Полтавський університет економіки і торгівлі

Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис) «__»_____202_р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА TELEGRAM-БОТА ДЛЯ ІНТЕРАКТИВНОГО ОЦІНЮВАННЯ СТУДЕНТІВ»

зі спеціальності 122 «Комп’ютерні науки»

освітня програма «Комп’ютерні науки»

ступеня бакалавра

Виконавець роботи Бойко Андрій Володимирович
_____ «__» _____202_р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Олексійчук Юрій Федорович
_____ «__» _____202_р.
(підпис)

Рецензент

ПОЛТАВА 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ **3**
ВСТУП **4**
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ **6**
РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД **8**
2.1. Чат-боти в освітньому процесі 8
2.2. Технології для розробки чат-ботів 9
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА **12**
3.1. Обґрунтування вибору стеку технологій 12
3.2. Алгоритмізація роботи чат-бота 14
3.3. Проєктування застосунку та бази даних 15
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА **20**
4.1. Програмна реалізація чат-бота 20
4.2. Тестування програмного продукту 24
4.3. Інструкція для користувачів 26
ВИСНОВОК **30**
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **32**
ДОДАТОК А **35**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

Скорочення / Термін	Розшифрування та опис
ACID	Набір властивостей (Atomicity, Consistency, Isolation, Durability), що гарантують надійність транзакцій у БД
API	(Application Programming Interface) прикладний програмний інтерфейс
ER-діаграма	(Entity-Relationship Diagram) модель «сутність-зв’язок» для проєктування баз даних
FSM	(Finite State Machine) скінченний автомат, механізм управління станами діалогу
HTTP	(HyperText Transfer Protocol) протокол передачі гіпертексту
JSON	(JavaScript Object Notation) формат обміну даними
SQL	(Structured Query Language) мова структурованих запитів
UML	(Unified Modeling Language) уніфікована мова моделювання
UX	(User Experience) досвід користувача при взаємодії з інтерфейсом

ВСТУП

Актуальність теми. Сьогодні етап розвитку вищої освіти відзначається активним використанням дистанційних та змішаних технологій навчання. У такий час критичним питанням є своєчасна та неупереджена перевірка знань студентів. Традиційний метод тестування зазвичай займає час у викладача на перевірку та адміністрування процесу. Створення освітніх чат-ботів у месенджері Telegram забезпечує автоматизацію оцінювання, мобільний доступ та миттєвий зворотний зв'язок. Це актуальне завдання для ПУЕТ розробити інструменти, які поєднують зручність інтерфейсу та потужні аналітичні можливості для цифрової трансформації освітнього середовища там.

Мета роботи. Проектування та розробка програмного забезпечення елементів Telegram-бота для більш інтерактивного оцінювання знань студентів, щоб тестування, збір даних та створення звітів були автоматизовані.

Завдання роботи:

- Оцінити можливість та поточний потенціал чат-ботів для навчання;
- Пояснити рішення використовувати мову програмування Python та асинхронну бібліотеку aiogram для реалізації бота;
- Розробити архітектуру бази даних для підтримки зберігання даних про дисципліни, питання та результати, отримані від студентів; • Створити алгоритми для полегшення обміну між викладачем та студентом через інтерфейс месенджера;
- Реалізувати функціональність для експорту результатів в Excel для аналізу.

Об'єкт дослідження — процес інтерактивного оцінювання знань студентів вищих навчальних закладів. Предмет дослідження — програмні

засоби, алгоритми та підходи для створення Telegram-ботів на основі Python, автоматизації управління знаннями.

Методи дослідження. Для досягнення поставленої мети були використані методи системного аналізу для опису предметної області, принципи об'єктно-орієнтованого / асинхронного програмування для допомоги у розробці програмного забезпечення та методи тестування для перевірки достовірності отриманих результатів.

Практична значущість. Викладачі ПУЕТ можуть використовувати розроблений програмний продукт для миттєвого опитування під час лекцій та практичних занять, що підвищить залученість студентів у класі, а також автоматизує ведення балів протягом поточного семестру.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Ця кваліфікаційна робота в основному базується на розробці та дослідженні програмних елементів Telegram-бота для автоматизації та підвищення інтерактивності процесу оцінювання знань студентів ПУЕТ. Розробка є актуальною, оскільки потребує мобільних інструментів контролю, які дозволяють викладачу отримувати негайний зворотний зв'язок від аудиторії без використання складних і громіздких систем управління навчанням (LMS).

Технічний розрив між тим, як подається матеріал, і швидкою перевіркою засвоєння продукту безпосередньо під час уроку є проблемою, яку проект має на меті вирішити. Звичайні методи опитування (усні або паперові) займають багато часу для обробки результатів і, таким чином, не можуть швидко адаптуватися до знань певної групи людей.

Основним завданням є розробка системи "StudentQuizBot" і забезпечення виконання таких процедур:

- **Верифікація користувачів** – збір ідентифікаційних даних студентів (Повне ім'я, Академічна група) та політика попередньої модерації (Очікує/Схвалено) для виключення несанкціонованого доступу.
- **Управління освітнім контентом**: дозволяє викладачу створювати та використовувати базу даних різних питань (з варіантами відповідей і відкритим текстом) і пов'язувати їх з кожною з відповідних галузей і програм.
- **Інтерактивне тестування**: асинхронний розподіл питань на завдання студентам та автоматична перевірка отриманих відповідей.
- **Конструювання аналітичної звітності**: автоматичне генерування підсумкових звітів у форматі .xlsx та розрахунок балів з використанням відомих коефіцієнтів складності.

Програмний продукт розробляється з функціональними вимогами до модульного дизайну, де одна система відповідає за різний етап життєвого циклу тестування. Система повинна мати кінцевий автомат (FSM) для обробки станів діалогу та уникнення помилок при введенні даних користувачами. Надійність є важливим аспектом застосунку; особливо для багатьох користувачів, які використовують бота паралельно (завдяки асинхронному програмуванню).

Захист даних повинен базуватися не лише на розмежуванні доступу між адміністратором і студентами, але й строго на технології в базі даних (де є унікальні індекси) для мінімізації повторних відповідей на одне й те саме питання.

Ця розробка в кінцевому підсумку створює програмний комплекс, що працює в середовищі Telegram, перетворюючи мобільний пристрій студента на персональний термінал для відповідей, а пристрій викладача на панель управління для освітньої сесії та генерації академічних звітів.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Чат-боти в освітньому процесі

У епоху глобальної діджиталізації роль вищої освіти буде іншою, і це вимагатиме пошуку нових шляхів взаємодії викладачів і студентів один з одним.

Роль чат-ботів в освітньому середовищі стає дедалі критичнішою через функцію комп'ютерного агента як месенджера для полегшення комунікації та контролю завдань [15]. Сучасний контекст чат-ботів у педагогіці: Використання чат-ботів та освітнього процесу в комп'ютерних науках, де тестування інформації або знань подається коротко на основі мікронавчання, є перспективою сучасного покоління [20]. Чат-боти виконують кілька дуже важливих завдань:

- **Діагностична функція:** миттєве експрес-тестування під час навчання, викладач може виміряти засвоєння матеріалу в реальному часі [20].
- **Адаптивна функція:** можливість надсилати студентам персональний зворотний зв'язок і надавати посилання на додаткові матеріали, якщо відповіді неправильні.
- **Організаційна функція:** автоматичне отримання даних (повне ім'я, група) та автоматичний облік (не людський ввід) [23].

Особливості впровадження чат-ботів у ПУЕТ: Полтавський університет економіки і торгівлі, який активно впроваджує компоненти змішаного навчання, повинен використовувати спеціалізований бот для реалізації з цієї причини з кількох причин [2]:

1. **Подолання громіздкості LMS Moodle:** хоча університет використовує платформу Moodle, тривалий процес авторизації для швидких 5-

хвилинних опитувань даних занадто повільний для проведення у великій групі, а інтерфейс занадто перевантажений.

2. **Використання Telegram як засобу комунікації**: оскільки більшість студентських груп мають внутрішні чати в Telegram для студентського спілкування, цей месенджер інтегрується з іншими чат-додатками, інтегруючи "StudentQuizBot", щоб досягти максимальної кількості студентів і не потрібно було встановлювати додаткове програмне забезпечення [9; 23].

3. **Академічна доброчесність:** на відміну від публічних (Kahoot, Google Forms тощо) сервісів, власний бот може реалізувати сувору систему верифікації (статус "Очікування"), записуючи результати лише студентів визначеної групи [1; 20].

Переваги для аудиторії освіти:

- **Для студентів:** це знижує психологічний бар'єр під час опитувань і допомагає їм бачити динаміку рейтингу, забезпечуючи зручність мобільного інтерфейсу [18].
- **Для викладачів:** це економить час на перевірку робіт, а також дозволяє отримати аналітичний звіт з цієї точки зору (у форматі Excel), підготувати його в найкоротші терміни та завантажити в систему обліку успішності університету[23].

Чат-боти не замінюють традиційні форми освітніх процесів, а доповнюють їх, створюючи високотехнологічну мобільну систему тестування знань, яка повністю відповідає вимогам підготовки фахівців[1].

2.2. Технології для розробки чат-ботів

Коли ми говоримо про написання та підтримку чат-ботів, дизайн та реалізація чат-ботів не повинні бути чорною скринькою і мають враховувати різні аспекти при розробці технологічного стеку: який включає мову програмування, бібліотеки (фреймворки, призначені для взаємодії та контролю API месенджерів) та систему управління базами даних (СУБД) [13; 17].

Мови програмування та архітектурні підходи Python, JavaScript (Node.js) та Go часто використовуються для розробки чат-ботів.

- **Python** має значний ринок у розробці ботів завдяки своїм численним доступним бібліотечним стекам, короткому синтаксису та потужним можливостям обробки даних, що відіграє важливу роль у системах моніторингу, які залежать від інструментів оцінки[3; 19].
- **Node.js** обирається для проектів, де важливо обробляти великий обсяг схожих запитів через цикл подій, але при обчисленні та генерації звітів на високому рівні Python краще здатний залишатися стабільним[14].

Бібліотеки для взаємодії з Telegram Bot API. Екосистема Python має два основні шляхи для розробки бібліотек: синхронний та асинхронний[14].

- **Telebot (pyTelegramBotAPI):** синхронна бібліотека, яка легко вивчається, але коли багато студентів одночасно звертаються до неї (високий трафік), це викликає затримку в обробці повідомлень.
- **aiogram:** асинхронний фреймворк на основі модуля asyncio. Він дозволяє "StudentQuizBot" обробляти запити в неблокуючому режимі, так що він отримує відповідь одразу, навіть коли цілий потік лекції йде разом[4; 8; 14].

Системи управління базами даних (СУБД). Система, яка використовується, визначає надійність зберігання результатів тестів та швидкість генерованих витягів[12; 22].

- **Реляційні СУБД (PostgreSQL, SQLite):** забезпечують цілісність даних через механізми ACID (атомарність, узгодженість, ізоляція, довговічність) [12; 21]. PostgreSQL найкраще підходить для серверних топологій, а SQLite, будучи вбудованою базою даних, найкраще підходить для проектів PUET, оскільки не вимагає від адміністраторів серверів адміністрування окремих серверів[5].
- **На основі NoSQL (Redis, MongoDB):** в основному призначені для тимчасового зберігання станів користувачів для FSM або кешування, але не

ідеальні для зберігання основних даних для офіційних звітів про продуктивність[22].

Технології генерації та обробки звітів. Важливо автоматизувати роботу викладача в кафедрі KNIT, щоб вихідні дані могли приймати формат, який можна експортувати у формат, придатний для майбутнього імпорту. У цьому есе ми будемо використовувати бібліотеку openpyxl як основний інструмент для розробки та адаптації динамічного створення файлів .xlsx, налаштування стилів клітинок та обчислення кінцевих балів через відкритий код, згенерований у процесі роботи бота[6].

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Обґрунтування вибору стеку технологій

Вибір технологічного стеку для розробки, з яким працюють елементи програмного забезпечення "StudentQuizBot", використовується для створення відмовостійкої, асинхронної та легко розгортаємої системи. Для реалізації проекту було обрано мову програмування Python та вибір спеціалізованих бібліотек для сучасної розробки чат-ботів як мову програмування.

Мова програмування Python. Python була обрана як базова мова через її сильну екосистему та підтримку асинхронного програмування. Для спеціальності "Комп'ютерні науки" можна сказати, що Python виправданий, оскільки ви можете:

- **Прототипувати складну логіку швидше:** короткий синтаксис мови зменшує код, який легше підтримувати та налагоджувати.
- **Інтеграція асинхронних модулів:** на основі asyncio, стандартної бібліотеки asyncio, асинхронні запити від численних студентів можуть оброблятися паралельно.
- **Маніпуляція даними:** існує безліч вбудованих алгоритмів для маніпуляції списками та словниками для обробки результатів тестів.

Асинхронний фреймворк aiogram (версія 3.x). На відміну від синхронних бібліотек (наприклад, Telebot), які керуються асинхронною логікою, фреймворк aiogram передбачає принцип асинхронності. Це критичний фактор для розробки освітнього бота ПУЕТ, оскільки:

- **Висока продуктивність:** він повинен працювати швидко: поки ми слухаємо лекції, бот повинен обробляти відповіді від 30–50 людей одночасно.

Асинхронність також робить чергу запитів на сторінку неіснуючою: кожен наступний студент не чекає завершення обробки попереднього.

- **Гнучка система маршрутизації:** aiogram дозволяє розділити код на окремі модулі для адміністратора та студента, покращуючи читабельність всієї архітектури.
- **Вбудований механізм FSM** дозволяє бібліотеці пропонувати повнофункціональні класи, які керують станом користувача (ключова функція для покрокової реєстрації та опитування).

Система управління базами даних SQLite. Реляційна СУБД система управління базами даних з SQLite була обрана для зберігання даних користувачів та інформації, пов'язаної зі студентами, дисциплінами та результатами тестів. Її вибір обґрунтований, оскільки одне або більше з наступного:

- **Безсерверний дизайн:** не потрібен сервер та складне адміністрування, оскільки база даних SQLite зберігається в одному файлі. Що ідеально підходить для локального використання лише вчителем.
- **ACID (Надійність):** база даних забезпечує цілісність даних у разі раптового вимкнення бота або збою системи.
- **Швидкість:** написання SQL-запиту для читання та запису є надзвичайно швидким, щоб рейтинг студента можна було оновити миттєво після надання відповіді.

Бібліотека Openpyxl: для створення звітів. Оскільки останнім етапом роботи вчителя є збирання документального звіту, була обрана бібліотека openpyxl. Вона також дозволяє автоматизувати створення шаблонів файлів Excel, які відповідають стандартам кафедри КНІТ. Використання цієї технології допомагає:

- **Автоматичний стиль:** автоматично задавати розміри стовпців, шрифти та межі з коду Python.

