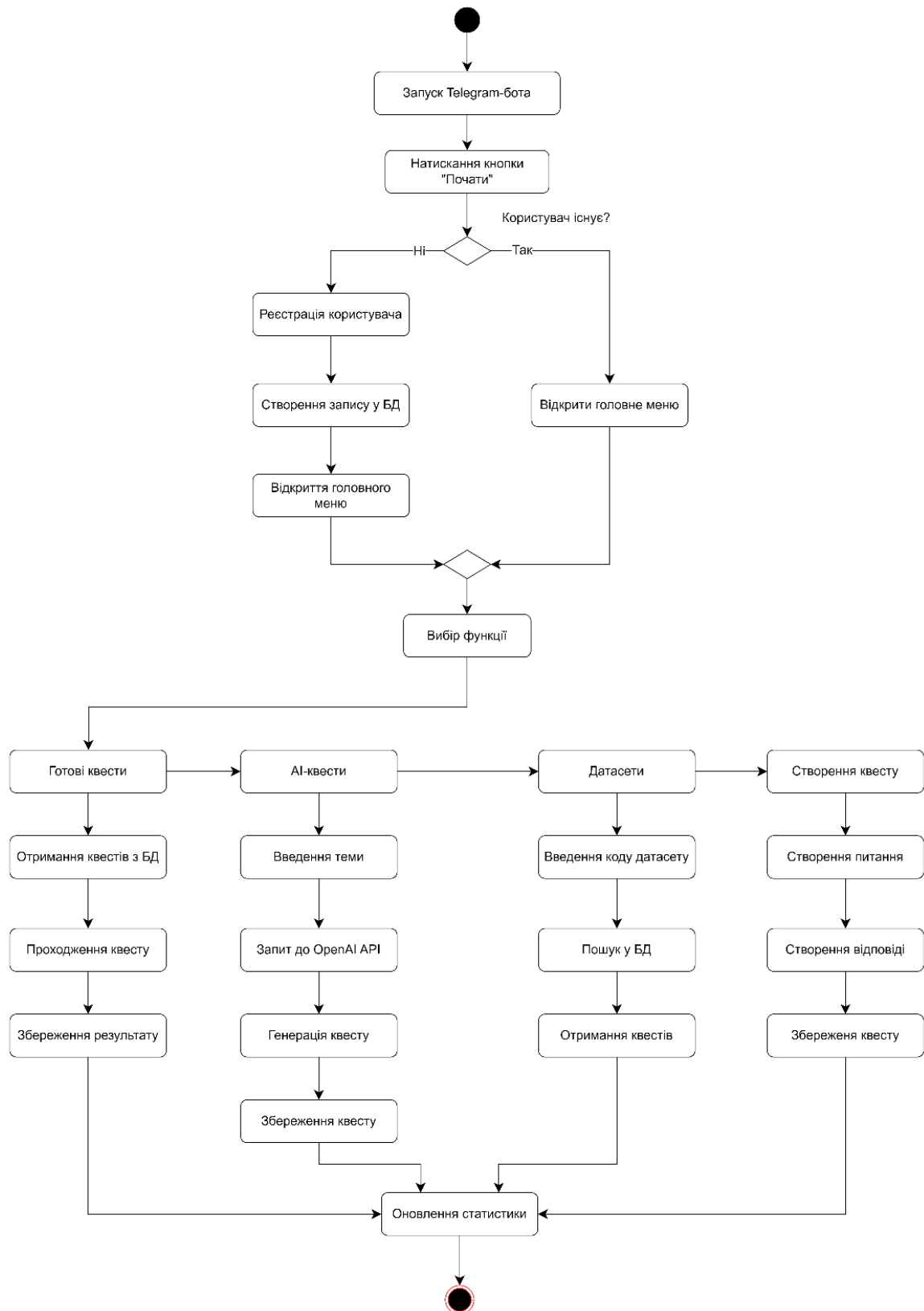


Рисунок 2.16 – Процес діяльності користувача під час роботи з системою

Робота з датасетами передбачає введення спеціального коду. Система шукає датасет у базі даних за цим кодом. Якщо його знайдено, користувач отримує доступ до квестів, включених до датасету, та може розпочати їх проходження.

Створення власного квесту відбувається через конструктор. Користувач послідовно вводить назву, текст питання, чотири варіанти відповідей, вказує правильний варіант та кількість балів. Після завершення всіх питань бот зберігає квест у базі даних. Надалі цей квест доступний для проходження самому автору та для публікації в датасеті.

Після завершення будь-якого квесту система оновлює статистику користувача: кількість пройдених квестів, правильних і неправильних відповідей, середній бал. Якщо досягнуто певної кількості завершених квестів, підвищується рівень користувача.

Окремий сценарій, публікація власного квесту. Автор обирає квест зі свого списку та натискає «Опублікувати». Система створює датасет, генерує унікальний код і повертає його автору. Інші користувачі можуть ввести цей код у розділі «Знайти квести» та отримати доступ до опублікованого квесту.

Описаний процес забезпечує інтуїтивно зрозумілу навігацію між основними функціями, дає змогу користувачеві швидко переходити від одного сценарію до іншого без зайвих дій. Кожна гілка має чітку послідовність кроків і зворотний зв'язок від бота.

Висновки до розділу

У другому розділі виконано аналіз предметної області автоматизованого формування квестів у месенджері Telegram. Визначено основні сутності системи: користувач, квест: вікторина або фізичний, питання, варіант відповіді, прогрес проходження, датасет та статистика. Вхідними даними є дії користувача через інтерфейс бота: вибір типу квесту, введення питань і

відповідей, натискання кнопок, а вихідними – сформовані квести, результати проходження, статистика та коди датасетів для обміну.

Запропоновано загальну концепцію системи, яка поєднує клієнт-серверну архітектуру з використанням Telegram Bot API, бази даних PostgreSQL та зовнішнього AI-модуля для генерації контенту. Розроблено діаграму варіантів використання, діаграму компонентів та діаграму класів, які описують статичну структуру системи. Спроектовано реляційну базу даних з таблицями `users`, `quests`, `questions`, `answers`, `quest_progress`, `datasets` та `user_statistics`. Для кожної таблиці визначено поля, типи даних та зв'язки: один-до-багатьох, багато-до-багатьох, один-до-одного.

У межах алгоритмічного забезпечення розроблено блок-схеми для шести ключових сценаріїв: створення користувацької вікторини, генерації вікторини через ШІ, генерації фізичного квесту, проходження квесту, публікації квесту в датасет та пошуку квестів за кодом датасету. Додатково створено чотири діаграми послідовності, які деталізують динамічну взаємодію між користувачем, ботом, базою даних та AI-модулем під час виконання цих сценаріїв.

Описано процес діяльності користувача, який охоплює всі можливі маршрути навігації: від реєстрації до проходження квестів, створення власних, генерації через ШІ та обміну через датасети. Розроблені моделі та алгоритми стануть основою для програмної реалізації Telegram-бота в третьому розділі.

РОЗДІЛ 3

ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки системи

Для реалізації Telegram-бота обрано мову програмування Python версії 3.10. Цей вибір зумовлений наявністю великої кількості бібліотек для роботи з Telegram API, асинхронного програмування та ORM. Основним фреймворком для створення бота став Aiogram (версія 3.x). Він надає зручні механізми маршрутизації повідомлень, обробки callback-запитів, машини станів FSM та підтримку асинхронності, що критично для роботи з великою кількістю одночасних користувачів.

Як реляційна база даних використовується PostgreSQL. Вона забезпечує надійне зберігання даних, підтримку транзакцій та індексів. Для доступу до бази даних застосовано SQLAlchemy, бібліотеку об'єктно-реляційного відображення. Її асинхронне розширення дозволяє виконувати запити без блокування основного потоку бота. Змінні конфігурації винесено у файл .env, завантаження якого виконується через бібліотеку python-dotenv.

Генерація квестів реалізована з використанням неофіційної бібліотеки g4f (GPT4Free). Вона виступає проміжним шаром для доступу до великих мовних моделей через OpenAI API без необхідності мати власний API-ключ. Хоча в реальному проєкті рекомендовано використовувати офіційний SDK OpenAI, для дипломної роботи такий підхід є прийнятним через відсутність фінансових витрат.

Для роботи з датою та часом використовується вбудований модуль datetime. Усі тимчасові позначки створення квесту, початок проходження, завершення зберігаються в UTC. Випадкові операції: генерація коду датасету, вибір випадкового квесту, виконуються за допомогою стандартної бібліотеки random та string.

3.2 Вимоги до технічного та програмного забезпечення

Для коректної роботи Telegram-бота необхідно забезпечити певні умови з боку сервера, на якому розгорнуто застосунок, а також з боку клієнтського пристрою користувача.

Серверна частина: мінімальні вимоги до апаратного забезпечення сервера:

- процесор: 1 ядро з частотою від 1 ГГц;
- оперативна пам'ять: від 512 МБ (рекомендовано 1 ГБ);
- вільне дискове місце: від 5 ГБ (для зберігання бази даних та логів);
- постійне з'єднання з інтернетом (швидкість від 1 Мбіт/с).

Програмне забезпечення сервера:

- операційна система: Linux (Ubuntu 20.04 або новіше, Debian 11) або Windows Server 2019;
- Python версії 3.10 або вище;
- менеджер пакетів pip;
- PostgreSQL версії 13 або вище;
- бібліотеки Python, перелічені у файлі requirements.txt (ключові: aiogram==3.x, sqlalchemy==2.x, asyncpg, python-dotenv, g4f).

Запуск бота виконується у фоновому режимі, наприклад, через systemd або pm2. Необхідно забезпечити автоматичне перезапускання процесу у разі падіння. Доступ до бази даних має бути налаштований через змінні оточення (файл .env), де зберігаються токен бота, URL підключення до PostgreSQL та інші секретні параметри.

Клієнтська частина: користувачеві для роботи з ботом потрібен лише пристрій із встановленим месенджером Telegram:

- смартфон або планшет на Android (версія 8.0+) чи iOS (версія 13+);
- комп'ютер з Windows, macOS або Linux та настільною версією Telegram;

– стабільне підключення до інтернету (для обміну повідомленнями достатньо 2G/3G або Wi-Fi).

Жодних додаткових застосунків, драйверів або розширень встановлювати не потрібно. Весь функціонал реалізовано безпосередньо в чаті з ботом.

Якщо використовується проксі-сервер або фаєрвол, необхідно відкрити вихідні TCP-з'єднання на порти 80 та 443 для HTTPS-запитів. Вхідні порти відкривати не потрібно, бот працює в режимі long polling, тобто сам ініціює з'єднання з Telegram.

Дотримання цих вимог забезпечує стабільну роботу системи в режимі 24/7 без втручання адміністратора.

3.3 Опис програмної реалізації

Програмний код Telegram-бота організований у вигляді модульної структури, яка відокремлює логіку роботи з базою даних, обробку повідомлень, клавіатури та стани. Коренева директорія проєкту містить файл .env для змінних оточення, requirements.txt з переліком залежностей та основну папку app. Дерево файлів проєкту наведено на рис. 3.1.

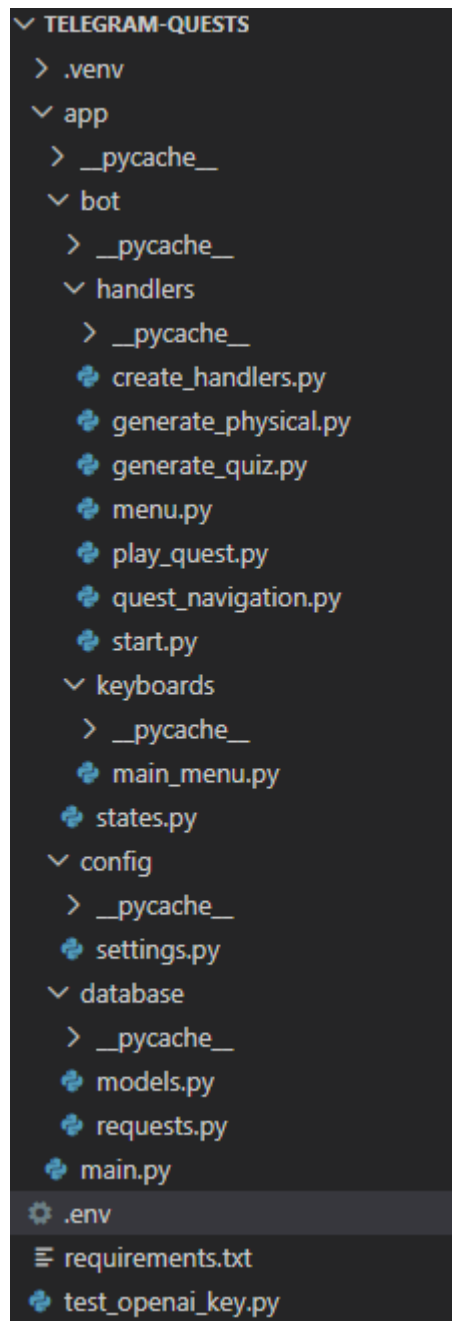


Рисунок 3.1 – Структура файлів проєкту

У папці `app/database` розміщено файли `models.py`, ORM-моделі таблиць, `requests.py`, функції для роботи з базою даних та `main.py`, точка входу для створення таблиць. Папка `app/bot` містить піддиректорії `handlers`, обробники повідомлень, `keyboards`, клавіатури та файл `states.py`, класи станів FSM. Кожен обробник відповідає за окремий сценарій:

- `start.py` – реєстрація та головне меню;

- `quest_navigation.py` – навігація між типами квестів і робота з датасетами;
- `create_handlers.py` – створення власних квестів;
- `play_quest.py` – проходження квестів;
- `generate_quiz.py` та `generate_physical.py` – генерація через ШІ;
- `menu.py` – профіль користувача.

3.3.1 Реалізація механізму реєстрації користувачів

Реєстрація користувача відбувається автоматично під час першого запуску бота. Для цього використовується обробник команди `/start`, який викликає функцію `get_or_create_user`. Вона перевіряє, чи існує в базі даних запис із переданим `telegram_id`. Якщо ні, створюється новий користувач у таблиці `users` та порожній запис статистики в таблиці `user_statistics`. Такий підхід позбавляє користувача необхідності вводити додаткові дані вручну.

Реалізація реєстрації виконана у файлі `requests.py`. Код містить три основні функції: `get_user_by_telegram`, `create_user` та `get_or_create_user`. Приклад наведено в лістингу 3.1.

Лістинг 3.1 – Функції для роботи з користувачами в базі даних

```

async def get_user_by_telegram(telegram_id: int):
    async with async_session() as session:
        query = select(User).where(User.telegram_id ==
telegram_id)
        result = await session.execute(query)
        return result.scalar_one_or_none()
async def create_user(telegram_id: int, username: str = None,
first_name: str = "User"):
    async with async_session() as session:
        user = User(telegram_id=telegram_id, username=username,
first_name=first_name)
        session.add(user)
        await session.commit()
        stat = UserStatistic(user_id=user.id)
        session.add(stat)
        await session.commit()
        return user
async def get_or_create_user(telegram_id: int, username: str =
None, first_name: str = "User"):
    user = await get_user_by_telegram(telegram_id)
    if not user:

```

```

        user = await create_user(telegram_id, username,
first_name)
    return user

```

Функція `get_user_by_telegram` виконує запит до таблиці `users` за полем `telegram_id`. Вона повертає об'єкт користувача або `None`, якщо запис не знайдено. У функції `create_user` спочатку створюється новий запис у таблиці `users`, а потім одразу пов'язаний запис у таблиці `user_statistics` з нульовими показниками. Обидві операції виконуються в окремих транзакціях із викликом `commit()`. Функція `get_or_create_user` поєднує попередні дві: спочатку намагається знайти користувача, а в разі невдачі створює нового.

У головному файлі `start.py` виклик реєстрації відбувається після натискання кнопки «Почати гру», лістинг 3.2 відповідаючи за цей процес наведено нижче

Лістинг 3.2 – Функція для реєстрації у базі

```

@router.message(F.text == '🎮 Почати гру')
async def start_game(message: Message):
    await get_or_create_user(
        telegram_id=message.from_user.id,
        username=message.from_user.username,
        first_name=message.from_user.first_name)
    await message.answer('✅ Ласкаво просимо! Обери дію:',
reply_markup=main_keyboard)

```

Таким чином, реєстрація є повністю прозорою для користувача. Після першого натискання кнопки система автоматично створює всі необхідні записи, і користувач одразу отримує доступ до головного меню.

3.3.2 Реалізація функціоналу створення квестів

Створення квестів реалізоване через механізм станів: FSM який надає бібліотека `Aiogram`. Це дозволяє розбити процес на послідовні кроки, де бот очікує введення певного типу даних: назва, питання, варіанти відповідей тощо. Уся логіка зосереджена у файлі `create_handlers.py`, а стани описані в окремому файлі `states.py`.

Визначення станів для створення квесту наведено в лістингу 3.2.

Лістинг 3.2 – Клас станів для створення квесту.

```
class CreateQuest(StatesGroup):
    waiting_for_title = State()
    waiting_for_physical_description = State()
    waiting_for_physical_points = State()
    waiting_for_question_text = State()
    waiting_for_answers = State()
    waiting_for_correct_answer = State()
    waiting_for_points_per_question = State()
    deciding = State()
```

Кожен стан відповідає одному кроку в діалозі: спочатку бот очікує назву, потім текст питання, варіанти відповідей, вибір правильної відповіді, кількість балів тощо. Окремий стан `deciding` використовується після додавання кожного питання, щоб запитати користувача, чи хоче він додати ще одне питання або завершити створення.

Процес починається після вибору користувачем пункту «Створити вікторину». Бот переходить у стан `waiting_for_title` та запитує назву майбутнього квесту. Обробник цього стану наведено в лістингу 3.3.

Лістинг 3.3 – Обробник введення назви квесту

```
@router.message(F.text == '📝 Створити вікторину')
async def create_quiz(message: Message, state: FSMContext):
    await state.set_state(CreateQuest.waiting_for_title)
    await state.update_data(quest_type='quiz', questions=[])
    await message.answer("Придумайте назву для квесту-вікторини:")
@router.message(CreateQuest.waiting_for_title)
async def get_title(message: Message, state: FSMContext):
    title = message.text.strip()
    await state.update_data(title=title)
    await state.set_state(CreateQuest.waiting_for_question_text)
    await message.answer("Введіть текст першого питання:")
```

Після отримання назви бот зберігає її в контексті стану, через `state.update_data` та переходить до стану очікування тексту першого питання. Усі введені дані накопичуються в пам'яті до моменту фінального збереження.

Цикл додавання питань реалізований через послідовну зміну станів. Для кожного питання користувач вводить текст, чотири варіанти відповідей, кожен з нового рядка, номер правильної відповіді (1–4) та кількість балів. Після цього бот знову повертається до стану `deciding` та пропонує додати ще одне питання або завершити створення. Обробник збереження наведено в лістингу 3.4.

Лістинг 3.4 – Завершення створення квесту та збереження в базі даних

```
@router.callback_query(lambda c: c.data == "finish_creation")
async def finish_creation_callback(callback: CallbackQuery,
state: FSMContext):
    data = await state.get_data()
    quest_type = data.get('quest_type')
    title = data.get('title')
    user = await get_user_by_telegram(callback.from_user.id)
    total_score = sum(q.get('points', 0) for q in
data.get('questions', []))
    quest_id = await create_quest(
        creator_id=user.id,
        title=title,
        description="Користувацька вікторина",
        theme="Користувацький",
        quest_type=quest_type,
        total_score=total_score,
        difficulty="user")
    for idx, q in enumerate(data.get('questions', [])):
        q_id = await add_question_to_quest(quest_id, q['text'],
idx, points=q['points'], qtype="single")
        for ans_idx, ans_text in enumerate(q['answers']):
            is_correct = (ans_idx == q['correct'])
            await add_answer_to_question(q_id, ans_text,
is_correct)
        await callback.message.answer(f"Квест «{title}» успішно
створено!")
    await state.clear()
```

Спочатку функція отримує всі накопичені дані з контексту стану. Обчислюється загальна сума балів. Виклик `create_quest` створює запис у таблиці `quests` і повертає `quest_id`. Далі для кожного питання виконується `add_question_to_quest`, яка додає запис до таблиці `questions`, а для кожного варіанта відповіді, `add_answer_to_question` до таблиці `answers` з позначкою `is_correct` для правильної відповіді. Після цього контекст стану очищується, а користувач отримує підтвердження.

Функції роботи з базою даних, `create_quest`, `add_question_to_quest`, `add_answer_to_quest`, реалізовані у файлі `requests.py` і використовують асинхронні запити через `SQLAlchemy`, лістинги 2.3, 2.4, 2.5 у підрозділі 2.2.5. Вони забезпечують транзакційне збереження всіх елементів квесту.

Створення фізичного квесту виконується за аналогічною схемою, але замість питань із варіантами відповідей користувач вводить лише опис завдання та кількість балів. Для фізичних квестів у таблиці `questions` створюється запис із типом `physical`, а в таблиці `answers` додається одна фіктивна правильна відповідь «Виконано».

3.3.3 Реалізація генерації квестів засобами штучного інтелекту

Генерація квестів із використанням мовних моделей реалізована для двох типів: вікторини (`quiz`) та фізичні квести. Обидва сценарії побудовані за схожою схемою: користувач обирає кількість завдань, тему або складність, після чого бот формує промпт, надсилає запит до AI-модуля, обробляє відповідь і одразу розпочинає проходження. Уся логіка зосереджена у файлах `generate_quiz.py` та `generate_physical.py`, а стани описані в `states.py`.

Після вибору режиму «Квест-вікторина» користувач послідовно обирає кількість питань: 10, 20, 30 або 40 та тему історія, українська література, філософія, хімія, фізика або випадкова тема. Бот формує промпт, який містить чітку інструкцію: згенерувати задану кількість питань, кожне з чотирма варіантами відповідей, лише одна правильна. Результат має бути поданий у вигляді масиву JSON-об'єктів із ключами `question`, `options` та `correct`. Реалізацію формування запиту та виклику AI наведено в лістингу 3.5.

Лістинг 3.5 – Функція генерації вікторини через AI

```
async def generate_quiz(message: Message, state: FSMContext,
count: int, topic_name: str, topic_code: str):
    prompt = f"""Згенеруй {count} тестових питань на тему
"{topic_name}". Кожне питання має 4 варіанти відповіді (лише
одна правильна). Представ результат у вигляді списку JSON
об'єктів з ключами: "question", "options" (масив з 4 рядків),
"correct" (індекс правильної відповіді, 0-3). Не додавай жодних
зайвих слів, тільки JSON масив."""
```

```

try:
    response = await g4f.ChatCompletion.create_async(
        model=g4f.models.default,
        messages=[{"role": "user", "content": prompt}],)
    text = response.strip()
    if text.startswith("` `` `json"):
        text = text[7:]
    if text.startswith("` `` `"):
        text = text[3:]
    if text.endswith("` `` `"):
        text = text[:-3]
    questions_data = json.loads(text)
    await state.update_data(questions=questions_data,
current_index=0, score=0)
    await state.set_state(GenerateQuiz.playing)
    await show_next_quiz_question(message, state)
except Exception as e:
    await message.answer(f"Помилка генерації: {str(e)}")
    await state.clear()

```

Промпт передається до бібліотеки g4f, яка виступає проміжним шаром для доступу до OpenAI API. Отриманий текст очищується від маркерів коду, після чого виконується парсинг JSON. Якщо структура відповідає очікуваній, масив питань зберігається в контексті FSM, і бот переходить у стан playing для негайного проходження. У разі помилки користувач отримує повідомлення, а стан очищується.

Процес є аналогічним, але замість JSON бот очікує простий текстовий список вправ. Користувач обирає кількість вправ: 5, 10, 15 або 20 та рівень складності легкий, середній, важкий. Складність впливає як на формулювання промπτу, так і на кількість балів за виконану вправу: 5, 10 або 15 балів відповідно. Реалізацію наведено в лістингу 3.6.