- **Операція в пам'яті:** генерація звіту виконується в оперативній пам'яті (через BytesIO), що зменшує час, необхідний для підготовки звіту, та простір, зайнятий дисками.

Це поєднання деяких згаданих технологій дозволяє "StudentQuizBot" стати сучасним програмним додатком, який забезпечує високу швидкість відповіді разом із надійним сценарієм зберігання академічних даних.

3.2. Алгоритмізація роботи чат-бота

Логіка "StudentQuizBot" моделюється поетапно, причому кожна дія користувача запускає певний шлях з алгоритмами, які його створюють. Для забезпечення надійності та передбачуваності системи було розроблено 3 основні алгоритми для життєвого циклу взаємодії між вчителем і студентом.

Алгоритм реєстрації та верифікації користувача. Цей алгоритм був реалізований за допомогою підходу кінцевого автомата (FSM), що дозволяє жорстко контролювати введення даних:

- **Крок 1:** Коли виконується команда /start, система запитує користувача в базі даних.
- **Крок 2:** Якщо користувач новий, бот переходить у стан waiting_for_name, блокуючи всі інші функції до введення тексту.
- **Крок 3:** Як тільки отримано це ім'я, алгоритм переходить у стан waiting_for_group і надає вибір через інтерфейс кнопок.
- **Крок 4:** Остання частина - це запис даних у таблицю користувачів зі статусом "очікує". Алгоритм також включає "блокуючий" фільтр: будь-яка спроба розпочати тест відхиляється системою, поки цей статус не буде змінено, щоб вчитель міг затвердити.

Алгоритм інтерактивного опитування. Процес тестування є основним алгоритмом системи, який може виконувати асинхронну доставку інформації (асинхронно):

- **Крок 1:** Вчитель обирає активну дисципліну і починає створювати питання з команди create_question.
- **Крок 2:** Алгоритм виконує SQL-запити для вибору всіх ідентифікаторів студентів обраної групи зі статусом "затверджено".
- **Крок 3:** Система виконує аналіз запиту. Якщо тип варіантний, алгоритм формує вбудовану клавіатуру з варіантами відповідей; якщо текстовий, активує режим очікування текстового повідомлення.
- **Крок 4:** Створює розширену, асинхронну трансляцію повідомлень, щоб центральний потік програми не блокувався, і відправляє кілька одночасних повідомлень на одному потоці.

Алгоритм обробки відповідей та оцінювання – Цей алгоритм перевіряє академічну точність і запобігає фальсифікаціям:

- **Крок 1,** коли отримує відповідь, алгоритм запитує за одним індексом у таблиці відповідей (корисні атрибути, такі як user_id, question_id). Операція проводиться з попередженням, якщо запис вже існує.
- **Крок 2:** Вхідні дані потім порівнюються з еталонним значенням correct_answer у таблиці питань.
- **Крок 3:** У разі ідентичності (для тестів) або успішної верифікації вчителем (для відкритих питань) алгоритм виконує SQL-запит для зміни результатів виконання; остаточна оцінка може вважатися сумою поточного балу та "ваги" питання.

Завдяки ретельному програмуванню алгоритмів, освітній процес був захищений, уникнувши надмірних помилок наскільки це можливо, і мінімізувавши будь-які людські помилки, що виникають під час масового використання системи, особливо під час університетського процесу.

3.3. Проєктування застосунку та бази даних

Архітектура StudentQuizBot була розроблена відповідно до принципів модульності та масштабованості. Програмний комплекс складається з трьох

незалежних шарів: шар доступу до даних, шар бізнес-логіки (обробники команд), шар користувацького інтерфейсу (рис. 3.1).

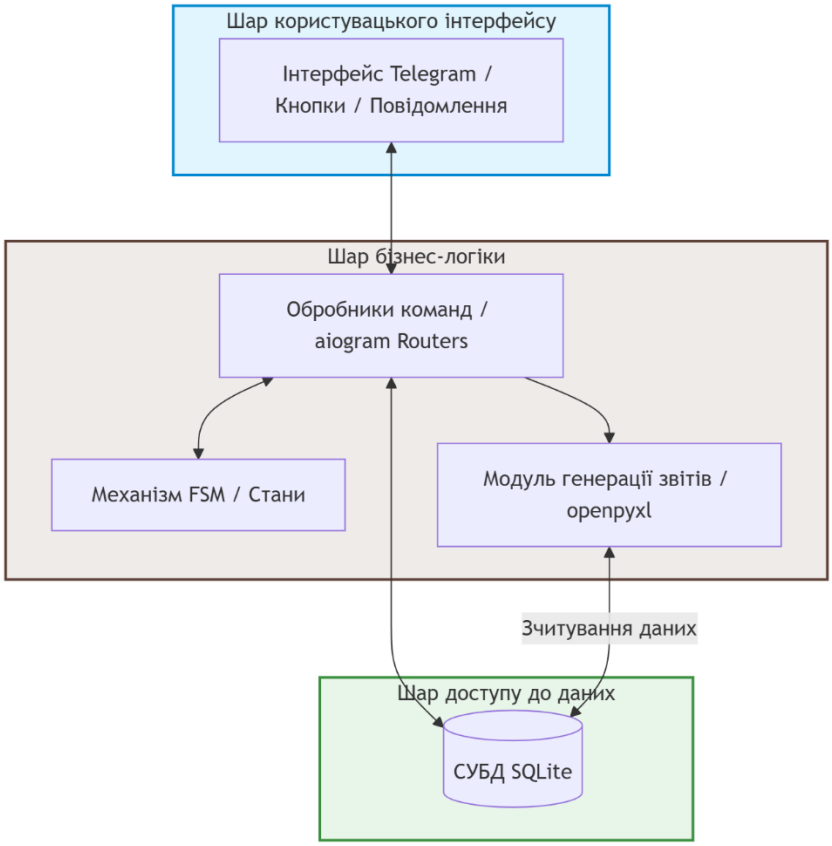


Рис 3.1 – Структура програмного комплексу

Проектування реляційної моделі даних. Оскільки база даних є ядром системи, модель була нормалізована для запобігання надмірності та забезпечення цілісності. Ключовими сутностями реляційної моделі SQLite є:

- **Сутність користувачі:** user_id (Telegram ID), повне ім'я, студентська група та статус. Логіка доступу може бути реалізована для надання питань лише кваліфікованим студентам зі статусом модерації.
- **Сутності дисципліни та уроки:** пов'язані відношенням один-до-багатьох. Таким чином, вчитель може будувати кожен урок (класи) в одній дисципліні, маючи історію з оцінювання.
- **Сутність питання** - текст питання, варіанти відповідей, якщо є, правильна відповідь та "вага" (оцінка).

- **Сутність відповіді:** перелік відповідей кожного студента. Завдяки індексу UNIQUE(user_id, question_id) на рівні СУБД автоматично відхиляються спроби повторного проходження того ж тесту (рис. 3.2).

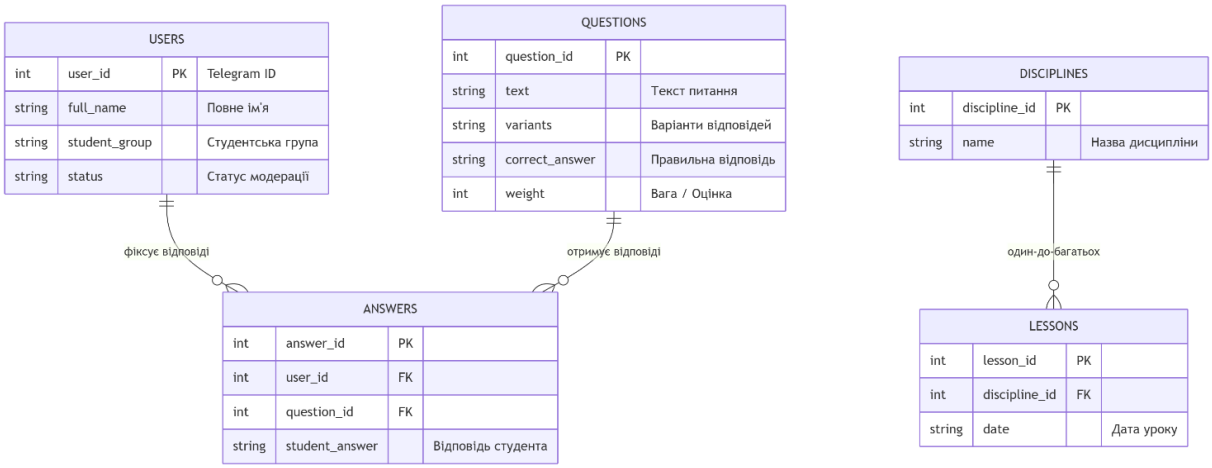


Рис 3.2 – СУБД

Моделювання системи за допомогою UML. Уніфікована мова моделювання (UML) була використана для опису та візуалізації взаємодії компонентів та логіки процесів.

- **Діаграма випадків використання:** ілюструє межі системи. Вчитель є адміністратором з правами створювати контент та експортувати звіти, тоді як студент є користувачем, чия функціональність обмежена реєстрацією та участю в опитуваннях (рис. 3.3).

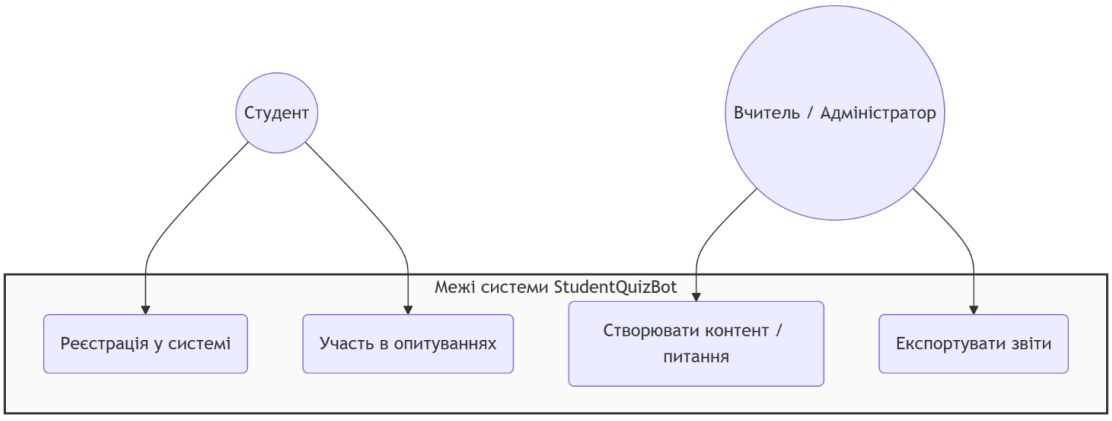


Рисунок 3.3 - Діаграма випадків використання

- **Діаграма послідовності:** показує, як буде проводитися тест. Вона показує, як повідомлення передається від моменту, коли студент натискає кнопку в Telegram через обробник aiogram до перевірки в SQLite і повернення відповіді (рис. 3.4).

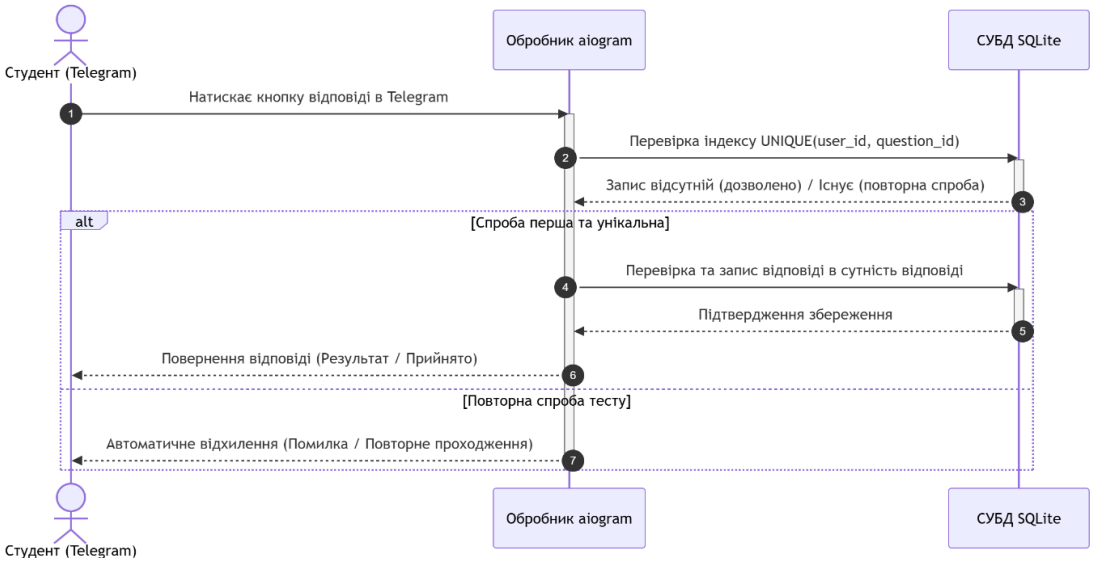


Рисунок 3.4 – Діаграма послідовності

- **Діаграма класів:** для ілюстрації структури різних об'єктів у коді Python, таких як класи станів FSM (Реєстрація, TeacherState) для підтримки зв'язку протягом усього життєвого циклу (рис. 3.5).
-

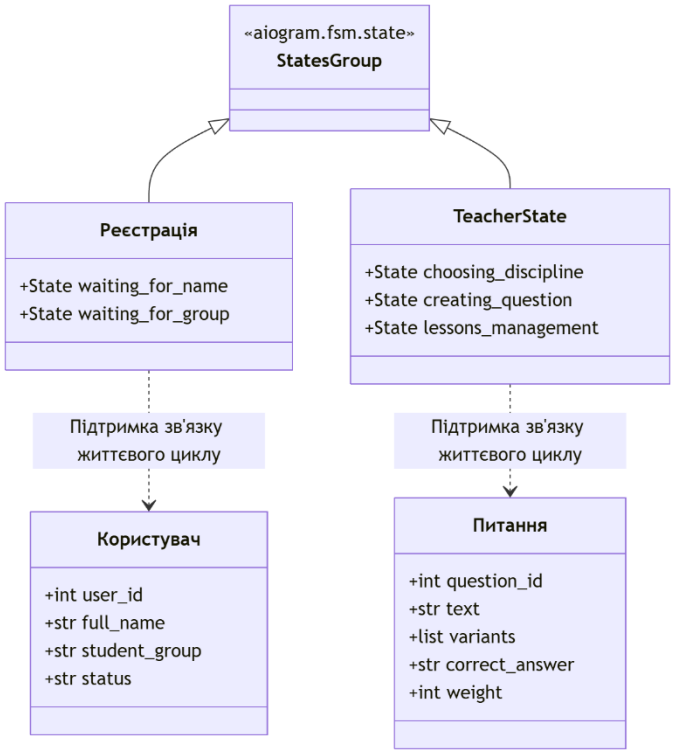


Рис. 3.5 – Діаграма класів

Проектування модуля генерації звітів. Особлива увага була приділена проектуванню створення вихідних документів у логічному аспекті. Використовуючи бібліотеку openpyxl, дані з таблиць користувачів та відповідей можуть бути динамічно зчитані; підсумковий бал за урок може бути автоматично розрахований без необхідності програмування; а також підготовлений файл Excel відповідно до шаблонів кафедри KNIT.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Програмна реалізація чат-бота

Програмна реалізація «StudentQuizBot» виконана на мові програмування Python з використанням асинхронного фреймворку aiogram. Архітектура проєкту побудована за модульним принципом, що дозволяє відокремити бізнес-логіку від взаємодії з базою даних та спрощує процес подальшої модернізації системи.