Лістинг 3.6 – Функція генерації фізичних вправ через AI

```

async def generate_exercises(message: Message, state:
FSMContext, count: int, diff: dict):
    prompt = f"Згенеруй {count} різноманітних фізичних вправ
{diff['prompt_suffix']}. Кожна вправа має бути коротким описом
дії. Виведи кожен вправу з нового рядка. Не використовуй
нумерацію, тільки текст вправ."
    try:
        response = await g4f.ChatCompletion.create_async(
            model=g4f.models.default,

```

```

        messages=[{"role": "user", "content": prompt}],)
    text = response.strip()
    exercises = [line.strip() for line in text.split('\n')
if line.strip()]
    if len(exercises) < count:
        exercises.extend([f"Вправа {i+1}" for i in
range(count - len(exercises))])
    exercises = exercises[:count]
    await state.update_data(exercises=exercises,
current_index=0, score=0)
    await state.set_state(GeneratePhysical.playing)
    await show_next_exercise(message, state)
except Exception as e:
    await message.answer(f"Помилка генерації: {str(e)}")
    await state.clear()

```

Отриманий текст розбивається на рядки, кожен непорожній рядок вважається окремою вправою. Якщо кількість отриманих вправ менша за замовлену, додаються стандартні вправи-заповнювачі, якщо більша, список обрізається. Згенерований масив зберігається в контексті FSM разом із поточним рахунком та індексом поточної вправи.

Згенеровані квести не зберігаються в базі даних. Вони існують лише протягом однієї сесії проходження. Це зроблено навмисно, щоб кожне звернення користувача до генерації давало унікальний набір завдань. За потреби користувач може повторно згенерувати квест із тими самими параметрами, щоразу результат буде різним. Такий підхід підвищує варіативність і запобігає завчанню відповідей.

Обробка помилок передбачає перехоплення винятків: мережеві помилки, тайм-аути, некоректний JSON або текст. У всіх випадках користувач отримує зрозуміле повідомлення, а стан генерації очищується, що дозволяє спробувати знову з головного меню.

3.3.4 Реалізація механізму проходження квестів

Проходження квесту є основним ігровим циклом системи. Незалежно від типу квесту, вікторина чи фізичний, логіка складається з послідовного відображення завдань, отримання відповідей, нарахування балів та фіксації результату. Реалізація виконана у файлі `play_quest.py`, а для згенерованих ІІІ

квестів у файлах `generate_quiz.py` та `generate_physical.py`. У всіх випадках використовується механізм FSM для збереження поточного стану проходження.

Після вибору квесту, зі списку, через датасет або генерації, бот ініціалізує дані сесії: `quest_id`, `quest_type`, `total_score`, `current_question_index`, `score` та список питань. Далі викликається функція `send_current_question`, яка відображає поточне завдання залежно від типу. Цикл триває, поки індекс поточного питання не досягне кінця списку. Після цього виконується завершення квесту зі збереженням результатів.

Для вікторини кожне питання супроводжується чотирма `inline`-кнопками з варіантами відповідей. Приклад реалізації відображення питання наведено в лістингу 3.7.

Лістинг 3.7 – Відображення питання вікторини.

```
async def send_current_question(message: Message, state:
FSMContext):
    data = await state.get_data()
    idx = data['current_question_index']
    questions = data['questions']
    if idx >= len(questions):
        await finish_quest(message, state)
        return
    if data['quest_type'] == 'quiz':
        question, answers = questions[idx]
        text = f"Питання
{idx+1}/{len(questions)}\n{question.question_text}\nБалів:
{question.points}"
        buttons = [[InlineKeyboardButton(text=a.answer_text,
callback_data=f"ans_{a.id}")] for a in answers]
        buttons.append([InlineKeyboardButton(text="Завершити
квест", callback_data="cancel_quest")])
        markup = InlineKeyboardMarkup(inline_keyboard=buttons)
        await message.answer(text, reply_markup=markup)
```

Після натискання кнопки спрацьовує обробник `process_quiz_answer`, лістинг 3.8. Він визначає, чи збігається вибрана відповідь із правильною `is_correct`. Якщо так, додає бали до поточного рахунку. Якщо ні, показує

правильну відповідь, але бали не нараховує. Потім індекс питання збільшується, і бот переходить до наступного.

Лістинг 3.8 – Обробка відповіді у вікторині.

```
@router.callback_query(PlayQuest.answering, lambda c:
c.data.startswith("ans_"))
async def process_quiz_answer(callback: CallbackQuery, state:
FSMContext):
    answer_id = int(callback.data.split("_")[1])
    data = await state.get_data()
    idx = data['current_question_index']
    question, answers = data['questions'][idx]
    selected = next((a for a in answers if a.id == answer_id),
None)
    if selected and selected.is_correct:
        new_score = data['score'] + question.points
        await state.update_data(score=new_score)
        await callback.answer("Правильно!")
    else:
        correct = next((a for a in answers if a.is_correct),
None)
        await callback.answer(f"Неправильно. Правильна
відповідь: {correct.answer_text}", show_alert=True)
        await state.update_data(current_question_index=idx + 1)
        await send_current_question(callback.message, state)
```

Для фізичних квестів замість варіантів відповідей використовуються дві кнопки: «Виконав» та «Пропустити». При натисканні «Виконав» нараховуються бали за поточну вправу; при натисканні «Пропустити», бали не нараховуються. В обох випадках відбувається перехід до наступної вправи. Реалізація аналогічна до вікторини, але без перевірки правильності.

Коли всі питання вичерпано, викликається функція `finish_quest`, лістинг 3.9. Вона оновлює запис прогресу в таблиці `quest_progress`, фіксує набраний рахунок, позначку `completed = True` та час завершення, а також оновлює статистику користувача через `update_user_statistics`, збільшує лічильники пройдених квестів, правильних відповідей, перераховує середній бал та рівень.

Лістинг 3.9 – Завершення квесту

```
async def finish_quest(message: Message, state: FSMContext):
    data = await state.get_data()
```

```

quest_id = data['quest_id']
score = data['score']
total = data['total_score']
user = await get_user_by_telegram(message.chat.id)
if user:
    await update_progress(user.id, quest_id, score,
completed=True)
    await update_user_statistics(user.telegram_id,
correct=score, wrong=total-score, completed_quest=True)
    await message.answer(f"Квест завершено! Ваш результат:
{score} / {total} балів.")
    await state.set_state(PlayQuest.finished)

```

Після завершення користувачеві пропонується продовжити або повернутися в головне меню. Якщо обрано продовження, система випадковим чином вибирає наступний квест із доступних і запускає його без додаткових дій користувача.

Під час проходження передбачено кнопку «Завершити квест». При її натисканні стан очищується, прогрес не зберігається, а користувач повертається в головне меню. Це дозволяє перервати квест без штрафу.

Таким чином, механізм проходження забезпечує покрокову взаємодію, зберігає проміжний стан через FSM та фіксує результати в базі даних після завершення. Окремі реалізації для вікторин та фізичних квестів враховують специфіку перевірки відповідей та нарахування балів.

3.3.5 Реалізація механізму роботи з датасетами

Механізм датасетів забезпечує обмін квестами між користувачами. Датасет – це набір квестів, об’єднаних спільним унікальним кодом. Автор публікує свій квест, отримує код, а інші користувачі вводять цей код у розділі «Знайти квести» та отримують доступ до проходження. Реалізація складається з функцій роботи з базою даних та обробників повідомлень.

Основні операції створення, пошуку та модифікації датасетів наведено в лістингу 3.10.

Лістинг 3.10 – Функції роботи з датасетами в базі даних

```

def generate_dataset_code(length=6):
    chars = string.ascii_uppercase + string.digits

```

```

    return ''.join(random.choices(chars, k=length))

async def create_dataset(creator_id: int, title: str,
description: str) -> Dataset:
    async with async_session() as session:
        code = generate_dataset_code()
        while True:
            existing = await
session.execute(select(Dataset).where(Dataset.dataset_code ==
code))
            if not existing.scalar_one_or_none():
                break
            code = generate_dataset_code()
            dataset = Dataset(title=title, description=description,
dataset_code=code, creator_id=creator_id)
            session.add(dataset)
            await session.commit()
            return dataset

async def add_quest_to_dataset(dataset_id: int, quest_id: int):
    async with async_session() as session:
        await
session.execute(dataset_requests.insert().values(dataset_id=dataset_id,
quest_id=quest_id))
        await session.commit()

async def get_dataset_by_code(code: str):
    async with async_session() as session:
        result = await
session.execute(select(Dataset).where(Dataset.dataset_code ==
code))
        return result.scalar_one_or_none()

async def get_requests_from_dataset(dataset_id: int):
    async with async_session() as session:
        dataset = await session.get(Dataset, dataset_id)
        if not dataset:
            return []
        await session.refresh(dataset,
attribute_names=["requests"])
        return list(dataset.requests)

async def is_quest_published(quest_id: int) -> bool:
    async with async_session() as session:
        result = await
session.execute(select(dataset_requests).where(dataset_requests.c.quest_id == quest_id))
        return result.first() is not None

async def remove_quest_from_dataset(quest_id: int):
    async with async_session() as session:
        await
session.execute(dataset_requests.delete().where(dataset_requests.c.quest_id == quest_id))

```

```

quest_id == quest_id))
    # якщо датасет залишився порожнім - видалити його
    dataset = await get_dataset_by_quest(quest_id)
    if dataset:
        result = await
session.execute(select(dataset_quests).where(dataset_quests.c.da
taset_id == dataset.id))
        if result.first() is None:
            await session.delete(dataset)
        await session.commit()

```

Функція `generate_dataset_code` створює випадковий рядок із шести символів. `create_dataset` перевіряє унікальність коду в циклі, потім додає запис до таблиці `datasets`. `add_quest_to_dataset` вставляє зв'язок у допоміжну таблицю `dataset_quests`. Для пошуку датасету за кодом використовується `get_dataset_by_code`, а для отримання списку квестів, `get_quests_from_dataset`. `is_quest_published` перевіряє, чи існує зв'язок для заданого квесту. `remove_quest_from_dataset` видаляє зв'язок, а якщо датасет стає порожнім, видаляє і сам датасет.

Коли автор обирає квест із власного списку та натискає «Опублікувати», спрацьовує обробник, наведений у лістингу 3.11.

Лістинг 3.11 – Обробник публікації квесту

```

@router.callback_query(lambda c:
c.data.startswith("publish_quest_"))
async def publish_selected_quest(callback: CallbackQuery):
    quest_id = int(callback.data.split("_")[2])
    user = await get_user_by_telegram(callback.from_user.id)
    quest = await get_quest_by_id(quest_id)
    if not quest or quest.creator_id != user.id:
        await callback.answer("Цей квест не належить вам.",
show_alert=True)
        return
    if await is_quest_published(quest_id):
        await callback.answer("Квест вже опубліковано!",
show_alert=True)
        return
    dataset = await create_dataset(user.id, title=f"Квест:
{quest.title}", description=f"Містить квест «{quest.title}»")
    await add_quest_to_dataset(dataset.id, quest_id)
    await callback.message.edit_text(
        f"Квест опубліковано! Код датасету:
<code>{dataset.dataset_code}</code>\n"

```

```

        f"Поділіться цим кодом з іншими, щоб вони могли додати  
ваш квест.",
        parse_mode="HTML"
    )

```

Спочатку перевіряється право власності та чи не опубліковано квест вже. Потім створюється датасет із назвою «Квест: [назва]», додається зв'язок, і код повертається автору.

Користувач обирає пункт «Знайти квести», після чого бот переходить у стан очікування коду. Обробник введення коду наведено в лістингу 3.12.

Лістинг 3.12 – Обробник пошуку датасету за кодом

```

@router.message(F.text == '🔍 Знайти квести')
async def find_requests(message: Message, state: FSMContext):
    await state.set_state("waiting_dataset_code")
    await message.answer("Введіть код датасету (наприклад,  
A3F9K2):")
@router.message(StateFilter("waiting_dataset_code"))
async def process_dataset_code(message: Message, state:
FSMContext):
    code = message.text.strip().upper()
    dataset = await get_dataset_by_code(code)
    if not dataset:
        await message.answer("Датасет з таким кодом не  
знайдено.")
        await state.clear()
        return
    requests = await get_requests_from_dataset(dataset.id)
    if not requests:
        await message.answer("У цьому датасеті немає квестів.")
        await state.clear()
        return
    buttons = [[InlineKeyboardButton(text=q.title,
callback_data=f"select_quest_{q.id}")] for q in requests]
    markup = InlineKeyboardMarkup(inline_keyboard=buttons)
    await message.answer(f"Знайдено датасет «{dataset.title}».  
Оберіть квест:", reply_markup=markup)
    await state.clear()

```

Код перетворюється у верхній регістр і виконується пошук. Якщо датасет знайдено, користувач отримує список квестів у вигляді кнопок. Вибір квесту перенаправляє до стандартного сценарію проходження.

Автор може будь-коли приховати квест. Обробник `unpublish_quest_` викликає `remove_quest_from_dataset` і повідомляє про успішне виконання. Якщо датасет після видалення єдиного квесту стає порожнім, він автоматично видалюється.

Таким чином, механізм датасетів повністю забезпечує життєвий цикл публікації, від створення та отримання коду до пошуку, проходження та зняття з публікації. Усі операції виконуються транзакційно, використовуються індекси для прискореного пошуку за кодом.

3.3.6 Реалізація системи збереження статистики користувачів

Система збереження статистики відстежує ефективність кожного користувача: кількість пройдених квестів, правильних і неправильних відповідей, середній бал та рівень. Дані зберігаються у двох таблицях: `users`, поля `level` та `exp` та `user_statistics`, усі інші показники. Оновлення статистики відбувається щоразу після завершення квесту. Реалізація виконана у файлі `requests.py`. Структура таблиці `user_statistics` описана в лістингу 3.13.

Лістинг 3.13 – Модель статистики користувача

```
class UserStatistic(Base):
    __tablename__ = "user_statistics"
    id: Mapped[int] = mapped_column(primary_key=True)
    user_id: Mapped[int] = mapped_column(ForeignKey("users.id"),
unique=True)
    total_questions: Mapped[int] = mapped_column(Integer,
default=0)
    completed_questions: Mapped[int] = mapped_column(Integer,
default=0)
    correct_answers: Mapped[int] = mapped_column(Integer,
default=0)
    wrong_answers: Mapped[int] = mapped_column(Integer,
default=0)
    average_score: Mapped[float] = mapped_column(Float,
default=0)

    user = relationship("User", back_populates="statistics")
```

Поле `user_id` має обмеження `unique=True`, тому кожен користувач може мати лише один запис статистики. Усі числові поля ініціалізуються нулями, `average_score` – нулем як `float`.

При реєстрації користувача автоматично створюється порожній запис статистики. Функція `get_user_statistics_by_user_db_id`, лістинг 3.14 повертає існуючий запис або створює новий, якщо його чомусь немає.

Лістинг 3.14 – Функція отримання статистики

```
async def get_user_statistics_by_user_db_id(user_db_id: int):
    async with async_session() as session:
        stat = await
session.execute(select(UserStatistic).where(UserStatistic.user_id == user_db_id))
        stat = stat.scalar_one_or_none()
        if not stat:
            stat = UserStatistic(user_id=user_db_id)
            session.add(stat)
            await session.commit()
        return stat
```

Основна функція оновлення `update_user_statistics` приймає Telegram ID користувача, кількість правильних і неправильних відповідей та ознаку завершення квесту. Її реалізацію наведено в лістингу 3.15.

Лістинг 3.15 – Функція оновлення статистики

```
async def update_user_statistics(user_id: int, correct: int = 0,
wrong: int = 0, completed_quest: bool = False):
    async with async_session() as session:
        user_db = await get_user_by_telegram(user_id)
        if not user_db:
            return
        stat = await
get_user_statistics_by_user_db_id(user_db.id)
        stat.correct_answers += correct
        stat.wrong_answers += wrong
        if completed_quest:
            stat.completed_quests += 1
            stat.total_quests += 1
            total_answered = stat.correct_answers +
stat.wrong_answers
            if total_answered > 0:
                stat.average_score = (stat.correct_answers /
```

```
total_answered) * 100
    # оновлення рівня користувача
    new_level = stat.completed_quests // 5 + 1
    if new_level > user_db.level:
        user_db.level = new_level
    await session.commit()
```

Спочатку функція знаходить користувача за `telegram_id`, потім отримує або створює запис статистики. Додає передані кількості правильних і неправильних відповідей. Якщо квест завершено успішно, збільшує лічильник `completed_quests`. Лічильник `total_quests` збільшується завжди, навіть якщо квест не завершено. Середній бал перераховується як відсоток правильних відповідей.

Рівень користувача обчислюється за формулою: `completed_quests // 5 + 1`. Тобто перші п'ять завершених квестів дають перший рівень, наступні п'ять, другий і так далі. Якщо новий рівень перевищує поточний, поле `level` у таблиці `users` оновлюється.

У головному меню після натискання кнопки «Мої данні» викликається обробник, який отримує дані з таблиць `users` та `user_statistics` і форматує їх у вигляді повідомлення (лістинг 3.16).

Лістинг 3.16 – Відображення статистики

```
@router.message(F.text == '📊 Мої данні')
async def show_profile(message: Message):
    user = await get_user_by_telegram(message.from_user.id)
    stat = await get_user_statistics_by_user_db_id(user.id)
    text = (
        f"Рівень: {user.level}\n"
        f"Пройдено квестів: {stat.completed_quests}\n"
        f"Всього спроб: {stat.total_quests}\n"
        f"Правильних відповідей: {stat.correct_answers}\n"
        f"Неправильних: {stat.wrong_answers}\n"
        f"Середній бал: {stat.average_score:.1f}%"
    )
    await message.answer(text)
```

Таким чином, статистика оновлюється автоматично після кожного завершеного квесту та зберігається в базі даних. Користувач може переглянути

її в будь-який момент. Система не зберігає деталізовану історію кожної відповіді, що достатньо для відстеження загального прогресу.

3.4 Тестування системи, перевірка коректності роботи та аналіз результатів

На першому етапі роботи з ботом потрібно його активувати. Для цього при відкритті бота треба натиснути кнопку «Старт». У такому випадку бот реєструє користувача в базі даних, оновлює панель кнопок на головне меню з певним функціоналом, рисунок 3.2.



Рисунок 3.2 – Початок роботи з ботом та перегляд оновленої клавіатури

Після реєстрації бот надає можливість переглянути статистику: рівень акаунту, кількість пройдених квестів, загальну кількість спроб, правильних і неправильних відповідей, середній бал. Далі можна переглянути квести, згенеровані автоматично за допомогою штучного інтелекту, розпочати проходження або скасувати квест без накопичення балів, рисунок 3.3.

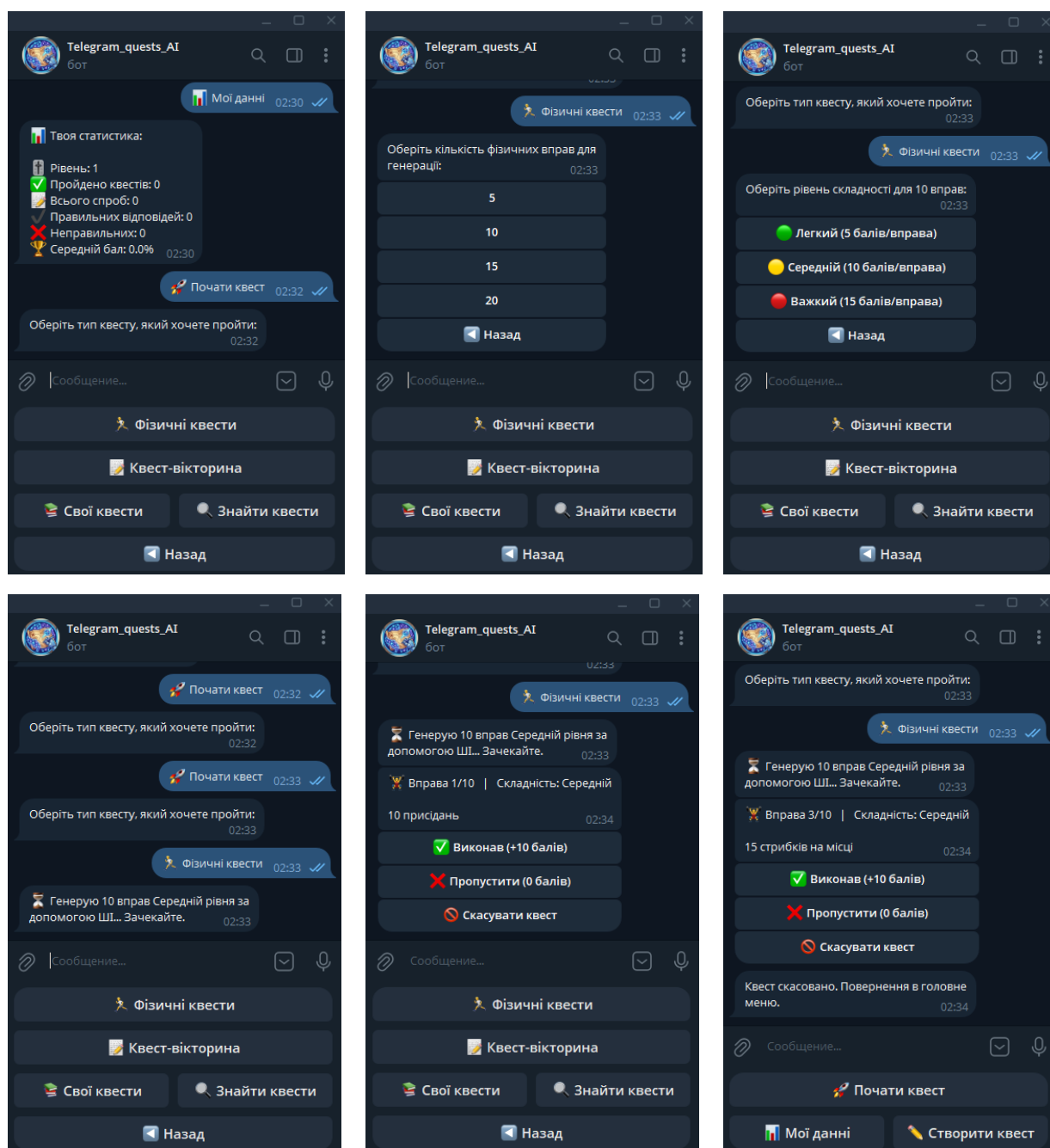


Рисунок 3.3 – Перегляд інформації, початок перегляду згенерованих квестів, проходження квесту та скасування

При виборі «Квест-вікторина» користувачеві треба обрати кількість питань та тему: історія, українська література, філософія, хімія, фізика або випадкові квести. Далі відбувається генерація та показ питань. На відміну від фізичних квестів, у вікторині є 4 варіанти відповіді. У разі неправильно обраної відповіді з'являється вікно з інформацією про правильний варіант. Після завершення вікторини підраховуються бали, рисунок 3.4.