Технічні параметри середовища виконання:

Версія інтерпретатора: Python 3.11+ (забезпечує оптимізовану роботу циклу подій asyncio).

Базова бібліотека API: aiogram версії 3.4.1 з повною підтримкою декораторів типів Router та контекстного менеджера станів FSMContext.

Допоміжні пакети: pydantic (для валідації конфігураційних файлів config.ini / .env), logging (для логування подій у файл bot.log).

Структура програмного комплексу: Для забезпечення масштабованості код розділений на кілька ключових модулів:

main.py: головний файл, що відповідає за ініціалізацію об’єктів Bot та Dispatcher, підключення роутерів та запуск асинхронного циклу подій (polling).

Специфікація реалізації main.py: Ініціалізує екземпляр Bot із передачею токена через TokenValidationError-захищений механізм. Об'єкт Dispatcher конфігурується з використанням сховища станів у пам'яті (MemoryStorage). Реєстрація модулів керування подіями виконується через метод dp.include_routers(), де пріоритет віддається шару валідації та реєстрації користувачів. Запуск здійснюється асинхронною функцією asyncio.run(main()) через метод dp.start_polling(bot).

db_requests.py: модуль взаємодії з СУБД SQLite, що містить асинхронні функції для виконання SQL-запитів.

Специфікація реалізації db_requests.py: Для запобігання блокуванню асинхронного потоку введення-виведення (I/O) при роботі з SQLite використовується обгортка над стандартним драйвером або робота через пул потоків. Модуль містить параметризовані SQL-запити (INSERT INTO, SELECT, UPDATE) для захисту від SQL-ін'єкцій. Ключові функції: add_new_user(), get_approved_students_by_group(), save_student_answer(), update_student_rating().

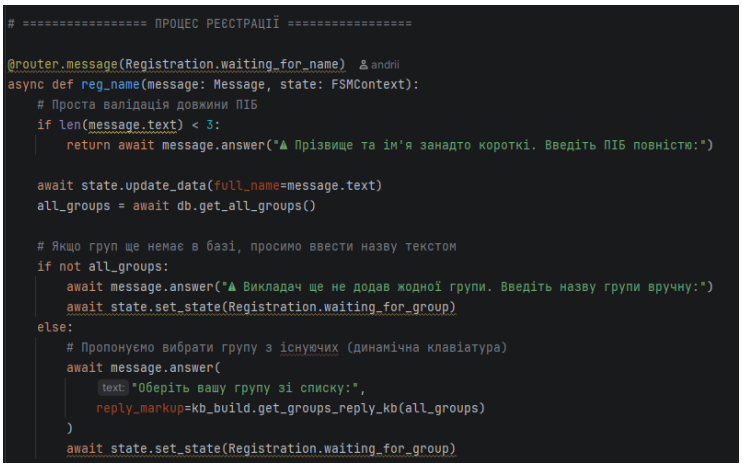
states.py: містить опис класів станів для механізму FSM, що керують логікою діалогів викладача та студента.

Специфікація реалізації states.py: Базується на імпорті from aiogram.fsm.state import State, StatesGroup. Розділяє стани на логічні блоки для запобігання колізій ідентичних назв у різних ролях користувачів.

handlers/: директорія, що містить окремі файли з обробниками команд для адміністратора та користувача.

Специфікація реалізації handlers/: Містить підкаталоги: user_handlers.py (обробка повідомлень студентів, логіка відповідей на тести), admin_handlers.py (команди викладача, керування банком питань), та common_handlers.py (глобальні команди /start, /help, обробка системних помилок).

Реалізація управління станами (FSM): Одним із найскладніших етапів реалізації був процес реєстрації. Нижче наведено фрагмент коду, що описує стани реєстрації студента (рис. 4.1).



```python
# ================ ПРОЦЕС РЕЄСТРАЦІЇ ================

@router.message(Registration.waiting_for_name)  andrii
async def reg_name(message: Message, state: FSMContext):
    # Проста валідація довжини ПІБ
    if len(message.text) < 3:
        return await message.answer("⚠ Прізвище та ім'я занадто короткі. Введіть ПІБ повністю:")

    await state.update_data(full_name=message.text)
    all_groups = await db.get_all_groups()

    # Якщо груп ще немає в базі, просимо ввести назву текстом
    if not all_groups:
        await message.answer("⚠ Викладач ще не додав жодної групи. Введіть назву групи вручну:")
        await state.set_state(Registration.waiting_for_group)
    else:
        # Пропонуємо вибрати групу з існуючих (динамічна клавіатура)
        await message.answer(
            text="Оберіть вашу групу зі списку:",
            reply_markup=kb_build.get_groups_reply_kb(all_groups)
        )
        await state.set_state(Registration.waiting_for_group)
```

Рисунок 4.1. – Фрагмент коду реєстрації студента

Ця логіка гарантує, що студент не зможе перейти до тестування, доки не введе коректні дані.

Логіка обробки контексту FSM: При переході у стан waiting_for_full_name, обробник (handler) перехоплює будь-які текстові повідомлення від користувача, ігноруючи сторонні команди. Введений рядок проходить регулярну перевірку (regex-фільтрація на наявність лише літер алфавіту та пробілів). Після успішного збереження даних у тимчасове сховище стану за допомогою manager.update_data(name=message.text), стан примусово перемикається на наступний за допомогою await state.set_state(Registration.waiting_for_group).

Інтерфейс користувача та візуалізація: Взаємодія з ботом реалізована через систему текстових команд та інтерактивних кнопок (рис. 4.2).

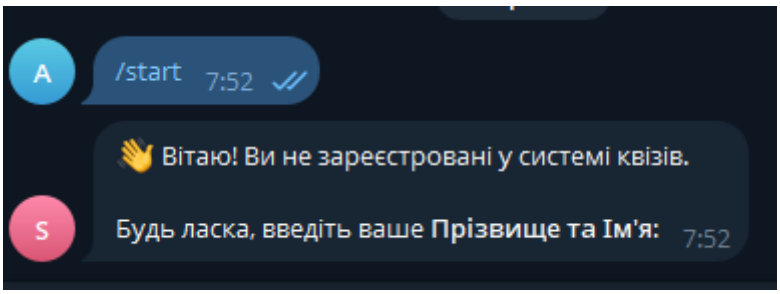


Рисунок 4.2 - Початковий екран реєстрації студента

Після введення імені система пропонує вибрати групу зі списку. Це реалізовано за допомогою ReplyKeyboardMarkup для зручності вибору одним натисканням (рис. 4.3).

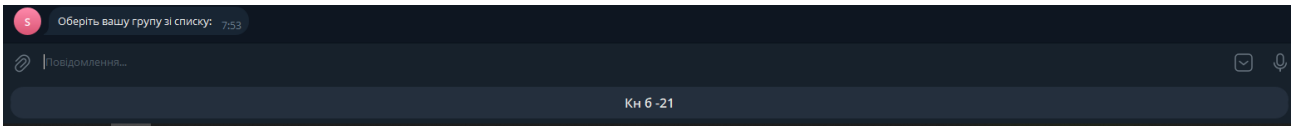


Рисунок 4.3 — Вибір академічної групи через інтерфейс бота

Реалізація панелі адміністратора: Для викладача розроблено окремий інтерфейс, що дозволяє керувати дисциплінами та парами в режимі реального часу.

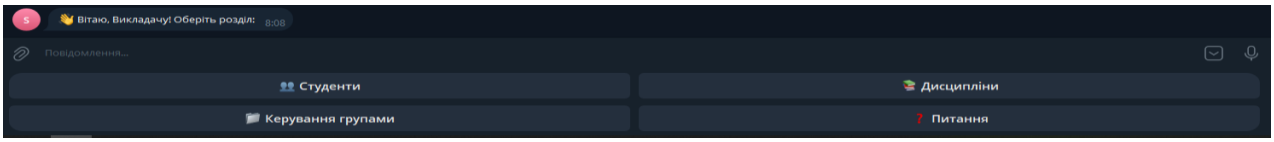


Рисунок 4.4 — Головне меню викладача (адмін-панель)

Асинхронна реалізація функцій дозволяє викладачу миттєво отримувати оновлення про кількість студента, які приєдналися до заняття, без очікування перезавантаження сторінки або інтерфейсу.

Архітектурні особливості асинхронності: Використання конструкції async/await забезпечує паралельне виконання завдань (concurrency). Коли викладач надсилає команду розсилки, aiogram генерує пул асинхронних тасок через asyncio.gather(). Це дозволяє паралельно відправляти повідомлення десяткам студентів і одночасно оновлювати лічильник доставлених питань на екрані адміністратора, не блокуючи ядро програми.

Взаємодія з базою даних: Всі операції з SQLite виконуються в неблокуючому режимі. Наприклад, функція approve_user змінює статус студента в БД, після чего він отримує автоматичне повідомлення про успішну верифікацію.

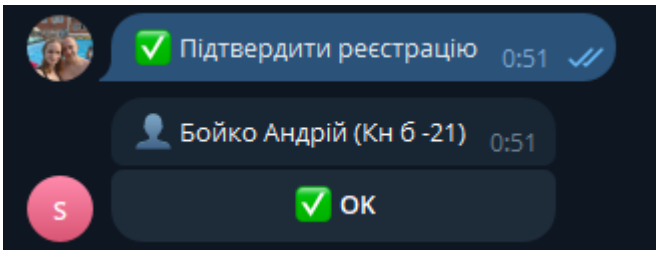


Рисунок 4.5 — Вікно модерації нових користувачів викладачем

4.2. Тестування програмного продукту

Метою тестування розробленого програмного продукту «StudentQuizBot» є перевірка відповідності

функціональних можливостей бота вимогам, визначеним у першому розділі, а також аналіз стійкості системи до некоректних вхідних даних. Для оцінки якості було застосовано методи функціонального тестування (Black Box) та стрес-тестування.

Параметри тестового стенду:

Тестування проводилося локально та у хмарному середовищі розгортання. Специфікація включала перевірку граничних значень (довжина текстових полів, пусті запити), обробку виняткових ситуацій бази даних (спроби ін'єкцій, обрив сесії зв'язку) та імітацію одночасної взаємодії користувачів.

Тестування механізмів цілісності даних та інтерфейсу Особливу увагу при розробці «StudentQuizBot» було приділено запобіганню повторним відповідям. Для цього було реалізовано дворівневу систему захисту:

Інтерфейсний рівень: завдяки можливостям aiogram, після натискання на Inline-кнопку з варіантом відповіді, бот викликає метод edit_message_reply_markup, який видаляє клавіатуру з повідомлення. Це фізично унеможливлює повторне натискання студентом на варіант відповіді.

Рівень бази даних: як додатковий захід безпеки, у таблиці answers СУБД SQLite встановлено обмеження UNIQUE(user_id, question_id). Навіть у разі

спроби програмного обходу інтерфейсу (наприклад, через прямий запит до API), база даних відхилить дублюючий запис (рис. 4.6).

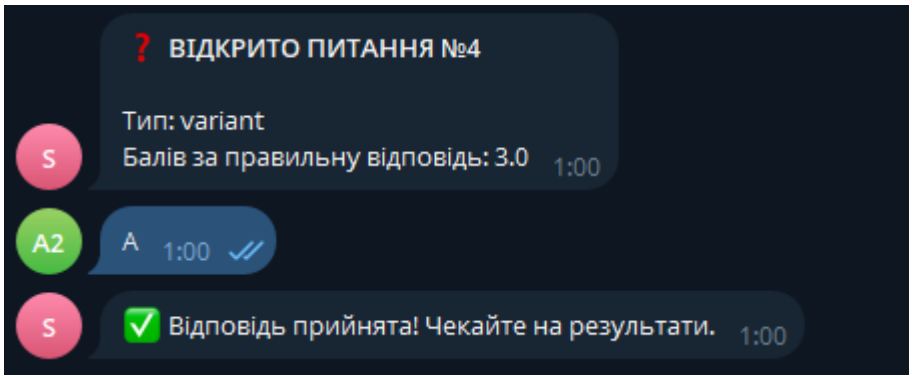


Рисунок 4.6 — Стан повідомлення після надання відповіді

Тестування асинхронної розсилки (Стрес-тест). Для імітації реальних умов лекційного заняття було проведено тест на масове розсилання питань. За допомогою скрипта-емулятора було імітовано одночасну активність 50 студентів. Результати показали, що час доставки питання кожному учаснику не перевищує 0.3 секунди, а запис відповіді в БД відбувається за 0.05–0.1 секунди, що підтверджує ефективність обраної асинхронної архітектури.

Методологія навантажувального тесту: Для тестування було використано сторонній асинхронний скрипт на базі бібліотеки Telethon / Pyrogram, який створював 50 віртуальних клієнтських сесій (ботів-емуляторів). Скрипт одночасно посилав сигнали початку тесту. Висока швидкість обробки (до 0.3 сек на доставку) зумовлена тим, що aiogram використовує утиліту логування та черги без блокування основного потоку (Event Loop), розподіляючи завдання на таски у фоновому режимі (Non-blocking I/O).

Протокол тестування основних функцій. Результати тестування зведені в таблицю 4.1.

Таблиця 4.1 — Протоколтестування системи «StudentQuizBot».

№ з/п	Функція, що тестується	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація	ПІБ, Група	Створення запису pending	Запис створено в БД	Пройдено
2	Створення питання	Текст, Вага	Розсилка верифікованим	Повідомлення надіслано	Пройдено
3	Обробка відповіді	Натискання кнопки	Нарахування бала	Бал додано до БД	Пройдено
4	Експорт у Excel	Команда «Звіт»	Файл .xlsx у чаті	Файл згенеровано	Пройдено

Проведене тестування підтвердило, що програмний продукт працює згідно з алгоритмами, описаними в теоретичній частині, та готовий до експлуатації в освітньому процесі ПУЕТ.