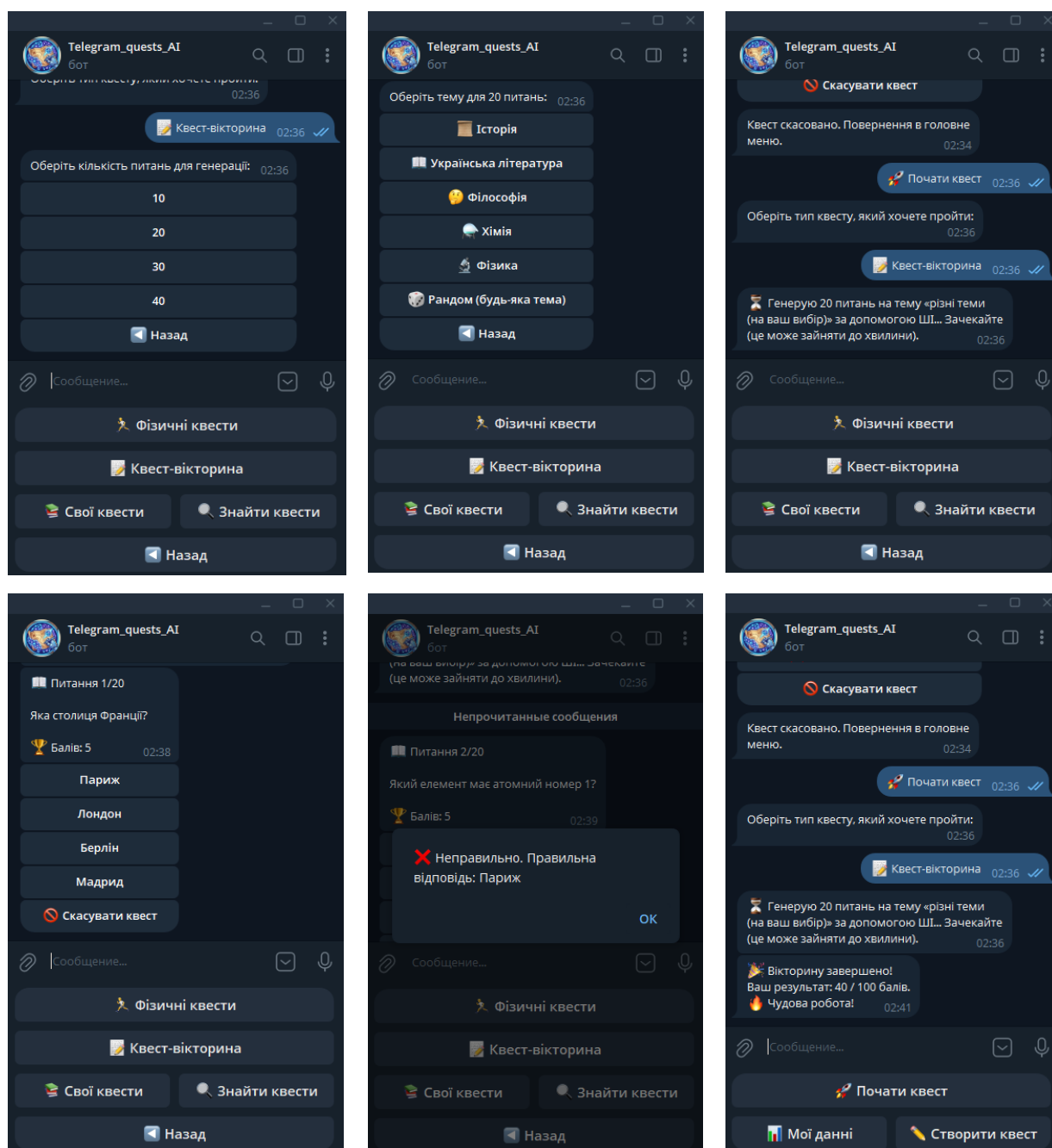


Рисунок 3.4 – Вибір кількості питань вікторини, вибір теми та проходження

Користувач також може створити власні фізичні квести або вікторини. Для цього потрібно ввести назву, опис дії (або питання), варіанти відповідей та кількість балів. Після збереження створений квест зберігається в бібліотеці, прив'язаний до акаунту. Проходження власного квесту відбувається так само, як і згенерованого. Скасування завершує квест без нарахування балів, рисунок 3.5.

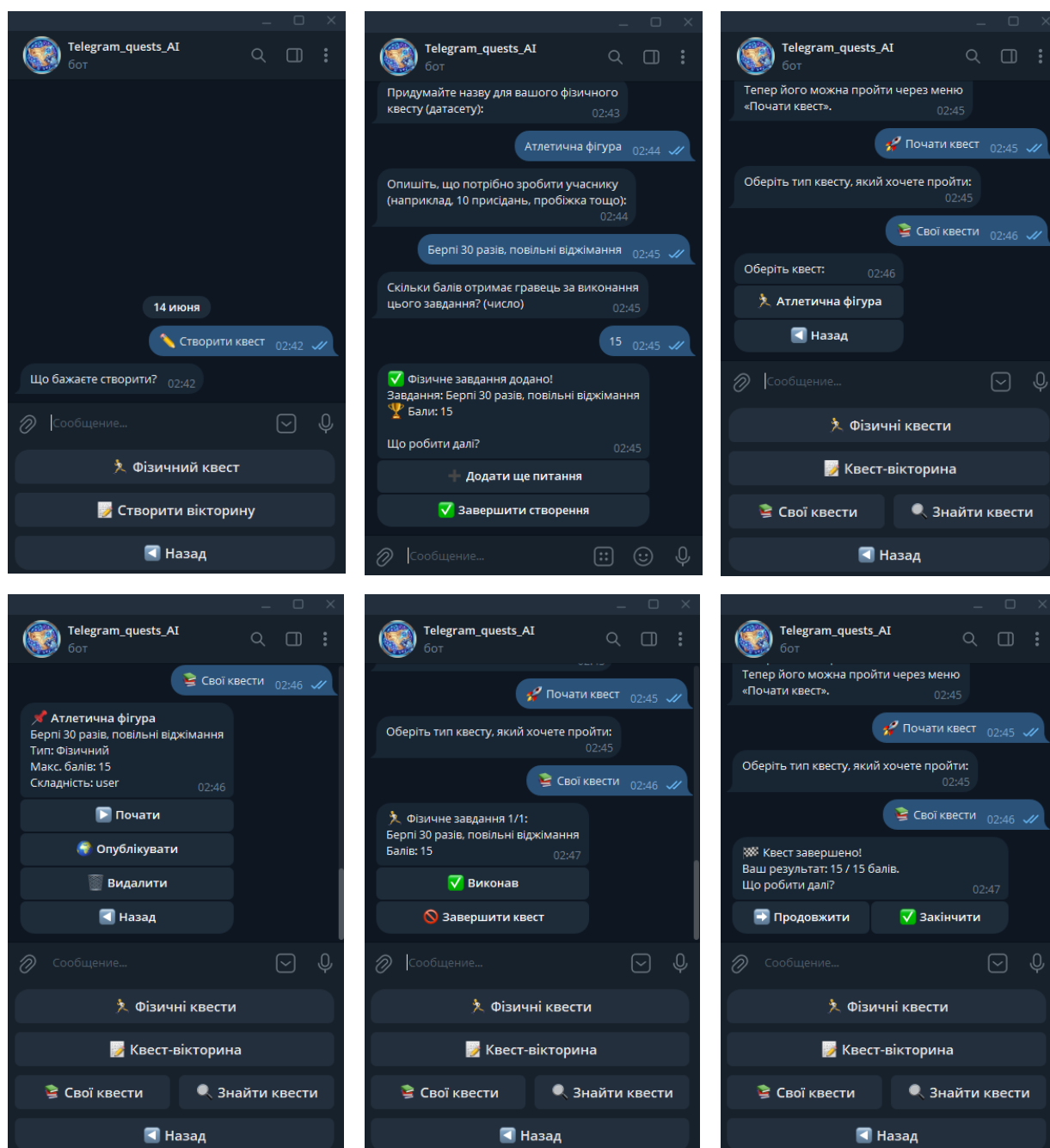


Рисунок 3.5 – Процес створення власного квесту, тестування та проходження

Додатково користувач може опублікувати власні квести. Для цього треба обрати квест з бібліотеки та натиснути кнопку «Опублікувати». Система надасть унікальний код датасету. Інший користувач може активувати цей датасет, ввівши код через кнопку «Знайти квести», рисунок 3.6.

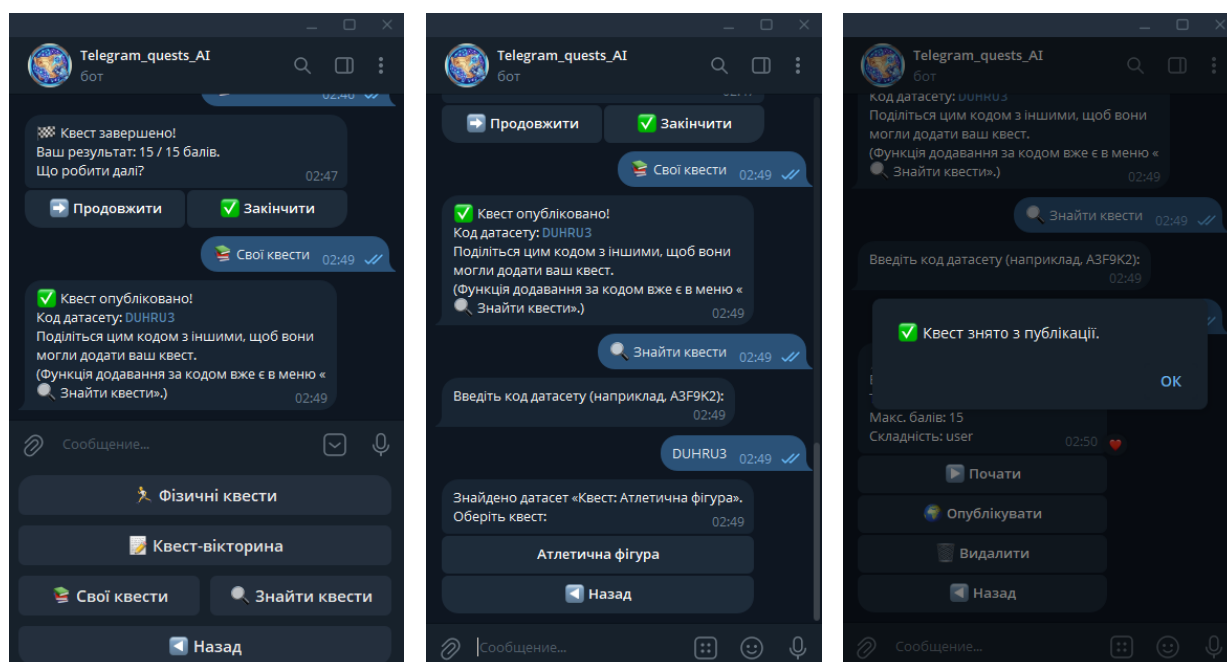


Рисунок 3.6 – Публікація та активація квесту та датасету

Також було протестовано можливість пройти згенеровану власноруч квест-вікторину. При проходженні пропонується 4 варіанти відповіді, з яких лише один правильний. Після завершення вікторини відбувається підрахунок кількості балів, рисунок 3.7.

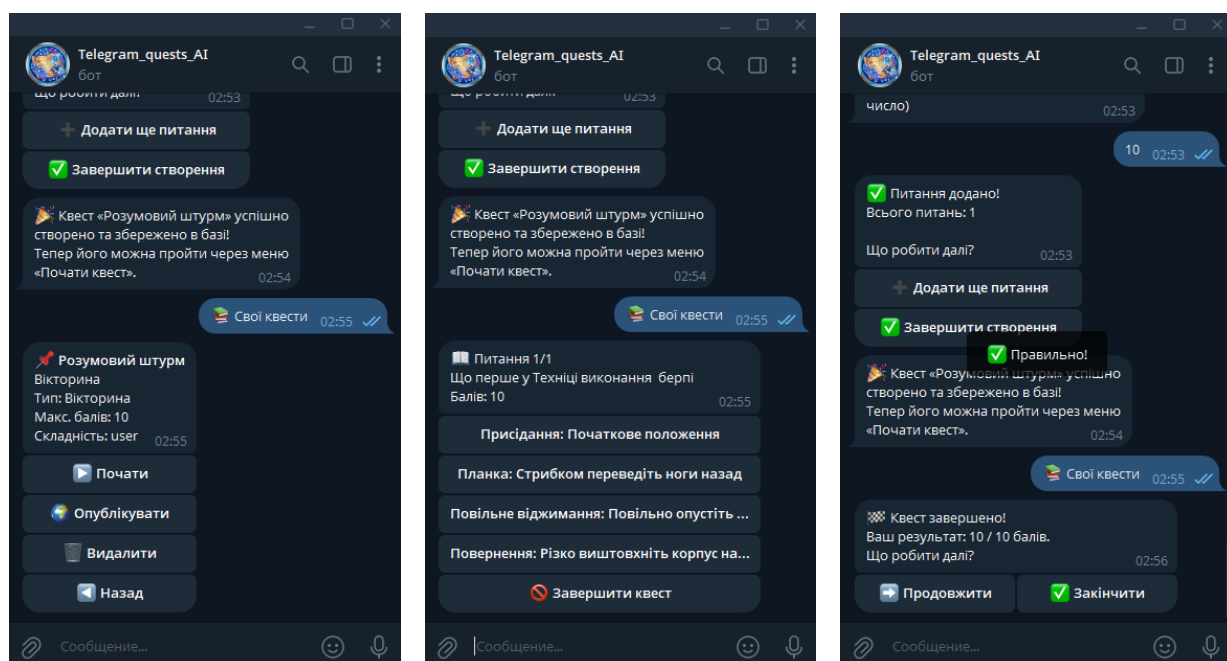


Рисунок 3.7 – Проходження квест-вікторини та завершення з нарахуванням балів

3.5 Керівництво користувача

Після запуску бота користувач взаємодіє з ним через текстові команди та кнопки. Інтерфейс побудовано на клавіатурах із двома типами кнопок: звичайні (ReplyKeyboard) з'являються в полі введення, а inline-кнопки розташовані безпосередньо під повідомленнями.