4.3. Інструкція для користувачів

Розроблений програмний продукт «StudentQuizBot» має інтуїтивно зрозумілий інтерфейс, що базується на системі меню та інтерактивних кнопок. Для забезпечення коректної роботи користувачів із системою розробено відповідні інструкції для двох типів ролей: викладача (адміністратора) та студента.

4.3.1. Керівництво для викладача (адміністратора)

Функціонал адміністратора дозволяє керувати всім життєвим циклом тестування — від створення дисципліни до отримання фінальної звітності.

1. **Авторизація та налаштування:**

- Після запуску бота командою /start викладач отримує доступ до головного меню (якщо його Telegram ID внесено до списку адміністраторів у конфігурації).
- У меню «Дисципліни» необхідно створити назву навчального курсу (наприклад, «Об'єктно-орієнтоване програмування») (рис. 4.7).
-

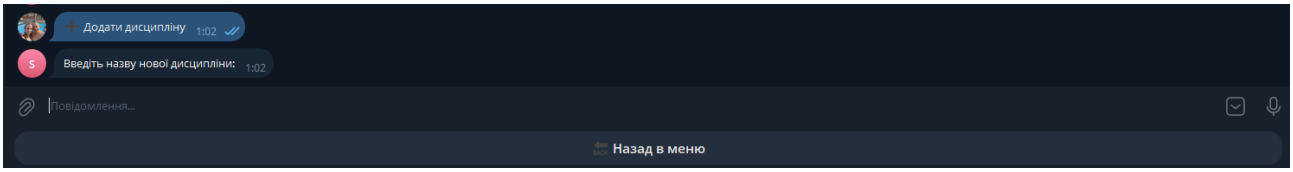


Рисунок 4.7 — Меню створення нової навчальної дисципліни викладачем

Робота з банком питань:

- Для кожного заняття викладач додає питання, вказуючи їх тип: variant (вибір із запропонованих варіантів) або text (відкрита відповідь).
- Обов'язково встановлюється «вага» питання в балах, що дозволяє автоматизувати розрахунок рейтингу.

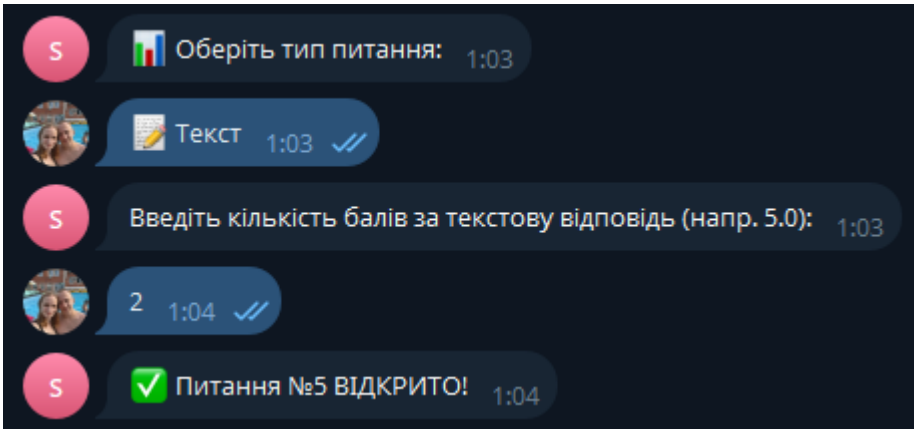


Рисунок 4.8 — Процес конфігурування картки питання в адмін-панелі

3. **Модерація та проведення сесії:**

- У розділі «Заявки» викладач підтверджує реєстрацію студентів, змінюючи їх статус із pending на approved.

- Під час пари викладач обирає команду «Почати пару», після чого бот автоматично надсилає активне питання всім верифікованим студентам.

4. **Генерація аналітичного звіту:**

- Після завершення опитування викладач натискає кнопку «Згенерувати звіт».
- Система формує файл у форматі .xlsx (Excel), який містить ПІБ студентів, їхні відповіді та підсумкові бали за заняття.
-

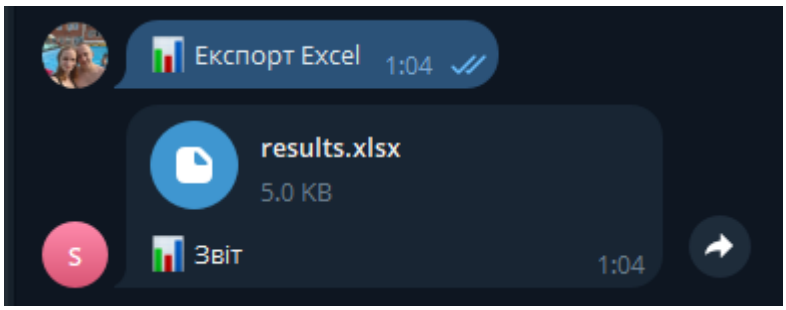


Рисунок 4.9 — Отримання підсумкового Excel-документа від бота

4.3.2. Керівництво для студента

Для студентів взаємодія з ботом спрощена до мінімуму, щоб не відволікати від навчального процесу.

1. **Реєстрація в системі:**

- При першому зверненні до бота студент повинен надати свої ідентифікаційні дані: Прізвище, Ім'я та По батькові.
- З випадаючого списку кнопок необхідно обрати свою академічну групу (наприклад, КН-31).

2. **Участь у тестуванні:**

- Після підтвердження реєстрації викладачем, студент отримує повідомлення про готовність до роботи.
- Коли викладач активує питання, воно з'являється у чаті. Студент обирає варіант відповіді на кнопці або вводить текст у поле повідомлення.
- Важливо: після надання відповіді кнопки зникають, що унеможливлює повторне голосування або зміну вибору (рис. 4.10).

- 

Рисунок 4.10 — Інтерфейс відображення питання на екрані студента

Перегляд статистики:

- За допомогою команди /my_stat студент може переглянути свою поточну успішність у межах обраної дисципліни (рис. 4.11).

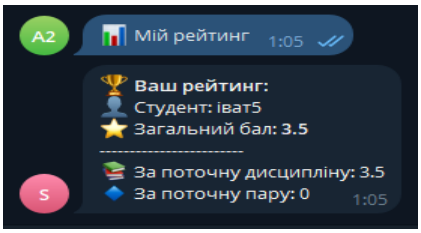


Рисунок 4.11 — Екран індивідуального звіту успішності студента

ВИСНОВОК

У ході виконання кваліфікаційної роботи було спроєктовано та програмно реалізовано систему «StudentQuizBot» — Telegram-бот для автоматизації контролю знань студентів. Робота була спрямована на вирішення актуальної проблеми швидкого та надійного оцінювання успішності під час навчальних занять.

За результатами проведеного дослідження та розробки було досягнуто наступних результатів:

1. **Проаналізовано предметну область** та обґрунтовано доцільність впровадження чат-ботів в освітній процес ПУЕТ як мобільної альтернативи громіздким LMS-системам.
2. **Обґрунтовано вибір технологічного стеку**, який базується на мові програмування Python та асинхронному фреймворку aiogram. Вибір СУБД SQLite дозволив створити автономну систему з високою швидкістю обробки транзакцій та повною відповідністю принципам ACID.
3. **Розроблено архітектуру бази даних**, що включає механізми верифікації користувачів та захисту від академічної недоброчесності через унікальні індекси відповідей.
4. **Реалізовано логіку управління станами (FSM)**, що забезпечує стабільну покрокову реєстрацію студентів та гнучке керування опитуваннями з боку викладача.
5. **Впроваджено модуль аналітичної звітності** на базі бібліотеки openpyxl, що дозволяє викладачу миттєво отримувати результати тестування у форматі Excel, готовому для подальшого використання в університетському документообігу.

6. **Проведено тестування**, результати якого підтвердили високу продуктивність системи: асинхронна архітектура забезпечує обробку запитів від групи студентів (до 50 осіб) без видимих затримок.

Практичне значення роботи полягає у створенні готового програмного продукту, який може бути впроваджений на кафедрі КНІТ для проведення експрес-опитувань, модульних контрольних робіт та збору статистики успішності. Розроблена система демонструє переваги асинхронного програмування та сучасних підходів до проєктування інтерфейсів у середовищі месенджерів.

Перспективи подальшого розвитку проєкту включають інтеграцію з API системи Moodle ПУЕТ для автоматичного перенесення оцінок у віртуальне навчальне середовище, а також додавання підтримки мультимедійних питань (зображень та відео) для розширення можливостей оцінювання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про вищу освіту»: від 01.07.2014 № 1556-VII / Верховна Рада України. URL: **https://zakon.rada.gov.ua/laws/show/1556-18**.
2. Проєктування інформаційних систем: навч. посіб. / В. В. Артеменко та ін. Полтава : ПУЕТ, 2021. 250 с..
3. Гвідо ван Россум, Фред Л. Дрейк-молодший. Мова програмування Python. URL: **https://docs.python.org/3/**.
4. aiogram Documentation. Асинхронний фреймворк для Telegram Bot API. URL: **https://docs.aiogram.dev/en/latest/**.
5. SQLite Documentation. Реляційна база даних SQLite. URL: **https://www.sqlite.org/docs.html**.
6. Openpyxl Documentation. Бібліотека для роботи з Excel у Python. URL: **https://openpyxl.readthedocs.io/**.
7. Що таке діаграма UML? Типи, символи та приклади : вебсайт. URL: **https://www.guru99.com/uk/uml-diagrams.html**.
8. Asyncio — асинхронне програмування в Python. Офіційна документація. URL: **https://docs.python.org/3/library/asyncio.html**.
9. Telegram Bot API. Офіційний посібник для розробників. URL: **https://core.telegram.org/bots/api**.
10. Бублик В. В. Об'єктно-орієнтоване програмування : підручник. Київ : IT-книга, 2019. 432 с. URL: https://itknyga.com.ua/documents/OOP_final.pdf
11. Глибовець М. М., Олецький О. В. Штучний інтелект : підручник для студентів вищих навчальних закладів. Київ : Вид. дім «Києво-Могилянська академія», 2018. 364 с. URL: https://pdf.lib.vntu.edu.ua/books/2020/Glybovec_2002_366.pdf

12. Дацун Н. В., Ситник Л. В. Системи управління базами даних: концептуальні основи та практичне застосування : навч. посіб. Полтава : ПУЕТ, 2022. 198 с.
13. Коротєєва Т. О. Інженерія програмного забезпечення : навч. посіб. Львів : Видавництво Львівської політехніки, 2021. 184 с.
14. Кривоносов Л. М., Федорченко В. М. Асинхронне програмування на базі мови Python: сучасні фреймворки та підходи. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво*. 2023. Вип. 51. С. 45–52.
15. Лавров С. А., Штельмах О. В. Проєктування та розробка подієво-орієнтованих систем із використанням Telegram Bot API. *Український журнал комп'ютерних наук та інформаційних технологій*. 2024. Т. 3, № 2. С. 112–120.
16. Литвин В. В., Пасічник В. В., Яцишин Ю. В. Об'єктно-орієнтоване моделювання програмних систем за допомогою UML : навч. посіб. Львів : Новий Світ-2000, 2020. 320 с.
17. Мельник А. О. Архітектура програмного забезпечення та патерни проєктування : підручник. Харків : Факт, 2022. 276 с.
18. Оліфіров О. В., Ткаченко О. М. Інформаційні системи та технології в освіті : монографія. Київ : Центр навчальної літератури, 2021. 210 с.
19. Пономаренко В. С., Риб'янцев А. А. Обробка та аналіз даних мовою Python : навч. посіб. Харків : ХНЕУ ім. С. Кузнеця, 2023. 164 с.
20. Сидоренко О. П., Василенко В. В. Побудова автоматизованих систем контролю знань в закладах вищої освіти. *Сучасні інформаційні технології та інноваційні методики навчання у підготовці фахівців*. 2025. № 73. С. 89–97.
21. Ткачук М. В., Гамзаєв Х. А. Реляційні бази даних: проектування, оптимізація та адміністрування : навч. посіб. Харків : НТУ «ХПІ», 2020. 240 с.
22. Шаховська Н. Б., Нога Р. Ю. Організація баз даних та знань : підручник. Львів : Новий Світ-2000, 2021. 412 с.

23. Щербаков О. В., Косенко Р. В. Технології створення сучасних чат-ботів для автоматизації бізнес-процесів та навчання. *Вісник ХНУ. Технічні науки*. 2024. № 4. С. 154–161.

24. Бойко А.В. АКТУАЛЬНІ ПИТАННЯ РОЗВИТКУ НАУКИ ТА ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ОСВІТИ У XXI СТОЛІТТІ ТЕЗИ ДОПОВІДЕЙ XLIX Міжнародної наукової конференції студентів та аспірантів (м. Полтава, 23 квітня 2026 року) С. 392 -394