Реєстрація та вхід: для початку роботи потрібно знайти бота в Telegram за ім'ям користувача (username) та надіслати команду /start. Бот відповість вітанням і кнопкою « Почати гру». Після натискання цієї кнопки система автоматично реєструє користувача, використовуючи його Telegram ID, ім'я та username. Жодних додаткових даних вводити не потрібно.

Головне меню: після реєстрації з'являється головне меню з трьома основними кнопками: « Почати квест», « Мої данні», « Створити квест». Також доступна кнопка « Назад» для повернення з будь-якого підменю.

Перегляд статистики: кнопка « Мої данні» показує поточний рівень, кількість пройдених квестів, загальну кількість спроб, правильних і неправильних відповідей, а також середній бал у відсотках. Ці дані оновлюються автоматично після кожного завершеного квесту.

Початок квесту: після натискання « Почати квест» бот пропонує вибрати тип квесту: « Фізичні квести», « Квест-вікторина», « Свої квести» або « Знайти квести».

1. Фізичні квести: бот генерує набір вправ через ШІ. Користувач обирає кількість вправ: 5, 10, 15, 20, та рівень складності: легкий, середній, важкий. Після генерації кожна вправа з'являється з кнопками «Виконав» і «Пропустити». За виконану вправу нараховуються бали: 5, 10 або 15 залежно від складності. Пропущена вправа не дає балів. Після останньої вправи бот показує загальний результат.

2. Квест-вікторина: бот пропонує обрати кількість питань: 10, 20, 30, 40 та тему: історія, українська література, філософія, хімія, фізика або випадкова. Після короткої паузи бот надсилає перше питання з чотирма варіантами відповідей. Користувач натискає одну з inline-кнопок. Якщо відповідь правильна – нараховується 5 балів, якщо ні – бот показує правильну відповідь без нарахування. Після завершення всіх питань виводиться підсумок.

3. Свої квести: цей пункт показує всі квести, які користувач створив раніше. Кожен квест відображається у вигляді кнопки з назвою. Після вибору бот показує детальну інформацію: опис, тип, максимальна кількість балів, складність. Власник квесту може почати його, опублікувати, зняти з публікації або видалити.

4. Знайти квести: призначений для доступу до чужих опублікованих квестів. Користувач вводить код датасету, шість символів, наприклад, A3F9K2. Якщо код правильний, бот показує список квестів з цього датасету, і користувач може обрати будь-який для проходження.

Створення власного квесту: натискання « Створити квест» відкриває два варіанти: « Фізичний квест» або « Створити вікторину».

Для вікторини: спочатку вводиться назва. Потім бот послідовно запитує текст питання, чотири варіанти відповідей, кожен з нового рядка, номер правильної відповіді (1–4) та кількість балів. Після кожного питання з'являються кнопки «Додати ще питання» або «Завершити створення». Після завершення квест зберігається в базі даних і стає доступним у розділі «Свої квести».

Для фізичного квесту: після введення назви бот запитує опис завдання, наприклад, «10 присідань», та кількість балів за виконання. Можна додати кілька завдань. Збереження відбувається аналогічно.

Публікація квесту: у розділі «Свої квести» після вибору свого квесту власник бачить кнопку «Опублікувати». Після натискання бот створює датасет, генерує унікальний код і показує його. Цей код можна надіслати іншим користувачам. Вони зможуть знайти квест через « Знайти квести».

За потреби власник може зняти квест із публікації кнопкою «Зняти з публікації».

Видалення квесту: поруч із кнопкою публікації для власного квесту є кнопка «Видалити». Після підтвердження квест безповоротно видаляється разом з усіма питаннями, відповідями та записами прогресу. Опублікований квест перед видаленням бажано зняти з публікації.

Завершення сесії: після завершення квесту бот показує результат і пропонує продовжити, обрати інший квест того ж типу або повернутися в головне меню. У будь-який момент проходження можна перервати кнопкою «Завершити квест» - результат не збережеться.

Інтерфейс бота супроводжується емодзі для візуального розрізнення пунктів. Усі помилкові дії, невірний код датасету, некоректне введення варіантів тощо, супроводжуються зрозумілими текстовими повідомленнями.

Висновки до розділу

У третьому розділі описано практичну реалізацію Telegram-бота для автоматизованого формування квестів. Визначено засоби розробки: мова Python, бібліотека Aiogram для асинхронної обробки повідомлень, PostgreSQL як система керування базами даних, SQLAlchemy ORM для доступу до даних, а також бібліотека g4f для інтеграції з великою мовною моделлю. Сформульовано вимоги до технічного та програмного забезпечення як для серверної частини, мінімальна конфігурація, операційна система, залежності, так і для клієнтського пристрою наявність Telegram.

Детально розглянуто програмну реалізацію кожного з шести ключових механізмів: реєстрація користувачів через Telegram ID, створення власних квестів через конструктор із покроковим збиранням даних, генерація вікторин та фізичних вправ за допомогою ШІ, проходження квестів із нарахуванням балів, робота з датасетами: публікація, пошук за кодом, зняття з публікації, а

також збереження та відображення статистики. Для кожного механізму наведено лістинги коду з поясненнями.

Результати тестування підтвердили коректну роботу всіх функціональних модулів: бот стабільно обробляє команди, генерує квести згідно з параметрами, зберігає дані в базі та оновлює статистику. Помилки введення обробляються з повідомленнями, що не порушує роботу застосунку. Керівництво користувача містить покрокову інструкцію з усіх доступних функцій – від реєстрації до проходження квестів та публікації власних наборів.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ

4.1 Організаційно-правові основи забезпечення безпеки праці

У сучасних умовах охорона праці є невід'ємним елементом організації трудової діяльності, оскільки спрямована на забезпечення безпечних умов виконання професійних обов'язків, збереження здоров'я працівників та запобігання виникненню виробничого травматизму [14]. Особливого значення питання безпеки праці набувають у сфері інформаційних технологій, де працівники тривалий час використовують персональні комп'ютери та інші технічні засоби, що може негативно впливати на стан їхнього здоров'я.

Відповідно до Закону України «Про охорону праці», охорона праці визначається як система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів [15], спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності. Реалізація положень цього закону забезпечує конституційне право працівників на належні, безпечні та здорові умови праці незалежно від сфери їхньої професійної діяльності.

Законодавство України у сфері охорони праці встановлює єдині вимоги щодо організації системи управління безпекою праці на підприємствах, в установах та організаціях [15]. До основних нормативно-правових актів, що регулюють зазначені питання, належать Закон України «Про охорону праці», Кодекс законів про працю України, державні санітарні норми та правила, а також інші нормативні документи, спрямовані на створення безпечного виробничого середовища.

Важливе місце у системі правового регулювання трудових відносин займає Кодекс законів про працю України, який визначає основні права та обов'язки працівників і роботодавців [14]. Норми Кодексу спрямовані на забезпечення належних умов праці, дотримання режимів праці та відпочинку,

підвищення продуктивності праці та захист трудових прав громадян. Виконання цих вимог сприяє формуванню сприятливого психологічного клімату в колективі та зниженню рівня професійних ризиків.

Підставою для виникнення трудових відносин є трудовий договір, відповідно до якого працівник зобов'язується виконувати визначену роботу, а роботодавець – забезпечити належні та безпечні умови праці, своєчасно виплачувати заробітну плату та дотримуватися вимог трудового законодавства. Таким чином, створення безпечного робочого середовища є одним із ключових обов'язків роботодавця.

Разом із тим у сфері інформаційних технологій поширеним є використання цивільно-правових форм співпраці. На відміну від трудового договору, цивільно-правові договори орієнтовані насамперед на досягнення конкретного результату виконаних робіт або наданих послуг. У таких випадках питання забезпечення безпечних умов праці та дотримання режимів праці й відпочинку часто покладаються безпосередньо на виконавця, що потребує відповідального ставлення до організації власного робочого місця.

Для фахівців, діяльність яких пов'язана з розробленням, адмініструванням та супроводом програмних продуктів, важливого значення набуває дотримання вимог щодо ергономічної організації робочого місця [16], [17], забезпечення належного освітлення, підтримання оптимальних параметрів мікроклімату та раціонального режиму роботи з комп'ютерною технікою. Недотримання цих вимог може призвести до перевтоми, погіршення зору, виникнення захворювань опорно-рухового апарату та зниження працездатності.

Отже, дотримання організаційно-правових вимог у сфері охорони праці сприяє створенню безпечного виробничого середовища, збереженню здоров'я працівників та підвищенню ефективності їхньої роботи.

4.2 Характеристика об'єкта та виявлення потенційних небезпек

Об'єктом дослідження у даному підрозділі є робоче місце адміністратора Telegram-бота для автоматизованого формування квестів різних типів. Основними функціями адміністратора є контроль коректності роботи програмного забезпечення, моніторинг активності користувачів, перевірка та модерація створеного контенту, аналіз статистичних даних, оновлення інформаційного наповнення системи та взаємодія з користувачами у разі виникнення технічних питань.

Робоче місце адміністратора розташовується в офісному приміщенні та обладнане персональним комп'ютером або ноутбуком із підключенням до мережі Інтернет [17]. Для виконання службових обов'язків використовуються програмні засоби для адміністрування Telegram-бота, веббраузер, системи керування базами даних, засоби обробки інформації та офісне програмне забезпечення.

У процесі роботи адміністратор значну частину робочого часу проводить за екраном монітора у сидячому положенні [16]. Робота характеризується високим рівнем зорового навантаження, необхідністю концентрації уваги та обробки значних обсягів інформації. Крім того, у зв'язку з потребою оперативного реагування на звернення користувачів та можливі технічні несправності можуть виникати психоемоційні навантаження.

Аналіз умов праці дозволив виявити низку потенційних небезпек і шкідливих факторів, які можуть негативно впливати на стан здоров'я працівника та безпеку виконання ним професійних обов'язків. До них належать фактори фізичної та психофізіологічної природи, а також небезпеки загального характеру, пов'язані з функціонуванням об'єкта в сучасних умовах. Результати виявлення потенційних небезпек наведено в таблиці 4.1.

Таблиця 4.1 – Виявлення потенційних небезпек стосовно об'єкта проектування

№	Потенційна небезпека	Джерело небезпеки	Можливі наслідки
1	Підвищене зорове навантаження	Тривала робота з монітором, недостатнє або надмірне освітлення	Втома очей, погіршення гостроти зору, головний біль
2	Статичне навантаження на опорно-руховий апарат	Тривале перебування у сидячому положенні, неправильно організоване робоче місце	Біль у спині та шийі, порушення постави, захворювання опорно-рухового апарату
3	Психоемоційне перенапруження	Висока концентрація уваги, необхідність швидкого реагування на технічні проблеми та звернення користувачів	Стрес, перевтома, зниження працездатності
4	Ураження електричним струмом	Несправність комп'ютерної техніки, пошкодження електропроводки	Травмування різного ступеня тяжкості, загроза життю
5	Пожежа	Коротке замикання електромережі, перегрівання обладнання	Опіки, задимлення приміщення, матеріальні збитки
6	Недотримання параметрів мікроклімату	Недостатня вентиляція, підвищена або знижена температура повітря	Погіршення самопочуття, швидка втомлюваність
7	Перевищення допустимого рівня шуму	Робота офісної техніки, сторонні джерела шуму	Зниження концентрації уваги, підвищена дратівливість
8	Переривання електропостачання	Аварійні відключення електроенергії	Втрата даних, припинення роботи системи
9	Військова загроза	Повітряні тривоги, ракетні обстріли, інші наслідки воєнних дій	Загроза життю та здоров'ю працівників, пошкодження майна
10	Надзвичайні ситуації природного або техногенного характеру	Пожежі, аварії комунікацій, стихійні явища	Порушення роботи об'єкта, ризик травмування персоналу

За результатами проведеного аналізу встановлено, що робота адміністратора Telegram-бота супроводжується впливом переважно психофізіологічних та фізичних факторів виробничого середовища. Водночас існують небезпеки загального характеру, зокрема ризик виникнення пожежі та

загрози, пов'язані з воєнним станом в Україні. Більшість виявлених небезпек може бути мінімізована шляхом впровадження організаційних і технічних заходів безпеки, що обумовлює необхідність подальшого дослідження ризику їх реалізації та розробки відповідних профілактичних заходів у наступному підрозділі.

4.3 Дослідження ризику реалізації потенційних небезпек на об'єкті проєктування та розробка заходів щодо їх попередження

Оцінювання ризику є важливим етапом системи управління охороною праці, який полягає у визначенні ймовірності виникнення небезпечних подій та оцінці можливих наслідків їх реалізації. Проведення оцінки ризиків дозволяє своєчасно виявляти потенційні небезпеки [15], визначати їх пріоритетність та розробляти заходи, спрямовані на зниження рівня професійного ризику.

Основними завданнями процедури оцінювання ризиків є виявлення джерел небезпеки, аналіз умов виникнення небезпечних ситуацій, визначення тяжкості можливих наслідків та обґрунтування профілактичних заходів. Результати оцінювання використовуються для прийняття управлінських рішень щодо забезпечення безпечних умов праці та запобігання нещасним випадкам.

Для аналізу ризиків, характерних для робочого місця адміністратора Telegram-бота для автоматизованого формування квестів різних типів, використано метод дерева відмов та матричний метод оцінювання ризику. Застосування зазначених методів дозволяє встановити взаємозв'язки між причинами виникнення небезпечних подій, а також визначити рівень їх значущості.

Для працівників, діяльність яких пов'язана з адмініструванням Telegram-бота для автоматизованого формування квестів, існує ризик виникнення психоемоційного виснаження [16]. Це пояснюється необхідністю постійного

контролю за роботою програмної системи, обробкою звернень користувачів, швидким реагуванням на можливі технічні збої та виконанням завдань у встановлені терміни. Основні фактори, що можуть призвести до психоемоційного перенапруження працівника, наведено на рисунку 4.1.

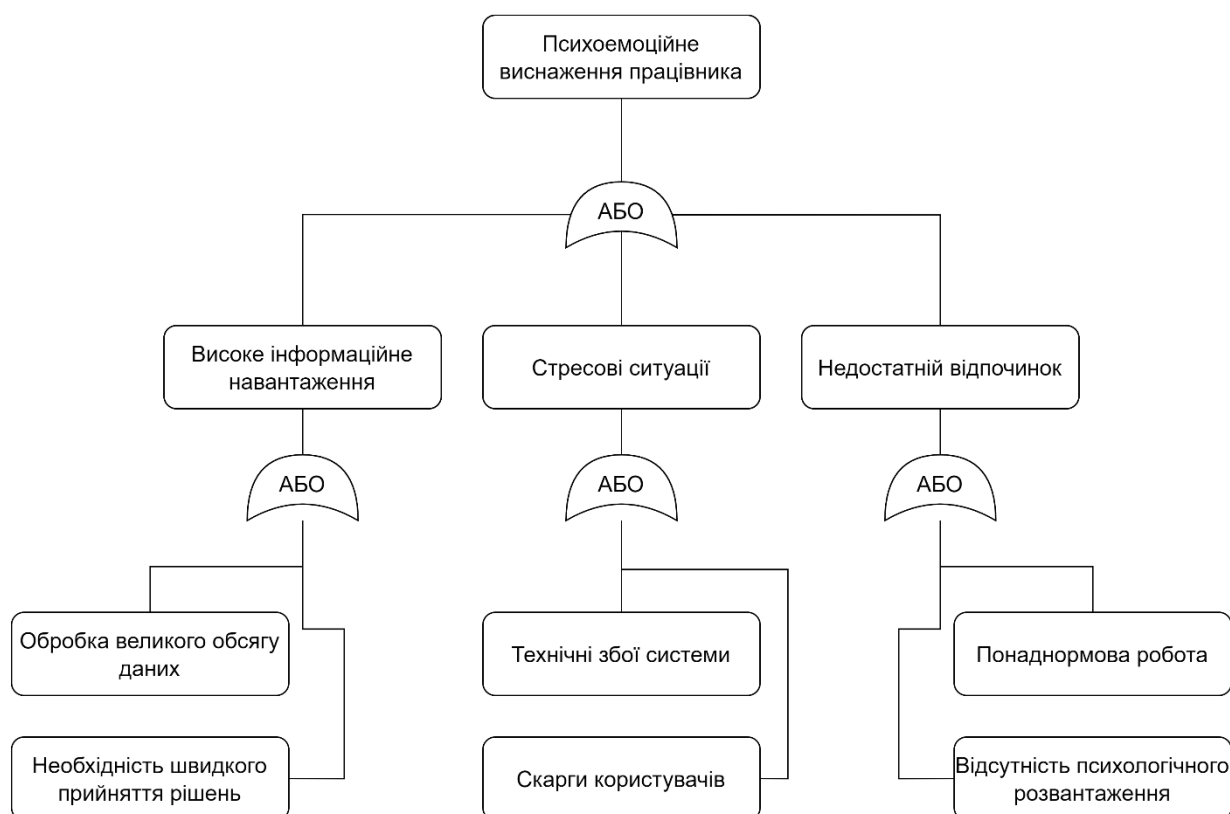


Рисунок 4.1 – Дерево відмов психоемоційного виснаження адміністратора Telegram-бота

Аналіз дерева відмов свідчить про те, що виникнення психоемоційного виснаження обумовлюється сукупністю організаційних та психологічних чинників. Найбільш вагомими серед них є високе інформаційне навантаження, стресові ситуації, пов'язані з технічними несправностями системи, а також недостатній рівень відпочинку. Для запобігання професійному вигоранню доцільно дотримуватися раціонального режиму праці та відпочинку, регламентувати тривалість роботи за комп'ютером, а також забезпечити можливість психологічного розвантаження працівників.

Тривала робота за комп'ютером може негативно позначатися на стані здоров'я працівника та спричиняти розвиток професійно зумовлених порушень. Найбільш поширеними наслідками є погіршення зору, захворювання опорно-рухового апарату та загальна перевтома організму. Причинно-наслідкові зв'язки, що можуть призвести до зазначених наслідків, подано на рисунку 4.2.

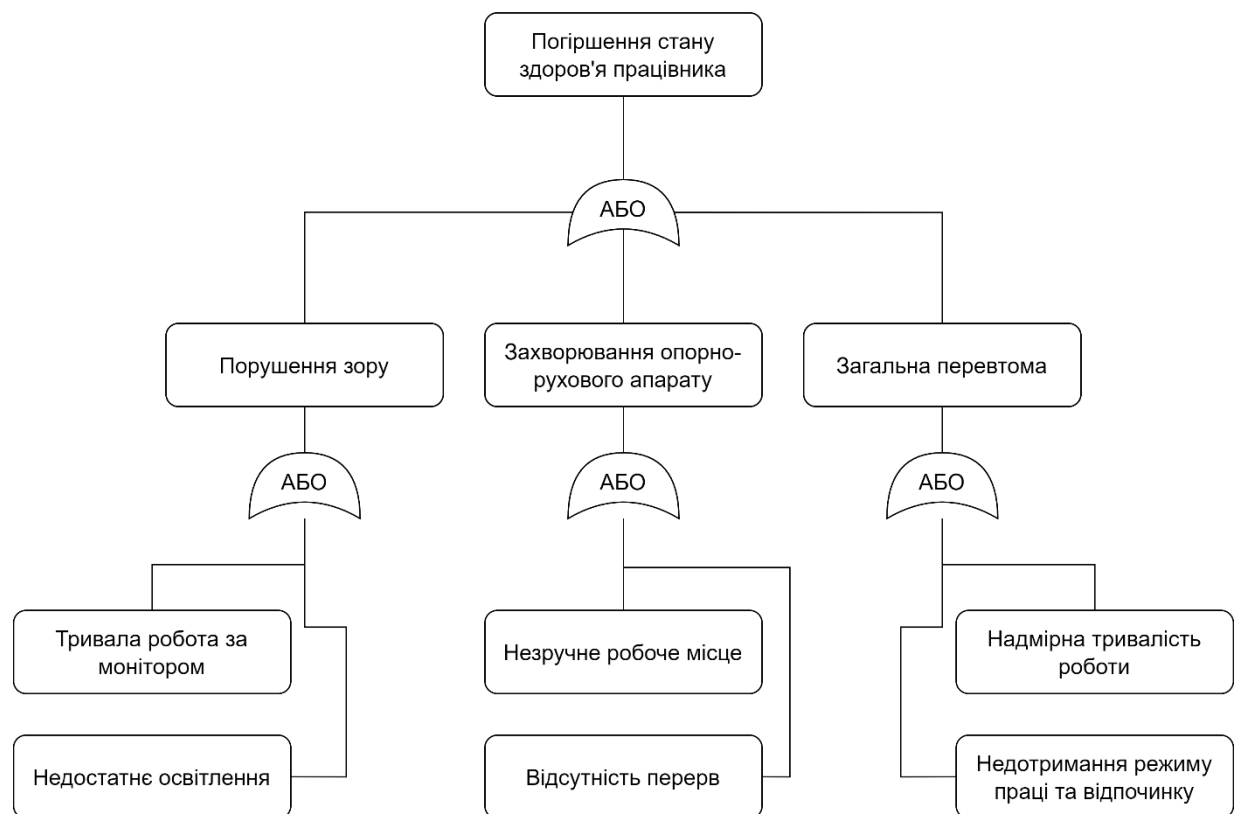


Рисунок 4.2 – Дерево відмов погіршення стану здоров'я працівника внаслідок тривалої роботи за персональним комп'ютером

Як видно з рисунка 4.2, основними факторами ризику є недостатнє освітлення робочого місця, відсутність регламентованих перерв, неправильна організація робочого місця та надмірна тривалість роботи за монітором. З метою зниження ймовірності реалізації зазначених небезпек необхідно використовувати ергономічні меблі [16], забезпечити відповідність освітлення встановленим нормам та дотримуватися рекомендацій щодо чергування періодів праці та відпочинку.

Оскільки робоче місце адміністратора Telegram-бота обладнане комп'ютерною технікою та підключене до електромережі, існує ризик виникнення пожежі [18]. Причинами її виникнення можуть бути несправності електрообладнання, пошкодження електропроводки, перевантаження мережі або недотримання правил пожежної безпеки. Дерево відмов, яке відображає основні причини виникнення пожежі в офісному приміщенні, наведено на рисунку 4.3.