ДОДАТОК А

Main.py

```
import asyncio  # Бібліотека для асинхронного програмування (основа aiogram)
import logging  # Модуль для ведення логів (відстеження помилок та дій)
import sys  # Модуль для взаємодії з операційною системою (вихід з програми)

from aiogram import Bot, Dispatcher  # Головні класи: Бот (зв'язок) та Диспетчер (обробка)
from aiogram.fsm.storage.memory import MemoryStorage  # Тимчасове сховище для станів (FSM)

import config  # Твій файл з токеном та ID адміна
import database.db_requests as db  # Модуль для роботи з базою даних SQLite
from handlers import user_handlers, admin_handlers  # Твої файли з логікою бота

# ================= НАЛАШТУВАННЯ ЛОГУВАННЯ ================
# Це дозволяє бачити в консолі, хто написав боту, які виникають помилки та статус з'єднання
logging.basicConfig(
    level=logging.INFO,  # Рівень логування (INFO показує основні події)
    format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",  # Формат дати та повідомлення
)

# ================= ГОЛОВНА АСИНХРОННА ФУНКЦІЯ
```

```
================
async def main():
    # 1. ПІДГОТОВКА БАЗИ ДАНИХ
    # Викликаємо функцію створення таблиць (users, groups, disciplines тощо)
    # Якщо база вже є, вона просто перевірить наявність таблиць
    await db.create_tables()

    # 2. ІНІЦІАЛІЗАЦІЯ ОБ'ЄКТІВ
    # Створюємо екземпляр бота, передаючи токен з конфігу
    bot = Bot(token=config.BOT_TOKEN)

    # Створюємо диспетчер. MemoryStorage означає, що дані станів (FSM)
    # зникнуть, якщо вимкнути бота (що нормально для лекційних квізів)
    dp = Dispatcher(storage=MemoryStorage())

    # 3. РЕЄСТРАЦІЯ ОБРОБНИКІВ (ROUTERS)
    # Порядок має значення! Роутер адміна ставимо першим, щоб він мав
пріоритет.
    # Диспетчер буде спочатку перевіряти повідомлення за фільтрами адміна.
    dp.include_routers(
        admin_handlers.router,
        user_handlers.router
    )

    # 4. ОБРОБКА ЧЕРГИ ПОВІДОМЛЕНЬ
    # delete_webhook видаляє старі запити. drop_pending_updates=True каже боту:
    # "Не відповідай на повідомлення, які тобі писали, поки ти був офлайн"
    await bot.delete_webhook(drop_pending_updates=True)

    # Виводимо в консоль повідомлення про успішний старт
```

```
    print("🚀 StudentQuizBot запущено!")
    print("📍 Локація бази даних: quiz_bot_v5.db")
    print("⌨️ Натисніть Ctrl+C для зупинки бота")

    # 5. ЗАПУСК ОПИТУВАННЯ (POLLING)
    # Бот починає постійно запитувати сервери Telegram: "Чи є нові повідомлення?"
    try:
        await dp.start_polling(bot)
    finally:
        # Цей блок виконається при зупинці (наприклад, Ctrl+C)
        # Важливо закрити сесію, щоб не було "витоку" ресурсів
        await bot.session.close()

# ================== ТОЧКА ВХОДУ В ПРОГРАМУ ==================
if __name__ == "__main__":
    try:
        # Запускаємо асинхронний цикл подій для функції main
        asyncio.run(main())
    except (KeyboardInterrupt, SystemExit):
        # Якщо користувач натиснув Ctrl+C, виходимо без помилок у консолі
        print("\n🛑 Бот зупинений вручну.")
        sys.exit(0)
```

db_requests.py

```
import aiosqlite  # Бібліотека для асинхронної роботи з SQLite (не блокує роботу бота)

DB_NAME = 'quiz_bot_v6.db'  # Назва файлу бази даних
```

```
async def create_tables():
    """Створює всі необхідні таблиці в базі даних SQLite, якщо вони не існують"""
    async with aiosqlite.connect(DB_NAME) as db:
        # 1. ТАБЛИЦЯ ГРУП (зберігає унікальні назви академічних груп)
        await db.execute('''CREATE TABLE IF NOT EXISTS groups (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            name TEXT UNIQUE
                        )''')

        # 2. ТАБЛИЦЯ КОРИСТУВАЧІВ (Студентів)
        await db.execute('''CREATE TABLE IF NOT EXISTS users (
            user_id INTEGER PRIMARY KEY,
            full_name TEXT,
            group_name TEXT,
            status TEXT DEFAULT 'pending', -- Статус 'pending' (очікує) або 'approved' (підтверджено)
            FOREIGN KEY (group_name) REFERENCES groups (name))''')

        # 3. ТАБЛИЦЯ ДИСЦИПЛІН (Предмети, прив'язані до конкретних груп)
        await db.execute('''CREATE TABLE IF NOT EXISTS disciplines (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            group_name TEXT,
            is_active BOOLEAN DEFAULT 1, -- 1 - предмет активний, 0 - в архіві
            FOREIGN KEY (group_name) REFERENCES groups (name))''')

        # 4. ТАБЛИЦЯ ЗАНЯТЬ (Пар) - фіксує початок і кінець конкретної пари
```

```
        await db.execute('''CREATE TABLE IF NOT EXISTS lessons (
            id INTEGER PRIMARY  KEY AUTOINCREMENT,
            discipline_id INTEGER,
            start_time DATETIME DEFAULT CURRENT_TIMESTAMP,
            end_time DATETIME,
            FOREIGN KEY (discipline_id) REFERENCES disciplines (id))''')

        # 5. ТАБЛИЦЯ ПИТАНЬ (Питання, що створюються під час пари)
        await db.execute('''CREATE TABLE IF NOT EXISTS questions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            lesson_id INTEGER,
            question_type TEXT, -- 'variant' (тест) або 'text' (відкрита відповідь)
            variants_count INTEGER,
            weight REAL DEFAULT 1.0,  -- Кількість балів за правильну відповідь
            correct_answer TEXT,
            is_open BOOLEAN DEFAULT 1,    -- Питання активне, поки викладач
його не закрив
            FOREIGN KEY (lesson_id) REFERENCES lessons (id))''')

        # 6. ТАБЛИЦЯ ВІДПОВІДЕЙ (Відповіді студентів на конкретні питання)
        await db.execute('''CREATE TABLE IF NOT EXISTS answers (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
            question_id INTEGER,
            answer_text TEXT,
            is_correct BOOLEAN, -- Результат перевірки: правильно/неправильно
            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users (user_id),
            FOREIGN KEY (question_id) REFERENCES questions (id),
            UNIQUE (user_id, question_id))''')
```

```
        await db.commit()  # Зберігаємо всі зміни в структурі БД

# --- ФУНКЦІЇ ДИСЦИПЛІН ---

async def create_discipline(name, group_name):
    """Створює нову дисципліну та очищає назви від зайвих пробілів"""
    async with aiosqlite.connect(DB_NAME) as db:
        try:
            await db.execute(
                'INSERT INTO disciplines (name, group_name) VALUES (?, ?)',
                (name.strip(), group_name.strip())
            )
            await db.commit()
            return True
        except Exception as e:
            print(f"Помилка створення дисципліни: {e}")
            return False

async def get_disciplines():
    """Отримує список усіх активних дисциплін"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute('SELECT id, name, group_name FROM disciplines
WHERE is_active = 1') as cursor:
            return await cursor.fetchall()  # Повертає список кортежів

async def get_all_disciplines_raw():
    """Повертає всі дисципліни незалежно від статусу (активні/архівні)"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute("SELECT id, name, group_name, is_active FROM
```

```
disciplines") as cursor:
            return await cursor.fetchall()

async def get_disciplines_by_group(group_name):
    """Фільтрує предмети за назвою групи (використовується студентами)"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute(
                'SELECT id, name FROM disciplines WHERE group_name = ? AND
is_active = 1',
                (group_name,)
        ) as cursor:
            return await cursor.fetchall()

async def close_discipline_db(discipline_id):
    """Відправляє дисципліну в архів (is_active = 0)"""
    async with aiosqlite.connect(DB_NAME) as db:
        await db.execute("UPDATE disciplines SET is_active = 0 WHERE id = ?",
(discipline_id,))
        await db.commit()

# --- ФУНКЦІЇ ПАР (LESSONS) ---

async def check_active_lesson():
    """Перевіряє, чи є зараз пара, у якої не встановлено час завершення (end_time
IS NULL)"""
    async with aiosqlite.connect(DB_NAME) as db:
        sql = '''
            SELECT l.id, l.discipline_id, d.group_name
            FROM lessons l
                    JOIN disciplines d ON l.discipline_id = d.id -- Об'єднуємо з
```

```
таблицею предметів для отримання групи
            WHERE l.end_time IS NULL LIMIT 1 \
            '''
        async with db.execute(sql) as cursor:
            return await cursor.fetchone()

async def start_lesson(discipline_id):
    """Створює запис про початок нової пари"""
    async with aiosqlite.connect(DB_NAME) as db:
        cursor = await db.execute('INSERT INTO lessons (discipline_id) VALUES (?)',
(discipline_id,))
        await db.commit()
        return cursor.lastrowid  # Повертає ID створеної пари

async def end_lesson_db(lesson_id):
    """Завершує пару та автоматично закриває всі відкриті питання в ній"""
    async with aiosqlite.connect(DB_NAME) as db:
        await db.execute("UPDATE lessons SET end_time = CURRENT_TIMESTAMP
WHERE id = ?", (lesson_id,))
        await db.execute("UPDATE questions SET is_open = 0 WHERE lesson_id = ?",
(lesson_id,))
        await db.commit()

# --- ФУНКЦІЇ ПИТАНЬ ---

async def create_question(lesson_id, q_type, variants, weight):
    """Створює нове питання для студентів"""
    async with aiosqlite.connect(DB_NAME) as db:
        cursor = await db.execute(
            'INSERT INTO questions (lesson_id, question_type, variants_count, weight,
```

```
is_open) VALUES (?, ?, ?, ?, 1)',
            (lesson_id, q_type, variants, weight)
        )
        await db.commit()
        return cursor.lastrowid

async def get_active_question():
    """Шукає останнє відкрите викладачем питання"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute(
                'SELECT id, lesson_id, question_type, variants_count, weight FROM
questions WHERE is_open = 1 ORDER BY id DESC LIMIT 1') as cursor:
            return await cursor.fetchone()

async def get_last_question_id(lesson_id):
    """Знаходить ID останнього питання конкретної пари (для перевірки
відповідей)"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute('SELECT id FROM questions WHERE lesson_id = ?
ORDER BY id DESC LIMIT 1',
                              (lesson_id,)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res else None

async def close_question_db(question_id, correct_answer):
    """Закриває питання та автоматично порівнює відповіді студентів з
еталоном"""
    reference = str(correct_answer).strip().lower()  # Приводимо еталон до нижнього
регістру
    async with aiosqlite.connect(DB_NAME) as db:
```

```
        # Оновлюємо статус питання в БД
        await db.execute('UPDATE questions SET is_open = 0, correct_answer = ?
WHERE id = ?',
                         (correct_answer, question_id))

        # Вибираємо всі відповіді студентів
        async with db.execute('SELECT id, answer_text FROM answers WHERE
question_id = ?', (question_id,)) as cursor:
            answers = await cursor.fetchall()

        # Почергово перевіряємо кожну відповідь через Python
        for ans_id, ans_text in answers:
            # Якщо текст збігається (без урахування регістру та пробілів) -
зараховуємо
            is_correct = 1 if str(ans_text).strip().lower() == reference else 0
            await db.execute('UPDATE answers SET is_correct = ? WHERE id = ?',
(is_correct, ans_id))

        await db.commit()

async def get_question_correct_answer(question_id):
    """Дістає правильну відповідь з бази для розсилки результатів"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute("SELECT correct_answer FROM questions WHERE id =
?", (question_id,)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res else "Не вказано"

# --- РОБОТА ЗІ СТУДЕНТАМИ ТА ВІДПОВІДЯМИ ---
```

```
async def add_user(user_id, full_name, group_name):
    """Реєструє студента або оновлює його дані (INSERT OR REPLACE)"""
    async with aiosqlite.connect(DB_NAME) as db:
        await db.execute('''INSERT OR REPLACE INTO users (user_id, full_name,
group_name, status)
                            VALUES (?, ?, ?, 'pending')''', (user_id, full_name, group_name))
        await db.commit()

async def get_user(user_id):
    """Отримує профіль користувача за його Telegram ID"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute('SELECT user_id, full_name, group_name, status FROM
users WHERE user_id = ?',
                              (user_id,)) as cursor:
            return await cursor.fetchone()


async def approve_user(user_id):
    """Підтверджує реєстрацію студента (виконується викладачем)"""
    async with aiosqlite.connect(DB_NAME) as db:
        await db.execute("UPDATE users SET status = 'approved' WHERE user_id =
?", (user_id,))
        await db.commit()

async def get_pending_users():
    """Показує викладачу список студентів, які чекають на підтвердження"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute("SELECT user_id, full_name, group_name FROM users
WHERE status = 'pending'") as cursor:
            return await cursor.fetchall()
```

```
async def get_all_users_raw():
    """Список усіх студентів для адмін-панелі"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute("SELECT user_id, full_name, group_name, status FROM
users") as cursor:
            return await cursor.fetchall()

async def get_all_student_ids():
    """Повертає список ID всіх схвалених студентів (для загальних розсилок)"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute("SELECT user_id FROM users WHERE status =
'approved'") as cursor:
            rows = await cursor.fetchall()
            return [row[0] for row in rows]

async def save_answer(user_id, question_id, answer):
    """Зберігає відповідь студента в базу"""
    async with aiosqlite.connect(DB_NAME) as db:
        try:
            await db.execute('INSERT INTO answers (user_id, question_id, answer_text)
VALUES (?, ?, ?)',
                             (user_id, question_id, answer))
            await db.commit()
            return True
        except:
            return False  # Поверне False, якщо студент вже відповідав (UNIQUE
constraint)

async def check_if_answered(user_id, question_id):
```

```
    """Перевіряє, чи відповідала вже людина на це питання"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute("SELECT id FROM answers WHERE user_id = ? AND question_id = ?",
                              (user_id, question_id)) as cursor:
            return await cursor.fetchone() is not None

async def update_answer_manual(answer_id, is_correct: bool):
    """Дозволяє викладачу вручну змінити статус відповіді (наприклад, якщо студент опечатався)"""
    async with aiosqlite.connect(DB_NAME) as db:
        await db.execute("UPDATE answers SET is_correct = ? WHERE id = ?", (1 if is_correct else 0, answer_id))
        await db.commit()

async def get_answers_by_question(question_id):
    """Збирає всі відповіді на конкретне питання разом із ПІБ студентів"""
    async with aiosqlite.connect(DB_NAME) as db:
        sql = 'SELECT a.id, u.full_name, a.answer_text, a.is_correct FROM answers a JOIN users u ON a.user_id = u.user_id WHERE a.question_id = ?'
        async with db.execute(sql, (question_id,)) as cursor:
            return await cursor.fetchall()

# --- ФУНКЦІЇ ГРУП ---

async def get_all_groups():
    """Дістає назви всіх академічних груп для клавіатур"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute('SELECT name FROM groups ORDER BY name') as cursor:
```

```
            rows = await cursor.fetchall()
            return [row[0] for row in rows]

async def add_group(name):
    """Додає нову групу в базу даних"""
    async with aiosqlite.connect(DB_NAME) as db:
        try:
            await db.execute('INSERT INTO groups (name) VALUES (?)',
(name.strip(),))
            await db.commit()
            return True
        except:
            return False  # Якщо назва вже є (UNIQUE)

async def delete_group(name):
    """Видаляє групу та обнуляє її у студентів, які в ній були"""
    async with aiosqlite.connect(DB_NAME) as db:
        try:
            await db.execute('DELETE FROM groups WHERE name = ?', (name,))
            await db.execute('UPDATE users SET group_name = NULL WHERE
group_name = ?', (name,))
            await db.commit()
            return True
        except Exception as e:
            print(f"Помилка видалення групи: {e}")
            return False

# --- РЕЙТИНГИ ---

async def get_student_total_score(user_id):
```

```
    """Рахує суму балів студента за всі правильні відповіді за весь час"""
    async with aiosqlite.connect(DB_NAME) as db:
        sql = 'SELECT SUM(q.weight) FROM answers a JOIN questions q ON
a.question_id = q.id WHERE a.user_id = ? AND a.is_correct = 1'
        async with db.execute(sql, (user_id,)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res and res[0] else 0


async def get_student_discipline_score(user_id, discipline_id):
    """Рахує бали студента в межах конкретного предмета"""
    async with aiosqlite.connect(DB_NAME) as db:
        sql = 'SELECT SUM(q.weight) FROM answers a JOIN questions q ON
a.question_id = q.id JOIN lessons l ON q.lesson_id = l.id WHERE a.user_id = ? AND
l.discipline_id = ? AND a.is_correct = 1'
        async with db.execute(sql, (user_id, discipline_id)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res and res[0] else 0

async def get_student_lesson_score(user_id, lesson_id):
    """Рахує бали студента за одну конкретну пару"""
    async with aiosqlite.connect(DB_NAME) as db:
        sql = 'SELECT SUM(q.weight) FROM answers a JOIN questions q ON
a.question_id = q.id WHERE a.user_id = ? AND q.lesson_id = ? AND a.is_correct =
1'
        async with db.execute(sql, (user_id, lesson_id)) as cursor:
            res = await cursor.fetchone()
            return res[0] if res and res[0] else 0

async def get_all_results():
```

```
    """Формує повний звіт для Excel-файлу (ПІБ, група, предмет, дата, бали)"""
    async with aiosqlite.connect(DB_NAME) as db:
        sql = '''
            SELECT u.full_name, \
                u.group_name, \
                d.name, \
                l.start_time, \
                q.question_type,
                CASE WHEN a.is_correct = 1 THEN q.weight ELSE 0 END as
earned_score
            FROM answers a
                JOIN users u ON a.user_id = u.user_id
                JOIN questions q ON a.question_id = q.id
                JOIN lessons l ON q.lesson_id = l.id
                JOIN disciplines d ON l.discipline_id = d.id
            ORDER BY l.start_time DESC \
            '''
        async with db.execute(sql) as cursor:
            return await cursor.fetchall()

async def get_student_ids_by_group(group_name):
    """Знаходить Telegram ID студентів конкретної групи для цільової розсилки
відповідей"""
    async with aiosqlite.connect(DB_NAME) as db:
        async with db.execute(
                "SELECT user_id FROM users WHERE group_name = ? AND status =
'approved'",
                (group_name,)
        ) as cursor:
```

```
        rows = await cursor.fetchall()
        return [row[0] for row in rows]
```

admin_handlers.py

```
from aiogram import Router, F
from aiogram.types import Message, CallbackQuery, InlineKeyboardMarkup,
InlineKeyboardButton, KeyboardButton, BufferedInputFile, ReplyKeyboardMarkup

from aiogram.fsm.context import FSMContext
from aiogram.filters import Command, StateFilter
from aiogram.utils.keyboard import ReplyKeyboardBuilder, InlineKeyboardBuilder

import config  # Налаштування (токен, ID адмінів)
import database.db_requests as db  # Запити до бази даних
import keyboards.reply as kb  # Готові Reply-клавіатури
import keyboards.builders as kb_build  # Динамічні клавіатури
from states.user_states import TeacherState  # Стани викладача (FSM)
from utils.excel_export import create_excel_report  # Модуль для створення Excel-
звітів

# Ініціалізація роутера для admin-хендлерів
router = Router()

# Чиста локальна кнопка повернення для кроків введення тексту (замість
ReplyKeyboardRemove)
local_back_kb = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="🔙 Назад в меню")]
], resize_keyboard=True)

# ================ ГОЛОВНЕ МЕНЮ ТА НАВІГАЦІЯ
```

```
================

# Обробка команди /start для адміністраторів
@router.message(Command("start"), F.from_user.id.in_(config.ADMIN_IDS),
StateFilter("*"))
async def cmd_start_admin(message: Message, state: FSMContext):
    await state.clear()  # Скидаємо будь-який активний стан користувача
    await message.answer("👋 Вітаю, Викладачу! Оберіть розділ:",
reply_markup=kb.teacher_main)
    await state.set_state(TeacherState.menu)  # Встановлюємо стан головного меню

# Універсальна кнопка повернення в головне меню
@router.message(F.text == "🔙 Назад в меню", StateFilter("*"))
@router.message(F.text == "🔙 Скасувати", StateFilter("*"))
async def back_to_main(message: Message, state: FSMContext):
    await state.clear()
    await message.answer("Головне меню:", reply_markup=kb.teacher_main)
    await state.set_state(TeacherState.menu)

# ================ ГІЛКА "ДИСЦИПЛІНИ" ================

# Вхід у підменю дисциплін
@router.message(F.text == "📚 Дисципліни", StateFilter("*"))
async def menu_disciplines(message: Message, state: FSMContext):
    await message.answer("Керування дисциплінами:",
reply_markup=kb.disciplines_sub)
    await state.set_state(TeacherState.disciplines_menu)

# Показ списку всіх дисциплін (включаючи архівні)
```

```
@router.message(F.text == "📚 Список дисциплін", TeacherState.disciplines_menu)
async def list_all_discs(message: Message):
    discs = await db.get_all_disciplines_raw()  # Отримуємо дані з БД
    if not discs:
        return await message.answer("Дисциплін ще не створено.")

    text = "📂 <b>Список усіх дисциплін:</b>\n\n"
    for d in discs:
        status = "✅ Активна" if d[3] else "📁 Архів"  # d[3] - це поле is_active
        text += f"ID: {d[0]} | <b>{d[1]}</b> ({d[2]})\nСтатус: {status}\n"
        text += "----------------------\n"
    await message.answer(text, parse_mode="HTML")

# Початок процесу додавання нової дисципліни
@router.message(F.text == "➕ Додати дисципліну",
TeacherState.disciplines_menu)
async def add_disc_start(message: Message, state: FSMContext):
    groups = await db.get_all_groups()
    if not groups:
        # Не дозволяємо додавати предмет, якщо немає жодної групи
        await message.answer("❌ <b>Помилка!</b> Спочатку створіть групу в
меню '📁 Керування групами'.",
                             reply_markup=kb.disciplines_sub, parse_mode="HTML")
        return
    await message.answer("Введіть назву нової дисципліни:",
reply_markup=local_back_kb)
    await state.set_state(TeacherState.create_discipline_name)
```

```
# Отримання назви та пропозиція обрати групу
@router.message(TeacherState.create_discipline_name)
async def add_disc_select_group(message: Message, state: FSMContext):
    if message.text in ["🔙 Назад в меню", "🔙 Скасувати"]:
        return await back_to_main(message, state)

    await state.update_data(disc_name=message.text)  # Зберігаємо назву в пам'ять FSM
    groups = await db.get_all_groups()
    # Динамічна клавіатура зі списком існуючих груп
    await message.answer(f"Оберіть групу для '{message.text}':", reply_markup=kb_build.get_groups_reply_kb(groups))
    await state.set_state(TeacherState.create_discipline_group)

# Фінальне збереження дисципліни в БД
@router.message(TeacherState.create_discipline_group)
async def add_disc_final(message: Message, state: FSMContext):
    if message.text in ["🔙 Назад в меню", "🔙 Скасувати"]:
        return await back_to_main(message, state)

    data = await state.get_data()  # Дістаємо назву, яку зберегли раніше
    if await db.create_discipline(data.get('disc_name'), message.text):
        await message.answer(f"✅ Дисципліну успішно створено!", reply_markup=kb.disciplines_sub)
    await state.set_state(TeacherState.disciplines_menu)

# Архівування (закриття) дисципліни з перевіркою активної пари
@router.message(F.text == "🔒 Закрити дисципліну (Семестр)", TeacherState.disciplines_menu)
```

```
async def close_disc_list(message: Message):
    disciplines = await db.get_disciplines()
    if not disciplines:
        return await message.answer("Немає активних дисциплін.")

    builder = InlineKeyboardBuilder()  # Використовуємо інлайн-кнопки (під текстом)
    for d in disciplines:
        builder.add(InlineKeyboardButton(text=f"❌ {d[1]} ({d[2]})", callback_data=f"close_disc_{d[0]}"))
    builder.adjust(1)
    await message.answer("Оберіть дисципліну для архівації:", reply_markup=builder.as_markup())

# Обробка натискання на інлайн-кнопку архівування
@router.callback_query(F.data.startswith("close_disc_"))
async def close_disc_confirm(callback: CallbackQuery):
    disc_id = int(callback.data.split("_")[2])  # Дістаємо ID з callback_data

    # ПЕРЕВІРКА: чи не йде зараз пара за цим предметом
    active_lesson = await db.check_active_lesson()
    if active_lesson and int(active_lesson[1]) == disc_id:
        await callback.message.answer(
            "❌ <b>Не вдалося закрити дисципліну!</b>\n\nЗа цим курсом прямо зараз проводиться пара. "
            "Спочатку закінчіть її кнопкою  <b>Закінчити пару</b>.",
            reply_markup=kb.disciplines_sub,
            parse_mode="HTML"
        )
```

```
        await callback.message.delete()
        return await callback.answer()

    await db.close_discipline_db(disc_id)
    await callback.message.edit_text("✅ Дисципліну перенесено в архів.")

# ================= ГІЛКА "КЕРУВАННЯ ГРУПАМИ" ================

# Список існуючих груп
@router.message(F.text == "💼 Керування групами", StateFilter("*"))
async def menu_groups(message: Message, state: FSMContext):
    groups = await db.get_all_groups()
    text = "💼 <b>Меню керування групами</b>\n\n"
    text += "Ваші групи:\n" + "\n".join([f"• <code>{g}</code>" for g in groups]) if groups else "⚠️ Груп немає."
    await message.answer(text, reply_markup=kb.groups_management_kb, parse_mode="HTML")
    await state.set_state(TeacherState.managing_groups)

# Створення нової групи
@router.message(F.text == "➕ Додати групу", TeacherState.managing_groups)
async def add_group_start(message: Message, state: FSMContext):
    await message.answer("Введіть назву нової групи (напр. КН-21):", reply_markup=local_back_kb)
    await state.set_state(TeacherState.adding_group)


@router.message(TeacherState.adding_group)
```

```
async def add_group_finish(message: Message, state: FSMContext):
    if message.text in ["🔙 Назад в меню", "🔙 Скасувати"]:
        return await back_to_main(message, state)

    if await db.add_group(message.text):
        await message.answer(f"✅ Групу додано!",
reply_markup=kb.groups_management_kb)
    else:
        await message.answer("❌ Така група вже існує.")
    await state.set_state(TeacherState.managing_groups)

# ================ ГІЛКА "СТУДЕНТИ" ================

# Перегляд списку та балів студентів
@router.message(F.text == "    Студенти", StateFilter("*"))
async def menu_students(message: Message, state: FSMContext):
    await message.answer("Керування студентами:", reply_markup=kb.students_sub)
    await state.set_state(TeacherState.students_menu)

@router.message(F.text == "    Список студентів", TeacherState.students_menu)
async def list_all_students(message: Message):
    users = await db.get_all_users_raw()
    if not users: return await message.answer("Студентів немає.")
    for u in users:
        score = await db.get_student_total_score(u[0])  # Рахуємо суму балів студента
        status = "✅" if u[3] == 'approved' else "⏳"  # Відображення статусу
        await message.answer(f"👤 <b>{u[1]}</b> ({u[2]})\nСтатус: {status}\n⭐
Бали: {score}", parse_mode="HTML")
```

```
# Генерація звіту в Excel
@router.message(F.text == "📊 Експорт Excel", TeacherState.students_menu)
async def export_excel_btn(message: Message):
    data = await db.get_all_results()  # Отримання зведених даних
    if not data: return await message.answer("Немає даних.")
    bio = await create_excel_report(data)  # Створення файлу в оперативній пам'яті
    await message.answer_document(
        BufferedInputFile(bio.read(), filename="results.xlsx"),
        caption="📊 Звіт"
    )

# Підтвердження нових користувачів
@router.message(F.text == "✅ Підтвердити реєстрацію",
TeacherState.students_menu)
async def approve_users_list(message: Message):
    users = await db.get_pending_users()  # Отримуємо тих, у кого статус 'pending'
    if not users: return await message.answer("Немає нових заявок.")
    for u in users:
        markup = InlineKeyboardMarkup(
            inline_keyboard=[[InlineKeyboardButton(text="✅ OK",
callback_data=f"approve_{u[0]}")]])
        await message.answer(f"👤 {u[1]} ({u[2]})", reply_markup=markup)


@router.callback_query(F.data.startswith("approve_"))
async def approve_confirm(callback: CallbackQuery):
    await db.approve_user(int(callback.data.split("_")[1]))
    await callback.message.edit_text(f"✅ Підтверджено!")
```

```
# Вхід у меню питань з головної панелі викладача
@router.message(F.text == "❓ Питання", StateFilter("*"))
async def menu_questions(message: Message, state: FSMContext):
    """Перевіряє, чи йде пара. Якщо так — відкриває підменю питань, якщо ні — блокує доступ."""
    await state.clear()
    active_lesson = await db.check_active_lesson()  # Перевіряємо статус пари в БД

    if active_lesson:
        # Зберігаємо ID пари у FSM-контекст, щоб квізи прив'язувалися до цієї лекції
        await state.update_data(current_lesson_id=active_lesson[0])

        await message.answer(
            "❓ <b>Панель управління експрес-тестами (Квізами):</b>",
            reply_markup=kb.questions_sub,
            parse_mode="HTML"
        )
        await state.set_state(TeacherState.questions_menu)
    else:
        # Якщо активної пари немає, не пускаємо в меню питань
        await message.answer(
            "❌ <b>Доступ заблоковано!</b>\n\n"
            "Наразі немає активних пар. Перейдіть у розділ 📚 <b>Дисципліни</b> та натисніть 🎓 <b>Почати пару</b>.",
            reply_markup=kb.teacher_main,
            parse_mode="HTML"
        )
```

```
# ================= ГІЛКА "ПАРУ ТА ПИТАННЯ" =================

# Вибір предмета для початку лекції
@router.message(F.text == "🎓 Почати пару", StateFilter("*"))
async def start_lesson_menu(message: Message):
    active = await db.check_active_lesson()
    if active:
        return await message.answer(f"⚠️ Пара вже триває для групи
<b>{active[2]}</b>!", parse_mode="HTML")

    async with db.aiosqlite.connect(db.DB_NAME) as connection:
        async with connection.execute("SELECT id, name, group_name FROM
disciplines WHERE is_active = 1") as cursor:
            discs = await cursor.fetchall()

    if not discs: return await message.answer("❌ Немає активних дисциплін.")

    builder = InlineKeyboardBuilder()
    for d in discs:
        builder.add(InlineKeyboardButton(text=f"{d[1]} ({d[2]})",
callback_data=f"start_lesson_{d[0]}"))
    builder.adjust(1)
    await message.answer("Оберіть дисципліну для початку:",
reply_markup=builder.as_markup())

# Фіксація початку пари в БД
@router.callback_query(F.data.startswith("start_lesson_"), StateFilter("*"))
async def start_lesson_confirm(callback: CallbackQuery, state: FSMContext):
    disc_id = int(callback.data.split("_")[2])
```

```
    lesson_id = await db.start_lesson(disc_id)  # Запис у таблицю lessons
    await state.update_data(current_lesson_id=lesson_id)  # Зберігаємо ID пари в сесію
    await callback.message.edit_text("✅ Пару розпочато!")
    await callback.message.answer("Керуйте питаннями:", reply_markup=kb.questions_sub)
    await state.set_state(TeacherState.questions_menu)

# Завершення лекції
@router.message(F.text == "    Закінчити пару", StateFilter("*"))
async def end_lesson(message: Message, state: FSMContext):
    active = await db.check_active_lesson()
    if not active: return await message.answer("Зараз немає пар.")
    await db.end_lesson_db(active[0])  # Проставляємо end_time в БД
    await state.clear()
    await message.answer(f"🏁 Пару закінчено.", reply_markup=kb.teacher_main)

# Створення нового питання під час пари
@router.message(F.text == "📢 Відкрити питання", TeacherState.questions_menu)
async def open_q_start(message: Message):
    # Тепер тут суворо викликається твоя кастомна kb.question_type_kb з кнопкою "Назад в меню"
    await message.answer("📊 Оберіть тип питання:", reply_markup=kb.question_type_kb)

# Вибір типу: Тест (варіанти)
@router.message(F.text == "🔠 Варіанти", TeacherState.questions_menu)
async def type_variant(message: Message, state: FSMContext):
    await state.update_data(q_type='variant')
```

```
    await message.answer(
        "Введіть параметри тесту в форматі: <code>[Кількість_варіантів] [Бали]</code>\n\nПриклад: `4 2.5`",
        reply_markup=local_back_kb, parse_mode="HTML")
    await state.set_state(TeacherState.setting_question)

# Вибір типу: Текстова відповідь
@router.message(F.text == "📝 Текст", TeacherState.questions_menu)
async def type_text(message: Message, state: FSMContext):
    await state.update_data(q_type='text')
    await message.answer("Введіть кількість балів за текстову відповідь (напр. 5.0):", reply_markup=local_back_kb)
    await state.set_state(TeacherState.setting_question)

# Збереження параметрів питання та його активація
@router.message(TeacherState.setting_question)
async def process_question_config(message: Message, state: FSMContext):
    if message.text in ["🔙 Назад в меню", "🔙 Скасувати"]:
        return await back_to_main(message, state)
    try:
        data = await state.get_data()
        parts = message.text.replace(',', '.').split()  # Обробка крапки/коми
        if data['q_type'] == 'variant':
            variants, weight = int(parts[0]), float(parts[1])
        else:
            variants, weight = 0, float(parts[0])
        q_id = await db.create_question(data.get('current_lesson_id'), data['q_type'], variants, weight)
        await state.update_data(current_question_id=q_id)
```

```
        await message.answer(f"✅ Питання №{q_id} ВІДКРИТО!",
reply_markup=kb.questions_sub)
        await state.set_state(TeacherState.questions_menu)
    except:
        await message.answer("⚠️ Помилка формату! Спробуйте ще раз:")

# Закриття прийому відповідей -> Очікуємо еталон
@router.message(F.text == "❌ Закрити питання", TeacherState.questions_menu)
async def close_q_start(message: Message, state: FSMContext):
    active_q = await db.get_active_question()
    if not active_q: return await message.answer("Немає активних питань.")
    await state.update_data(current_question_id=active_q[0])
    await message.answer("Введіть ПРАВИЛЬНУ (еталонну) відповідь для
перевірки системи:", reply_markup=local_back_kb)
    await state.set_state(TeacherState.waiting_for_correct_answer)


# Отримання еталону -> Перевід до вибору показу результатів через yes_no_kb
@router.message(TeacherState.waiting_for_correct_answer)
async def close_q_finish(message: Message, state: FSMContext):
    if message.text in ["🔙 Назад в меню", "🔙 Скасувати"]:
        return await back_to_main(message, state)

    data = await state.get_data()
    q_id = data.get('current_question_id')
    await db.close_question_db(q_id, message.text)  # Фіксуємо еталон та
порівнюємо відповіді в БД

    await message.answer(
```

```
        "Прийом відповідей зупинено. Бажаєте надіслати правильну відповідь студентам для самоперевірки?",
        reply_markup=kb.yes_no_kb)
    await state.set_state(TeacherState.confirm_visibility)

# Цільова розсилка правильної відповіді студентам активної групи
@router.message(TeacherState.confirm_visibility)
async def show_results_decision(message: Message, state: FSMContext):
    if message.text in ["🔙 Назад в меню", "🔙 Скасувати"]:
        return await back_to_main(message, state)

    if message.text == "  □ Показати студентам":
        data = await state.get_data()
        q_id = data.get('current_question_id')

        active_lesson = await db.check_active_lesson()
        if not active_lesson:
            return await message.answer("❌ Помилка: пару вже закінчено.", reply_markup=kb.questions_sub)

        lesson_group = active_lesson[2]  # Отримуємо цільову групу
        correct_ans = await db.get_question_correct_answer(q_id)
        students = await db.get_student_ids_by_group(lesson_group)  # Тільки студенти цієї групи

        await message.answer(f"📢 Надсилаю результати групі <b>{lesson_group}</b>...", parse_mode="HTML")

        for s_id in students:
```

```
            try:
                await message.bot.send_message(
                    chat_id=s_id,
                    text=f"🏁 <b>Питання закрите!</b>\n✅ Правильна відповідь:
<b>{correct_ans}</b>",
                    parse_mode="HTML"
                )
            except Exception:
                pass
        await message.answer(f"✅ Результати надіслані.",
reply_markup=kb.questions_sub)
    else:
        await message.answer("Збережено без розсилки.",
reply_markup=kb.questions_sub)

    await state.set_state(TeacherState.questions_menu)


# Ручна перевірка або зміна оцінок
@router.message(F.text == "📝 Перевірка відповідей",
TeacherState.questions_menu)
async def review_answers(message: Message):
    active_lesson = await db.check_active_lesson()
    if not active_lesson: return await message.answer("Немає пари.")
    q_id = await db.get_last_question_id(active_lesson[0])
    if not q_id: return await message.answer("Питань не було.")
    answers = await db.get_answers_by_question(q_id)
    if not answers: return await message.answer("Відповідей немає.")
    for ans in answers:
```

```
        status = "✅" if ans[3] else "❌"
        new_val = 1 if not ans[3] else 0
        markup = InlineKeyboardMarkup(
            inline_keyboard=[
                [InlineKeyboardButton(text="Змінити статус",
callback_data=f"change_{ans[0]}_{new_val}_{q_id}")]])
        await message.answer(f"👤 {ans[1]}\nВідповідь: {ans[2]}\nСтатус: {status}",
reply_markup=markup)

# Обробка зміни статусу відповіді (правильно <-> неправильно)
@router.callback_query(F.data.startswith("change_"))
async def change_mark_callback(callback: CallbackQuery):
    _, ans_id, new_status, q_id = callback.data.split("_")
    await db.update_answer_manual(int(ans_id), bool(int(new_status)))
    await callback.answer("Статус змінено")
    await callback.message.delete()


# --- ВИДАЛЕННЯ ГРУПИ ---

@router.message(F.text == "❌ Видалити групу", TeacherState.managing_groups)
async def delete_group_start(message: Message):
    groups = await db.get_all_groups()
    if not groups:
        return await message.answer("📂 Список груп порожній.")

    builder = ReplyKeyboardBuilder()
    for g in groups:
        builder.add(KeyboardButton(text=f"   Видалити {g}"))
```

```
    builder.add(KeyboardButton(text="🔙 Назад в меню"))
    builder.adjust(2)
    await message.answer("Оберіть групу:",
reply_markup=builder.as_markup(resize_keyboard=True))

@router.message(F.text.startswith("    Видалити "), TeacherState.managing_groups)
async def delete_group_finish(message: Message):
    group_name = message.text.replace("    Видалити ", "")
    if await db.delete_group(group_name):
        await message.answer(f"✅ Grupu <b>{group_name}</b> видалено!",
reply_markup=kb.groups_management_kb,
                             parse_mode="HTML")
    else:
        await message.answer("❌ Помилка.")
```