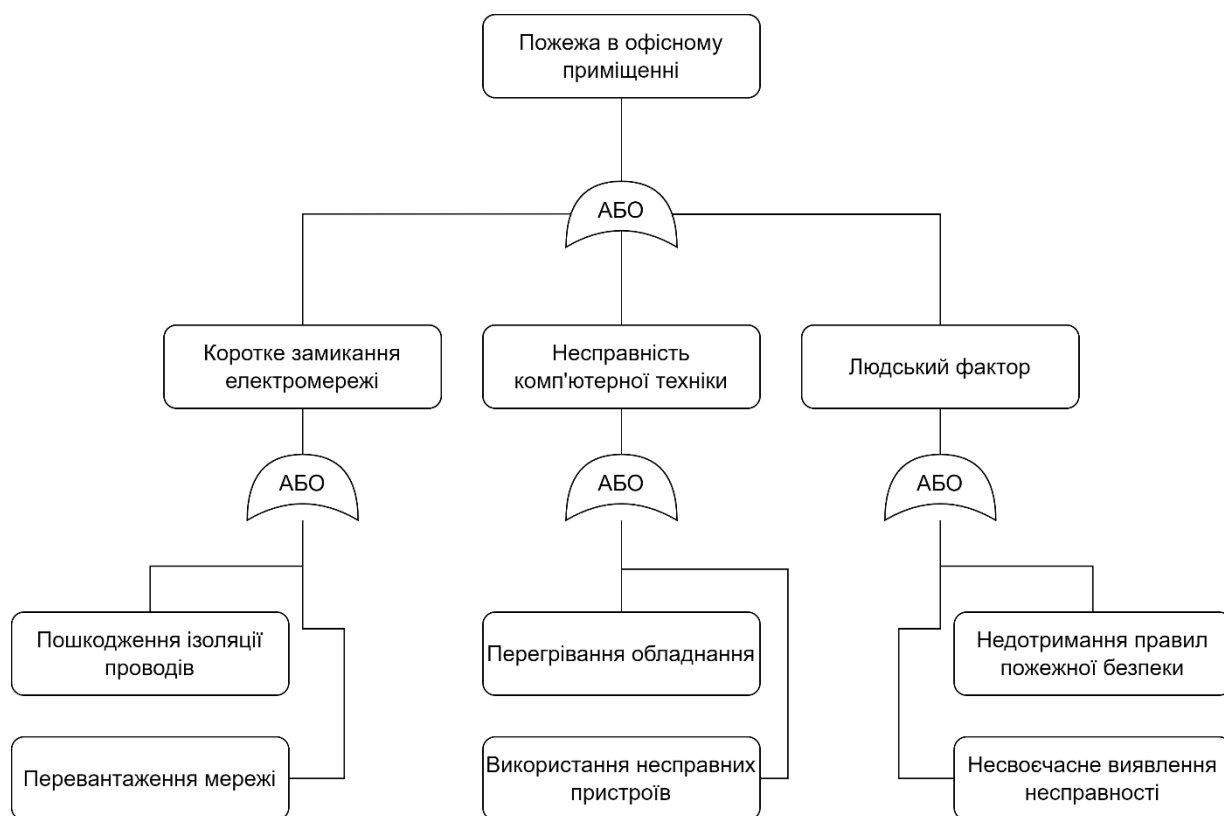


Рисунок 4.3 – Дерево відмов виникнення пожежі в офісному приміщенні

Проведений аналіз дозволяє зробити висновок, що найбільш небезпечними факторами є несправність електричного обладнання та людський фактор. Для зменшення ризику виникнення пожежі необхідно здійснювати періодичний контроль технічного стану електрообладнання [17], [18], не допускати використання несправних подовжувачів і проводів, а також забезпечити наявність первинних засобів пожежогасіння та проведення відповідних інструктажів з пожежної безпеки.

За результатами проведеного дослідження встановлено, що діяльність адміністратора Telegram-бота супроводжується впливом переважно психофізіологічних та фізичних факторів. Водночас своєчасне впровадження організаційних і технічних заходів дає змогу суттєво знизити рівень професійних ризиків та забезпечити належні умови праці. Рекомендовані заходи щодо мінімізації виявлених небезпек наведено в таблиці 4.2.

Таблиця 4.2 – Заходи щодо зниження ризиків

№	Небезпека	Запропонований захід	Очікуваний результат
1	Підвищене зорове навантаження	Раціональна організація освітлення, дотримання режиму праці та відпочинку	Зменшення втоми очей та профілактика порушень зору
2	Статичне навантаження на опорно-руховий апарат	Використання ергономічних меблів, виконання виробничої гімнастики	Зниження ризику розвитку захворювань опорно-рухового апарату
3	Психоемоційне перенапруження	Регламентування робочого часу, чергування видів діяльності	Зменшення рівня стресу та підвищення працездатності
4	Пожежа	Регулярна перевірка електрообладнання, забезпечення приміщення первинними засобами пожежогасіння	Зниження ймовірності виникнення пожежі та мінімізація її наслідків
5	Військова загроза	Дотримання алгоритму дій під час повітряної тривоги, використання укриттів	Підвищення рівня безпеки працівників
6	Перебої електропостачання	Використання джерел безперебійного живлення та резервного збереження даних	Запобігання втраті інформації та забезпечення безперервності роботи

Висновки до розділу

У даному розділі було розглянуто питання забезпечення безпеки праці під час виконання професійних обов'язків, пов'язаних із адмініструванням Telegram-бота для автоматизованого формування квестів різних типів. Проаналізовано організаційно-правові основи охорони праці в Україні та встановлено, що створення безпечних і здорових умов праці є одним із пріоритетних напрямів державної політики у сфері трудових відносин. Визначено, що дотримання вимог законодавства у сфері охорони праці сприяє збереженню життя і здоров'я працівників та підвищенню ефективності їхньої професійної діяльності.

У ході дослідження було охарактеризовано робоче місце адміністратора Telegram-бота та проаналізовано потенційні небезпеки, що можуть виникати під час виконання посадових обов'язків. Встановлено, що найбільш імовірними є ризики, пов'язані з підвищеним зоровим навантаженням, статичним навантаженням на опорно-руховий апарат, психоемоційним перенапруженням, виникненням пожежі та впливом надзвичайних ситуацій.

Для дослідження причин виникнення окремих небезпечних ситуацій застосовано метод дерева відмов, що дозволило визначити основні фактори, які можуть призвести до реалізації виявлених ризиків. За результатами аналізу запропоновано комплекс організаційних і технічних заходів, спрямованих на підвищення рівня безпеки праці, зокрема забезпечення ергономічності робочого місця, дотримання режиму праці та відпочинку, здійснення контролю технічного стану електрообладнання, проведення інструктажів з пожежної безпеки та дотримання алгоритмів дій під час надзвичайних ситуацій.

ВИСНОВКИ

У роботі виконано проєктування та реалізацію Telegram-бота для автоматизованого формування квестів. На етапі аналізу визначено ключові процеси: створення квестів (вручну або через ШІ), проходження з накопиченням балів, публікацію та пошук за кодом. Огляд платформ TeachQuest і Habitica показав, що вони не підтримують автоматичну генерацію контенту та не мають механізму обміну наборами за кодом. Це стало основою для постановки завдань власної розробки.

Під час проєктування побудовано діаграму варіантів використання, діаграму компонентів, діаграму класів та ER-діаграму бази даних. Розроблено шість алгоритмів: створення користувацької вікторини, генерації вікторини ШІ, проходження квесту, генерації фізичного квесту, публікації в датасет, пошуку за кодом датасету. Також створено чотири діаграми послідовності для основних сценаріїв. Кожен алгоритм і діаграму детально описано.

Для реалізації обрано Python, Aiogram (асинхронна обробка та FSM), PostgreSQL, SQLAlchemy ORM, бібліотеку g4f для доступу до генеративної моделі. У розділі програмної реалізації описано шість механізмів: реєстрація через Telegram ID, конструктор вікторин і фізичних квестів, генерація через ШІ з очищенням JSON, проходження з перевіркою відповідей, робота з датасетами (публікація, код, пошук, зняття), збереження статистики з обчисленням середнього балу та рівня.

У розділі охорони праці розглянуто нормативні вимоги, виявлено шкідливі фактори роботи програміста та запропоновано заходи їх мінімізації.

Підсумком є повністю працездатний Telegram-бот, що забезпечує автоматичну генерацію унікальних вікторин і фізичних квестів, конструктор для створення власних завдань, обмін квестами через датасети з кодом доступу та детальну статистику користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. TeachQuest – вебплатформа для гейміфікації освітнього процесу, що використовує RPG-елементи для підвищення залученості учнів. URL: <https://teach-quest.com/>
2. Habitica – вебсервіс та мобільний застосунок для управління завданнями і формування звичок із використанням механік гейміфікації. URL: <https://habitica.com/>
3. Hamari J., Koivisto J., Sarsa H. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In: Proceedings of the 47th Hawaii International Conference on System Sciences, 2014. URL: <https://doi.org/10.1109/HICSS.2014.377>
4. Mekler E. D., Brühlmann F., Tuch A. N., Opwis K. Towards Understanding the Effects of Individual Gamification Elements on Intrinsic Motivation and Performance. Computers in Human Behavior. 2017. Vol. 71. P. 525–534. URL: <https://doi.org/10.1016/j.chb.2015.08.048>
5. Sailer M., Hense J. U., Mayr S. K., Mandl H. How Gamification Motivates: An Experimental Study of the Effects of Specific Game Design Elements on Psychological Need Satisfaction. Computers in Human Behavior. 2017. Vol. 69. P. 371–380. URL: <https://doi.org/10.1016/j.chb.2016.12.033>
6. Aiogram 3.19.0 documentation. URL: <https://docs.aiogram.dev/en/v3.19.0/>
7. datetime – Basic date and time types. Python Documentation. URL: <https://docs.python.org/3/library/datetime.html>
8. asyncio – Asynchronous I/O. Python Documentation. URL: <https://docs.python.org/3/library/asyncio.html>
9. SQLAlchemy Documentation. URL: <https://docs.sqlalchemy.org/en/20/>
10. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/>
11. OpenAI API Documentation. URL: <https://platform.openai.com/docs/>

12. Telegram Bot API Documentation. URL:
<https://core.telegram.org/bots/api>
13. diagrams.net (draw.io) Documentation. URL:
<https://www.diagrams.net/doc/>
14. Кодекс законів про працю України. URL:
<https://zakon.rada.gov.ua/laws/show/322-08#Text>
15. Про охорону праці. URL: <https://zakon.rada.gov.ua/laws/show/2694-12#Text>
16. Про затвердження Мінімальних вимог безпеки та охорони здоров'я при роботі з екранними пристроями. URL:
<https://zakon.rada.gov.ua/laws/show/z0508-18#Text>
17. Про затвердження Мінімальних вимог безпеки і охорони здоров'я при використанні працівниками робочого обладнання під час роботи. URL:
<https://zakon.rada.gov.ua/laws/show/z0097-18#Text>
18. Про затвердження Правил пожежної безпеки в Україні. URL:
<https://zakon.rada.gov.ua/laws/show/z0252-15#Text>
19. Венжега О. Є., Братерська Н. М. Гейміфікаційні механізми підвищення мотивації користувачів у Telegram-ботах фізичної активності. Матеріали XIX Всеукраїнської науково-технічної конференції здобувачів вищої освіти «Сталий розвиток міст: поствоєнний період» (91-ї науково-технічної конференції ХНУМГ ім. О. М. Бекетова) : в 5-и ч. / Ч. 2. – Харків : ХНУМГ ім. О. М. Бекетова, 2026. 300 с., с.112-114.
<https://science.kname.edu.ua/images/dok/konferentsii/2026/Tezi%20konferencij/C.%202%20Energeticna%20informacijna%20ta%20transportna%20infrostruktura.pdf>
20. Венжега О. Є., Черкасова В. В. Автоматизована платформа створення квестів: інтеграція Telegram-бота та аналітичної вебпанелі (Харківський національний університет міського господарства імені О. М. Бекетова). Матеріали III (IX) Міжнародної науково-практичної здобувачів вищої освіти і молодих учених (Харків-Запоріжжя-Дніпро, 25–27 березня 2026

р.) / М-во освіти і науки України, ХНУМГ ім. О. М. Бекетова [Електронний ресурс] Електрон. дані. – Дніпро : Свідлер А.Л., 2026. 786 с., с.457-459.
https://knameedumy.sharepoint.com/:b:/g/personal/conference_ittp_2026_kname_edu_ua/IQCnuJ4s1jv9Tb0jYRQuTBqGAeAYynGiW0qtG4DdATmIUBw?e=EwW94G