navigation.py

```
from aiogram import Router, F
from aiogram.types import Message
from aiogram.fsm.context import FSMContext
from aiogram.filters import Command, StateFilter

import config
import database.db_requests as db
import keyboards.reply as kb
from states.user_states import TeacherState

router = Router()

# ================= ГОЛОВНЕ МЕНЮ ТА НАВІГАЦІЯ
```

```
================

# Обробка команди /start — вхідна точка для викладача
@router.message(Command("start"), F.from_user.id.in_(config.ADMIN_IDS), StateFilter("*"))
async def cmd_start_admin(message: Message, state: FSMContext):
    await state.clear()  # Очищаємо будь-які старі стани, щоб не "застрягти" в меню
    await message.answer("👋 Вітаю, Викладачу! Оберіть розділ:", reply_markup=kb.teacher_main)
    await state.set_state(TeacherState.menu)  # Переводимо у стан головного меню

# Кнопка "Назад в меню" — скидає поточний стан і повертає головні кнопки викладача
@router.message(F.text == "🔙 Назад в меню", StateFilter("*"))
async def back_to_main(message: Message, state: FSMContext):
    await state.clear()  # Скидаємо підменю (напр. керування групами чи дисциплінами)
    await message.answer("Головне меню:", reply_markup=kb.teacher_main)
    await state.set_state(TeacherState.menu)

# Кнопка "Скасувати" — використовується під час налаштування питань, щоб відкотити дію назад
@router.message(F.text == "🔙 Скасувати", StateFilter("*"))
async def cancel_question_process(message: Message, state: FSMContext):
    await state.set_state(TeacherState.questions_menu)  # Повертаємо стан меню питань
    await message.answer("Скасовано.", reply_markup=kb.questions_sub)
```

user_handlers.py

```
from aiogram import Router, F
from aiogram.types import Message, ReplyKeyboardRemove
from aiogram.fsm.context import FSMContext
from aiogram.filters import CommandStart, StateFilter

import database.db_requests as db  # Робота з базою даних
import keyboards.builders as kb_build  # Динамічні клавіатури (варіанти відповідей)
import keyboards.reply as kb  # Статичні кнопки (головне меню студента)
from states.user_states import Registration, StudentState  # Стани FSM

router = Router()

# ================ РЕЄСТРАЦІЯ ТА СТАРТ ================

# Обробляється при першому запуску (/start) або при натисканні кнопки оновлення стану
@router.message(F.text == "    Оновити стан (Start)", StateFilter("*"))
@router.message(CommandStart(), StateFilter("*"))
async def cmd_start(message: Message, state: FSMContext):
    u_id = message.from_user.id
    user = await db.get_user(u_id) # Перевіряємо, чи є студент у базі
    await state.clear() # Скидаємо стани, щоб уникнути конфліктів при перезавантаженні

    # 1. ПЕРЕВІРКА РЕЄСТРАЦІЇ
    if not user:
        await message.answer(
            "👋 Вітаю! Ви не зареєстровані у системі квізів.\n\n"
```

```
            "Будь ласка, введіть ваше <b>Прізвище та Ім'я</b>:",
            reply_markup=ReplyKeyboardRemove(), # Прибираємо старі кнопки
            parse_mode="HTML"
        )
        await state.set_state(Registration.waiting_for_name) # Переходимо до збору даних
        return

    # 2. ПЕРЕВІРКА ПІДТВЕРДЖЕННЯ АДМІНОМ (Викладачем)
    if user[3] == 'pending':
        return await message.answer(
            "⏳ Ваша заявка ще розглядається викладачем.\n"
            "Натисніть кнопку оновлення пізніше.",
            reply_markup=kb.student_main
        )

    # 3. ПЕРЕВІРКА НАЯВНОСТІ АКТИВНОЇ ПАРИ
    lesson = await db.check_active_lesson()  # Шукаємо пару з end_time IS NULL
    if not lesson:
        return await message.answer("    Пара ще не почалася. Відпочивайте.", reply_markup=kb.student_main)

    # 4. ФІЛЬТР ЗА ГРУПОЮ (Ключовий момент безпеки)
    # Пара має бути розпочата саме для тієї групи, в якій навчається студент
    student_group = user[2]
    lesson_group = lesson[2]

    if student_group != lesson_group:
        return await message.answer(
```

```
            f"□□ Зараз іде пара для групи <b>{lesson_group}</b>.\n"
            f"Ваша група (<b>{student_group}</b>) зараз відпочиває.",
            reply_markup=kb.student_main,
            parse_mode="HTML"
        )

    # 5. ПЕРЕВІРКА НАЯВНОСТІ АКТИВНОГО ПИТАННЯ
    q = await db.get_active_question()
    if not q:
        return await message.answer("🎓 Пара триває, але активних питань зараз немає.", reply_markup=kb.student_main)

    # 6. ПЕРЕВІРКА, ЧИ СТУДЕНТ ВЖЕ ВІДПОВІДАВ
    # UNIQUE constraint у базі не дасть записати дублікат, але ми видаємо ввічливе попередження
    if await db.check_if_answered(u_id, q[0]):
        return await message.answer("✅ Ви вже відповіли на це питання. Очікуйте наступного.",
                                    reply_markup=kb.student_main)

    # 7. ВИДАЧА ПИТАННЯ СТУДЕНТУ
    # Якщо тип 'variant' — будуємо кнопки А, Б, В... Якщо 'text' — ховаємо кнопки для ручного введення
    markup = kb_build.get_variants_kb(q[3]) if q[2] == 'variant' else ReplyKeyboardRemove()
    await message.answer(
        f"❓ <b>ВІДКРИТО ПИТАННЯ №{q[0]}</b>\n\n"
        f"Тип: {q[2]}\n"
        f"Балів за правильну відповідь: {q[4]}",
```

```
        reply_markup=markup,
        parse_mode="HTML"
    )
    await state.update_data(active_question_id=q[0]) # Запам'ятовуємо ID питання в сесії
    await state.set_state(StudentState.answering) # Чекаємо на відповідь

# ================ ПРОЦЕС РЕЄСТРАЦІЇ ================

@router.message(Registration.waiting_for_name)
async def reg_name(message: Message, state: FSMContext):
    # Проста валідація довжини ПІБ
    if len(message.text) < 3:
        return await message.answer("⚠ Прізвище та ім'я занадто короткі. Введіть ПІБ повністю:")

    await state.update_data(full_name=message.text)
    all_groups = await db.get_all_groups()

    # Якщо груп ще немає в базі, просимо ввести назву текстом
    if not all_groups:
        await message.answer("⚠ Викладач ще не додав жодної групи. Введіть назву групи вручну:")
        await state.set_state(Registration.waiting_for_group)
    else:
        # Пропонуємо вибрати групу з існуючих (динамічна клавіатура)
        await message.answer(
            "Оберіть вашу групу зі списку:",
            reply_markup=kb_build.get_groups_reply_kb(all_groups)
```

```
        )
        await state.set_state(Registration.waiting_for_group)

@router.message(Registration.waiting_for_group)
async def reg_group(message: Message, state: FSMContext):
    data = await state.get_data()
    # Зберігаємо студента в базу зі статусом 'pending' (очікує підтвердження)
    await db.add_user(message.from_user.id, data.get('full_name'), message.text)

    await message.answer(
        "✅ <b>Реєстрація завершена!</b>\n\n"
        "Ваша заявка надіслана викладачу. Коли він її підтвердить, ви зможете брати участь у квізах.",
        reply_markup=kb.student_main,
        parse_mode="HTML"
    )
    await state.clear() # Очищуємо стан після успішної реєстрації

# ================ ВІДПОВІДЬ НА ПИТАННЯ ================

@router.message(StudentState.answering)
async def submit_answer(message: Message, state: FSMContext):
    # Перевірка на системні команди: якщо студент натиснув кнопки меню замість відповіді
    if message.text == "📊 Мій рейтинг":
        await state.clear()
        return await show_rating(message)
    if message.text == "🔄 Оновити стан (Start)":
        return await cmd_start(message, state)
```

```
    data = await state.get_data()
    q_id = data.get('active_question_id')
    active_q = await db.get_active_question()

    # ПЕРЕВІРКА АКТУАЛЬНОСТІ: Чи не закрив викладач питання, поки студент думав?
    if not active_q or active_q[0] != q_id:
        await message.answer("⏰ Час вийшов! Викладач вже закрив це питання.", reply_markup=kb.student_main)
    else:
        # Зберігаємо відповідь. Перевірка правильності відбудеться автоматично при закритті питання викладачем.
        await db.save_answer(message.from_user.id, q_id, message.text)
        await message.answer("✅ Відповідь прийнята! Чекайте на результати.", reply_markup=kb.student_main)

    await state.clear()

# ================= РЕЙТИНГ ================

@router.message(F.text == "📊 Мій рейтинг", StateFilter("*"))
async def show_rating(message: Message):
    u_id = message.from_user.id
    user = await db.get_user(u_id)

    # Тільки для зареєстрованих та схвалених
    if not user or user[3] != 'approved':
        return await message.answer("❌ Рейтинг доступний тільки зареєстрованим
```

студентам.")

```
    # Отримуємо різні типи балів з БД
    total = await db.get_student_total_score(u_id)
    lesson = await db.check_active_lesson()

    text = f"🏆 <b>Ваш рейтинг:</b>\n"
    text += f"👤 Студент: {user[1]}\n"
    text += f"⭐ Загальний бал: <b>{total}</b>\n"
    text += "----------------------"

    # Якщо зараз триває пара — показуємо деталізацію за предмет та поточне заняття
    if lesson:
        # lesson[1] - discipline_id, lesson[0] - lesson_id
        d_score = await db.get_student_discipline_score(u_id, lesson[1])
        l_score = await db.get_student_lesson_score(u_id, lesson[0])
        text += f"\n📚 За поточну дисципліну: {d_score}"
        text += f"\n🔹 За поточну пару: {l_score}"

    await message.answer(text, parse_mode="HTML", reply_markup=kb.student_main)
```

builders.py

```
from aiogram.utils.keyboard import InlineKeyboardBuilder, ReplyKeyboardBuilder
from aiogram.types import InlineKeyboardButton, KeyboardButton

# ================= 1. КЛАВІАТУРА ВАРІАНТІВ ВІДПОВІДЕЙ ================
```

```
# Використовується студентами для вибору варіанту (А, Б, В...) у тестових питаннях
def get_variants_kb(count: int):
    """
    Створює кнопки з літерами-варіантами для студентів.
    :param count: Кількість варіантів, яку вказав викладач при створенні питання.
    """
    builder = ReplyKeyboardBuilder()  # Створюємо конструктор звичайної клавіатури

    # Використовуємо український алфавіт для маркування кнопок
    variants = ['А', 'Б', 'В', 'Г', 'Ґ', 'Д', 'Е', 'Є', 'Ж']

    # limit гарантує, що ми не вийдемо за межі нашого списку літер
    limit = min(count, len(variants))

    # Циклом додаємо кожну літеру як окрему кнопку
    for i in range(limit):
        builder.add(KeyboardButton(text=variants[i]))

    # adjust(4) автоматично розставляє кнопки по 4 штуки в кожному ряду
    builder.adjust(4)

    # as_markup перетворює конструктор у готову для Telegram клавіатуру
    return builder.as_markup(resize_keyboard=True)

# ================= 2. КЛАВІАТУРА ВИБОРУ ДИСЦИПЛІНИ (INLINE) ================
# Використовується викладачем для швидкого вибору предмета при старті пари
```

```
def get_disciplines_kb(disciplines):
    """
    Створює інлайн-кнопки (кнопки під повідомленням).
    :param disciplines: Список кортежів з БД [(id, назва, група), ...]
    """
    builder = InlineKeyboardBuilder()  # Створюємо конструктор інлайн-клавіатури

    for d in disciplines:
        # Формуємо текст кнопки: Назва предмету + назва групи у дужках
        btn_text = d[1]
        if len(d) > 2:
            btn_text = f"{d[1]} ({d[2]})"

        # callback_data — це прихована команда, яку отримає бот при натисканні.
        # Вона містить ID дисципліни, щоб ми знали, яку саме пару починати.
        builder.add(InlineKeyboardButton(
            text=btn_text,
            callback_data=f"start_lesson_{d[0]}"
        ))

    builder.adjust(1)  # Кожна дисципліна займає окремий рядок для зручності натискання
    return builder.as_markup()

# ================= 3. КЛАВІАТУРА ВИБОРУ ГРУПИ (REPLY) ================
# Використовується студентами при реєстрації та викладачем при створенні предмета
def get_groups_reply_kb(groups_list):
    """
```

```
    Створює звичайні кнопки з назвами груп.
    :param groups_list: Список назв груп прямо з бази даних.
    """
    builder = ReplyKeyboardBuilder()

    # Якщо груп у базі ще немає (наприклад, перший запуск), повертаємо
порожнечу
    if not groups_list:
        return None

    # Додаємо кожну назву групи як кнопку
    for group in groups_list:
        builder.add(KeyboardButton(text=str(group)))

    builder.adjust(2)  # Розміщуємо по 2 групи в ряд для економії місця на екрані
    return builder.as_markup(resize_keyboard=True)
```

reply.py

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

# ================= ГОЛОВНЕ МЕНЮ ВИКЛАДАЧА
================
# З'являється після команди /start, якщо ID користувача в списку ADMIN_IDS
teacher_main = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="    Студенти"), KeyboardButton(text="📚 Дисципліни")],
    [KeyboardButton(text="💼 Керування групами"), KeyboardButton(text=" ❓
Питання")]
], resize_keyboard=True) # resize_keyboard=True робить кнопки компактними, а
не на весь екран
```

```
# ================ ПІДМЕНЮ "ДИСЦИПЛІНИ" ================
# Тут зосереджено все керування предметами та заняттями (парами)
disciplines_sub = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="➕ Додати дисципліну"), KeyboardButton(text="📚 Список дисциплін")],
    [KeyboardButton(text="🎓 Почати пару"), KeyboardButton(text="  Закінчити пару")],
    [KeyboardButton(text="🔒 Закрити дисципліну (Семестр)")], # Переведення предмета в архів
    [KeyboardButton(text="🔙 Назад в меню")]
], resize_keyboard=True)

# ================ ПІДМЕНЮ "ПИТАННЯ" ================
# Керування активністю під час проведення пари
questions_sub = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="📢 Відкрити питання"), KeyboardButton(text="❌ Закрити питання")],
    [KeyboardButton(text="📝 Перевірка відповідей")], # Для перегляду того, як відповіли студенти
    [KeyboardButton(text="🔙 Назад в меню")]
], resize_keyboard=True)

# ================ ПІДМЕНЮ "СТУДЕНТИ" ================
# Робота з базою користувачів та вивантаження звітів
students_sub = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="✅ Підтвердити реєстрацію"), KeyboardButton(text="👥 Список студентів")],
    [KeyboardButton(text="📊 Експорт Excel")], # Генерація та надсилання .xlsx
```

```
файлу
    [KeyboardButton(text="🔙 Назад в меню")]
], resize_keyboard=True)

# ================= ВИБІР ТИПУ ПИТАННЯ =================
# З'являється після натискання "📢 Відкрити питання"
question_type_kb = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="🔠 Варіанти"), KeyboardButton(text="📝 Текст")], # Тест або вільна відповідь
    [KeyboardButton(text="🔙 Скасувати")]
], resize_keyboard=True)

# ================= ПІДТВЕРДЖЕННЯ РОЗСИЛКИ =================
# З'являється після закриття питання викладачем
yes_no_kb = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="  □ Показати студентам"), KeyboardButton(text="🙈 Не показувати")]
], resize_keyboard=True)

# ================= ГОЛОВНЕ МЕНЮ СТУДЕНТА =================
# Мінімалістичне меню для перевірки активних завдань та своїх балів
student_main = ReplyKeyboardMarkup(keyboard=[
    [KeyboardButton(text="    Оновити стан (Start)"), KeyboardButton(text="📊 Мій рейтинг")]
], resize_keyboard=True)

# ================= КЕРУВАННЯ ГРУПАМИ =================
# Дозволяє створювати нові академічні групи або видаляти існуючі
groups_management_kb = ReplyKeyboardMarkup(
```

```
    keyboard=[
        [KeyboardButton(text="➕ Додати групу"), KeyboardButton(text="❌
Видалити групу")],
        [KeyboardButton(text="🔙 Назад в меню")]
    ], resize_keyboard=True
)
```

user_states.py

```
from aiogram.fsm.state import State, StatesGroup

# ================= СТАННИ РЕЄСТРАЦІЇ =================
class Registration(StatesGroup):
    """
    Група станів для нових користувачів.
    Допомагає боту послідовно зібрати дані при першому вході.
    """
    waiting_for_name = State()   # Бот чекає на введення Прізвища та Імені
    waiting_for_group = State()  # Бот чекає на вибір або введення назви групи

# ================= СТАНИ СТУДЕНТА =================
class StudentState(StatesGroup):
    """
    Стани, в яких може перебувати студент під час навчання.
    """
    answering = State()  # Студент отримав питання і бот чекає на його відповідь

# ================= СТАНИ ВИКЛАДАЧА (АДМІНІСТРАТОРА)
================
class TeacherState(StatesGroup):
    """
```

```
    Велика група станів для керування всіма процесами бота.
    Розбита на логічні блоки для зручної навігації.
    """
    menu = State()  # Головне адмін-меню

    # --- Блок Дисциплін ---
    disciplines_menu = State()  # Меню керування предметами
    create_discipline = State() # Загальний стан створення (може бути застарілим)
    create_discipline_name = State()  # Очікування назви нового предмета
    create_discipline_group = State() # Очікування вибору групи для цього
предмета

    # --- Блок Питань (Керування парою) ---
    questions_menu = State()            # Меню керування питаннями під час пари
    setting_question = State()          # Введення параметрів (варіанти, бали) для
нового питання
    waiting_for_correct_answer = State() # Очікування еталонної відповіді від
викладача
    confirm_visibility = State()        # Очікування рішення: показувати результати
студентам чи ні

    # --- Блок Студентів та Груп ---
    students_menu = State()   # Меню перегляду списків та експорту результатів
    managing_groups = State() # Меню керування академічними групами
    adding_group = State()    # Очікування назви нової групи (напр. "КН-21")
```

excel_export.py

```
import openpyxl  # Бібліотека для створення та редагування файлів Excel
from io import BytesIO  # Модуль для роботи з потоками байтів (пам'яттю)
```

```
async def create_excel_report(data):
    """
    Генерує Excel-файл на основі даних із бази даних.
    :param data: Список кортежів (результатів), отриманих через db.get_all_results()
    :return: BytesIO об'єкт (файл у пам'яті)
    """
    # 1. Створюємо нову робочу книгу Excel
    wb = openpyxl.Workbook()
    # Активуємо перший аркуш
    ws = wb.active
    # Задаємо назву вкладці внизу файлу
    ws.title = "Успішність"

    # 2. ФОРМУЄМО ЗАГОЛОВКИ (Перший рядок таблиці)
    headers = ["ПІБ Студента", "Група", "Дисципліна", "Дата пари", "Тип питання", "Отриманий бал"]
    ws.append(headers)  # Додаємо список як новий рядок у таблицю

    # 3. ЗАПИСУЄМО ДАНІ
    # Цикл проходить по кожному рядку результатів, який ми витягли з SQL-запиту
    for row in data:
        # row містить: (full_name, group_name, discipline, date, q_type, score)
        # Перетворюємо кортеж у список і додаємо в Excel
        ws.append(list(row))

    # 4. АВТОМАТИЧНЕ ПІДЛАШТУВАННЯ ШИРИНИ СТОВПЦІВ
    # Це робить звіт гарним: стовпці розширюються під довжину тексту (наприклад, довгі ПІБ)
```

```
    for column_cells in ws.columns:
        # Знаходимо найдовше значення в поточному стовпці
        length = max(len(str(cell.value) or "") for cell in column_cells)
        # Встановлюємо ширину стовпця з невеликим запасом (+2)
        ws.column_dimensions[column_cells[0].column_letter].width = length + 2

    # 5. ЗБЕРЕЖЕННЯ В БУФЕР (BytesIO)
    # Замість того, щоб зберігати файл на жорсткий диск сервера, ми тримаємо його в оперативній пам'яті.
    # Це швидше і безпечніше (не накопичуються зайві файли).
    bio = BytesIO()
    wb.save(bio)  # Записуємо Excel-книгу в об'єкт bio
    bio.seek(0)  # Переводимо "курсор" читання на початок файлу для подальшої відправки в Telegram

    return bio  # Повертаємо готовий "файл у пам'яті"
```

.env

```
BOT_TOKEN=TOKEN bot
ADMIN_IDS=ID admin
```

config.py

```
import os
from dotenv import load_dotenv

load_dotenv()

# Bot token from @BotFather
BOT_TOKEN = os.environ.get("BOT_TOKEN", "")
```

```
# Admin Telegram IDs (comma-separated in env, e.g. "647694431,123456789")
_raw_admin_ids = os.environ.get("ADMIN_IDS", "")
ADMIN_IDS = [int(x.strip()) for x in _raw_admin_ids.split(",") if x.strip().isdigit()]
```

requirements.txt

```
aiogram==3.4.1
aiosqlite==0.20.0
openpyxl==3.1.2
python-dotenv==1.0.1
pandas==2.2.1
```